rasdaman documentation

Release 10.1.0

rasdaman team

Mar 15, 2023

CONTENTS

This is the documentation of *rasdaman community*, a multi-dimensional Array Analytics Engine that enables efficient querying and manipulation of multi-dimensional arrays of unlimited size.

Contents:

CHAPTER

ONE

INTRODUCTION

The goodies of database technology for building flexible, large-scale information systems are well known:

- **information integration** bringing along better consistency and much easier administration
- **flexibility** by replacing static APIs by dynamic query languages
- **scalability** achieved through advanced storage and processing techniques, in particular: query optimization which is a powerful performance booster
- **maturity** of decades of development and functionality richness, as opposed to 1.0 versions of reinvented wheels (load balancing, indexing, transaction handling, catalog metadata management, . . .)

Unfortunately, these advantages until now can be reaped only for alphanumeric (and, more recently, also vectorial) data types. Raster data (also called gridded data, sampled data, etc.) do not benefit, since traditional databases do not support the information category of large, multidimensional arrays.

This gap is closed by the rasdaman technology which offers distinct array management features. Its conceptual model supports arrays of any number of dimensions and over virtually any cell ("pixel", "voxel") type. The rasdaman query language, rasql, is crafted along standard SQL and gives high-level, declarative access to any kind of raster data. Its architecture principle of **tile stream processing**, together with highly effective optimizations, has proven scalable into multi-Terabyte object sizes. Rasdaman has been developed since 1996 and since has reached a high level of maturity itself being in operational use since many years.

The rasdaman project strongly commits itself to open standards, in particular those of the geo service community. We actively participate in the development and maintenance of the Open Geospatial Consortium open geo raster standards. Among other activities, we have developed specification and reference implementation of OGC WCS 2.0, WCPS, and WPS.

The free and open-source **rasdaman community** version is available for free download; generally, rasdaman is available in a dual [wiki:License license model]. If the **scientific background** of rasdaman is of interest, then check out our Publications . . . and cite our papers!

Finally, for the **legalese** see Imprint and Disclaimer.

1.1 Features

Technically, rasdaman is a domain independent Array DBMS, which makes it suitable for all applications where raster data management is an issue. The petascope component of rasdaman adds on geo semantics for example, with full support for the OGC standard interfaces WCS, WCPS, WCS-T, and WMS; see matrix below for details and EarthLook for a kaleidoscope of hands-on interactive demos.

Historically, rasdaman has pioneered the field of Array Databases, being the first system of this kind. The rasdaman technology has been developed over a series of EU funded projects and then marketed by rasdaman GmbH, a research spin-off dedicated to its commercial support, since 2003. In 2008/2009, the company has teamed up with Jacobs University for a code split to establish *rasdaman community* (encompassing a complete Array DBMS) as an open-source project. The original rasdaman code remains as *rasdaman enterprise*. Both are kept in sync at any time, and both rasdaman GmbH and Jacobs University contribute actively to the open-source project. The features covered here concern the open-source community version; a summary of *rasdaman community* versus *rasdaman enterprise* was presented here earlier, but has been meanwhile abandoned as OSGeo frowned on this.

Contributions to the rasdaman community code come from a worldwide team of collaborators. Notably, a significant extent of the fixes and new functionality is coming from rasdaman GmbH (such as WMS recently). **All community contributions submitted are made available in rasdaman community immediately after checking them for correctness and coherence (eg, with the code guide); no contribution whatsoever goes into rasdaman enterprise first**. The only action the company undertakes is to keep both rasdaman variants in sync by merging the rasdaman community tree into rasdaman enterprise, which typically occurs upon release of versions so as to keep both in sync. Aside from that, rasdaman enterprise is developed exclusively by the company and does not contain any community code that rasdaman community does not contain. So rest assured that your valuable contributions are to the benefit of the worldwide user community.

1.1.1 Array data model

Arrays are determined by their extent ("domain") and their cell ("pixel", "voxel"). Extents are given by a lower and upper bound taken from the integer domain (so negative boundaries are possible as long as the lower bound remains below the upper bound). For the cells, all base and composite data types allowed in languages like C/C++ (except for pointers and arrays) can be defined as cell types, including nested structs.

Over such typed arrays, collections (ie, tables - ODMG style, again) are built. Collections have two columns (attributes), a system-maintained object identifier (OID) and the array itself. This allows to conveniently embed arrays into relational modeling: foreign keys in conventional tables allow to reference particular array objects, in connection with a domain specification even parts of arrays.

As such, rasdaman is prepared for the forthcoming ISO SQL/MDA ("Multi-Dimensional Arrays") standard, which actually is crafted along rasdaman array model and query language. This standard will define arrays as new attribute types, following an "array-as-an-attribute" approach for optimal integration with relations (as opposed to an “attribute-as-table” approach - as pursued, e.g., by SciDB and SciQL - which has some remarkable shortcomings in practice).

1.1.2 Query language

The rasdaman query language, rasql, offers raster processing formulated through expressions over raster operations in the style of SQL. Consider the following query: “*The difference of red and green channel from all images from collection LandsatImages where somewhere in the red channel intensity exceeds 127*”. In rasql, it is expressed as

```
select ls.red - ls.green
from LandsatImages as ls
where max_cells( ls.red ) > 127
```

Rasql is a full query language, supporting *select*, *insert*, *update*, and *delete*. Additionally, the concept of a partial update is introduced which allows to selectively update parts of an array. In view of the potentially large size of arrays this is a practically very relevant feature, e.g., for updating satellite image maps with new incoming imagery.

Query formulation is done in a declarative style (queries express what the result should look like, not how to compute it). This allows for extensive optimization on server side. Further, rasql is safe in evaluation: every valid query is guaranteed to terminate in finite time.

1.1.3 C++ and Java API

Client development is supported by the C++ API, *raslib*, and the Java API, *rasj*; both adhere to the ODMG standard. Communication with a rasdaman database is simple: open a connection, send the query string, receive the result set. Iterators allow convenient acecss to query results.

Once installed, go into the share/rasdaman/examples subdirectory to find sample code.

1.1.4 Tiled storage

On server side, arrays are stored inside a standard database. To this end, arrays are partitioned into subarrays called *tiles*; each such tile goes into a BLOB (binary large object) in a relational table. This allows conventional relational database systems to maintain arrays of unlimited size.

A spatial index allows to quickly locate the tiles required for determining the tile set addressed by a query.

The partitioning scheme is open - any kind of tiling can be specified during array instantiation. A set of tiling strategies is provided to ease administrators in picking the most efficient tiling.

1.1.5 Tile streaming

Query evaluation in the server follows the principle of *tile streaming*. Each operator node processes a set of incoming tiles and generates an output tile stream itself. In many cases this allows to keep only one database tile at a time in main memory. Query processing becomes very efficient even on low-end server machines.

1.1.6 Server multiplexing

A rasdaman server installation can consist of an arbitrary number of rasdaman server processes. A dynamic scheduler, *rasmgr*, receives incoming connection requests and assigns a free server process. This server process then is dedicated to the particular client until the connection is closed. This allows for highly concurrent access and, at the same time, increases overall safety as clients are isolated against each other.

1.2 Rasdaman Application Domains

Its features make rasdaman suitable for all applications where raster data management is an issue, such as:

earth sciences

1-D sensor time series; 2-D airborne/satellite image maps; 3-D satellite image time series; 3-D geo tomograms; 4-D climate and ocean data; … At EarthLook there is a demonstration of services on 1-D to 4-D geo raster objects. The workhorse of the service stack is rasdaman, running on top of PostgreSQL.

space sciences

2-D visibility maps; x/y/frequency observation data cubes; 4-D cosmological simulation data; …

life sciences

3-D brain activation maps; 3-D/4-D gene expression maps; …

engineering

1-D measurement time series; 3-D/4-D simulation result data; …

multimedia

1-D audio; 2-D imagery; 3-D movies; …

See the publication list for descriptions of a variety of projects where rasdaman has been successfully used.

1.3 OGC geo standards support

While rasdaman itself is domain agnostic and supports any array application, the *petascope* servlet, as part of rasdaman, adds in geo semantics, such as dealing with geo coordinates. To this end, rasdaman implements the Open Geospatial Consortium standards for gridded coverages, i.e., multi-dimensional raster data. The OGC service interfaces supported are - Web Coverage Service: a versatile, modular suite for accessing and server-side processing of coverages, - Web Coverage Processing Service: OGC's Big Datacube Analytics language, - Web Map Service: for rendering coverage data into maps which can be displayed with a wide range of open-source and commercial clients.

The Princial Architect of rasdaman, Peter Baumann, is chair of the OGC WCS Standards Working Group (WCS.SWG) and editor of coverage model (GMLCOV), WCPS, and most of the WCS specifications, rasdaman naturally has become Reference Implementation for several of these standards and usually implements them first and way ahead of other systems, even before final adoption. Likewise, any changes to coverage-related specifications usually are verified in rasdaman first and, hence, become available early. The same holds for the OGC conformance testing of coverage services where rasdaman code contributors have a lead. In summary, rasdaman can be considered the most comprehensive and best tested implementation of the OGC coverage standards.

1.4 How to Contribute

There are lots of ways to get involved and help out with the rasdaman project:

Help us spot & fix bugs.

Which software is perfect? We know there are some bugs in rasdaman, see the open tickets (or the low complexity tickets for beginners. Whether you add a ticket or provide a fix, all is most welcome.

Write documentation.

Users can always benefit from better documentation. Currently the documentation is in reStructuredText format, and HTML/PDF is automatically generated. We're eager for any documentation contributions.

Contribute to the Wiki.

Of course you can also contribute to the wiki, for example by adding HowTos and FAQs. Send a message with a change request to *patch* in the domain *rasdaman.org*.

Help plan and design the next version.

Browse this section of the website, we use "Feature" tickets to hold ideas for new features; add your own and/or discuss a topic on the dev list.

1.5 Reporting problems

Reporting problems should be done by sending an email to the rasdaman-users mailing list. Your email should include a report with *relevant information* about the issue, either prepared with help of the *prepare_issue_report.sh script*, or manually specified as bellow:

1. OS distribution and version (see /etc/os-release)
2. Rasdaman version

- Ubuntu: `apt-cache show rasdaman`
- CentOS: `yum info rasdaman`

3. Concise description of your activity that led to the problematic behavior: queries, package management commands, etc.
4. Properties of the data operated on, including (but not limited to) data format, pixel type, coordinate reference system, dimension, along with data ingestion details (ingredients files, scripts); include a small resized data sample if possible, output of WCS DescribeCoverage of affected coverages, dbinfo on the underlying collections and spatial domain:

```
coverageId=???
ows_endpoint="http://localhost:8080/rasdaman/ows"
describe_cov_req="service=WCS&version=2.0.1&
↪request=DescribeCoverage&coverageId=${coverageId}"

curl "${ows_endpoint}?${describe_cov_req}" > DescribeCoverage.
↪xml
rasql -q "select dbinfo(c) from $coverageId as c" --out string >
↪ dbinfo.json
rasql -q "select sdom(c) from $coverageId as c" --out string >␣
↪sdom.txt
tar cfz /tmp/data_details.tar.gz DescribeCoverage.xml dbinfo.
↪json sdom.txt
```

   Attach `/tmp/data_details.tar.gz` to the report, along with the ingredient files. Sample data should be uploaded elsewhere, e.g. Google Drive, if it is larger than 20 MB.

5. Relevant log files in `/opt/rasdaman/log` and `/var/log/tomcat*/`; you can compress the last 20 log files as follows (but try to execute the problematic query / operation last, just before the step below):

```
cd /opt/rasdaman/log
petascope_log="$(sudo find /var/log/ -name petascope.log)"
tar cfz /tmp/rasdaman_logs.tar.gz $(ls -t | head -n 20)
↪$petascope_log
```

   Attach `/tmp/rasdaman_logs.tar.gz` to the report.

   Prior to sending your request, you should inspect the log files, they may already provide a clue that helps you resolve the issue.

1.5.1 Script for issue reporting

Rasdaman distributes with a `prepare_issue_report.sh` script in `/opt/rasdaman/bin`, which helps prepare a report for an issue encountered while operating rasdaman. Running the script will open an editor where you can enter a description of how the issue got triggered.

Various options can be specified to control what additional information is included in order to help developers in understanding and reproducing the issue. Following the options, you can specify files to include in the report, e.g. screenshots, ingredient files for importing data, sample (downsized if possible) data, etc.

Everything is compressed into a single archive in the current working directory from which the script is executed, and the path to it is printed at the end.

By default the script will try to include config files, latest 200 log files, petascopedb, and RASBASE, as long as the resulting archive is not larger than 20 MB to make it suitable for sending by email. Parts which are too large will be left out, in reverse order of priority (first RASBASE, then petascopedb, etc). The limit can be changed with –limit-size <N>. As soon as a particular –include-* option is specified, the default behavior is no longer in effect and exclusively the specified options are considered.

Check `prepare_issue_report.sh --help` for a list of all available options.

Examples

1. Describe the issue, including config files and 100 most recent log files, as well as a screenshot illustrating the problem:

```
$ prepare_issue_report.sh --include-recent-logs 100 -f␣
↪screenshot.png \
                          --no-coverage-id
```

2. Describe the issue, include config files, all log files, petascopedb and RASBASE, as well as sample data and ingredients:

```
$ prepare_issue_report.sh --include-all-logs --include-
↪petascopedb \
                          --include-rasbase -f sample_data.tar.
↪gz \
                          -f ingredients.json --no-coverage-id
```

3. Like the first example, but also include information about coverage TestCov:

```
$ prepare_issue_report.sh --include-recent-logs 100 --coverage-
↪id TestCov \
                          -f screenshot.png
```

4. Provide a screenshot and include details up to a maximum archive size of 20 MB (default behavior):

```
$ prepare_issue_report.sh -f screenshot.png --no-coverage-id
```

CHAPTER

TWO

INSTALLATION AND ADMINISTRATION GUIDE

2.1 Preface

2.1.1 Overview

This guide provides information about how to use the rasdaman array database system, in particular: installation and system administration.

For storage of multi-dimensional array data, rasdaman can be configured to use some conventional database system (such as PostgreSQL) or use its own storage manager. For the purpose of this documentation, we will call the conventional database system to which rasdaman is interfaced the *base DBMS*, understanding that this base DBMS is in charge of all alphanumeric data maintained as relational tables or object-oriented semantic nets.

This guide is specific for *rasdaman community*; for *rasdaman enterprise* (what's the difference?) contact rasdaman GmbH.

2.1.2 Audience

The information in this manual is intended primarily for database and system administrators.

2.1.3 Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as *raswct*, release notes, and additional information on the rasdaman wiki.

2.2 Getting Started

There are several ways to ride rasdaman - installing from source, installing prefabricated RPM/DEB packages, or downloading a preconfectioned Virtual Machine. Make your choice below!

Hardware & Software Requirements

It is recommended to have at least 8 GB main memory. Disk space depends on the size of the databases, as well as the requirements of the base DBMS of rasdaman chosen. The footprint of the rasdaman installation itself is around 400 MB.

In order to download, build, and run rasdaman, various tools and libraries are required. This varies depending on the mode of installation (*Official Packages* or *Build From Source Manually*). Some packages, such as HDF4, are optional. This means that the feature (e.g. support for the HDF4 data format) is not available unless its use is specified during configuration. See *Download and Install rasdaman* for more information on the cmake configure parameters.

Rasdaman is continuously tested on the platforms listed below. The rasdaman code has been developed on SUN/Solaris and HP-UX originally, and has been ported to IBM AIX, SGI IRIX, and DEC Unix - but that was way back in the last millennium.

- Ubuntu 18.04, 20.04, 22.04
- CentOS 7

In general, compiling rasdaman should work on distributions with gcc 4.8 or later and Java 8 or later.

Alternative 1: Packages

Get preconfectioned packages for installing *RPM-based systems* on CentOS or *Debian-based systems* on Debian / Ubuntu; this is the recommended way - among others because the package manager will be able to manage your installation.

Alternative 2: Guided Build

Download and compile rasdaman with the help of an *automated installer*. On supported operating systems this option works automatically out of the box, but also allows to easily adjust the build and install procedures by editing a configuration file. As such it is mainly aimed at non-developers who would like to customize their rasdaman installation to something different than the official packages.

Alternative 3: Source Code

Download and compile rasdaman; this is the most flexible alternative; however, it requires some experience in manual compilation and is generally done by developers who plan to contribute code to the rasdaman repository.

Alternative 4: Virtual Machine

By *downloading a Virtual Machine* you get a fully configured system with rasdaman installed and ready to run. This alternative does not require any system administration skills other than starting the VM and working with the rasdaman services, e.g., via the OGC standards based geo service interface.

Support

Installation information, FAQs, and troubleshooting information is available on www.rasdaman.org.

For support in installing rasdaman and any other question you may contact rasdaman GmbH at www.rasdaman.com.

2.2.1 Official Packages

This page describes installation of rasdaman RPM or Debian packages.

During generation of these packages, some configuration decisions have been made (which can be chosen freely when *compiling from source*). Most importantly, the rasdaman engine in the packages uses embedded SQLite for managing its array metadata. Notice, though, that the geo service component, petascope, currently still relies on a PostgreSQL database; this is planned to be changed in the near future.

Debian-based systems

Currently the following Debian-based distributions are supported:

- Ubuntu 18.04 / 20.04 / 22.04

Installation

1. Import the rasdaman repository public key to the apt keychain:

```
$ wget -O - https://download.rasdaman.org/packages/rasdaman.gpg␣
↪| sudo apt-key add -
```

Note: You may need to update the ca-certificates package to allow SSL-based applications (e.g. `apt-get update` or `wget/curl`) to check for the authenticity of SSL connections:

```
$ sudo apt-get install ca-certificates
```

2. Add the rasdaman repository to apt. There are three types of packages:

   - **stable:** these packages are only updated on stable releases of rasdaman, and hence recommended for operational production installations.

```
# For ubuntu 22.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb jammy stable" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list
```

(continues on next page)

(continued from previous page)

```
# For ubuntu 20.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb focal stable" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list

# For ubuntu 18.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb bionic stable" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list
```

- **testing:** updated more frequently with beta releases, so aimed for feature testing in non-critical installations.

```
# For ubuntu 22.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb jammy testing" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list

# For ubuntu 20.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb focal testing" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list

# For ubuntu 18.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb bionic testing" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list
```

- **nightly:** updated nightly, so that they have the latest patches. It is not recommended to use these packages in a production installation as things could sometimes break.

```
# For ubuntu 22.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb jammy nightly" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list

# For ubuntu 20.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb focal nightly" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list

# For ubuntu 18.04
$ echo "deb [arch=amd64] https://download.rasdaman.org/
↪packages/deb bionic nightly" \
| sudo tee /etc/apt/sources.list.d/rasdaman.list
```

3. rasdaman can be installed now:

```
$ sudo apt-get update
$ sudo apt-get install rasdaman
```

If during the install you get a prompt like the below, type **N** (default option):

```
Configuration file `/etc/opt/rasdaman/petascope.properties'
 ==> Modified (by you or by a script) since installation.
 ==> Package distributor has shipped an updated version.
   What would you like to do about it ?  Your options are:
    Y or I  : install the package maintainer's version
    N or O  : keep your currently-installed version
      D     : show the differences between the versions
      Z     : start a shell to examine the situation
 The default action is to keep your current version.
*** petascope.properties (Y/I/N/O/D/Z) [default=N] ?
```

If you are automating the installation (in a script for example), you can bypass this prompt with an apt-get option as follows:

```
$ apt-get -o Dpkg::Options::="--force-confdef" install -y␣
↪rasdaman
```

You will find the rasdaman installation under `/opt/rasdaman/`. Finally, to make rasql available on the PATH for your system user:

```
$ source /etc/profile.d/rasdaman.sh
```

5. Check that the rasdaman server can answer queries:

```
$ rasql -q 'select c from RAS_COLLECTIONNAMES as c' --out string
```

Typical output:

```
rasql: rasdaman query tool v1.0, rasdaman v10.0.0 -- generated␣
↪on 26.02.2020 08:44:56.
opening database RASBASE at localhost:7001...ok
Executing retrieval query...ok
Query result collection has 0 element(s):
rasql done.
```

6. Check that petascope is initialized properly, typically at this URL:

```
http://localhost:8080/rasdaman/ows
```

7. If SELinux is running then possibly some extra configuration is needed to get petascope run properly. See *here* for more details.

Updating

The packages are updated whenever a new rasdaman version is released. To update your installation:

```
$ sudo apt-get update
$ sudo service rasdaman stop
$ sudo apt-get install rasdaman
```

Note: You may need to update the ca-certificates package to allow SSL-based applications (e.g. `yum update` or `wget/curl`) to check for the authenticity of SSL connections:

```
$ sudo apt-get install ca-certificates
```

RPM-based systems

Currently the following RPM-based distributions are supported:

- CentOS 7

Installation

1. Add the rasdaman repository to yum. There are three types of packages:

   - **stable:** these packages are only updated on stable releases of rasdaman, and hence recommended for operational production installations.

     ```
     $ sudo curl "https://download.rasdaman.org/packages/rpm/
     ↪stable/CentOS/7/x86_64/rasdaman.repo" \
               -o /etc/yum.repos.d/rasdaman.repo
     ```

   - **testing:** updated more frequently with beta releases, so aimed for feature testing in non-critical installations.

     ```
     $ sudo curl "https://download.rasdaman.org/packages/rpm/
     ↪testing/CentOS/7/x86_64/rasdaman.repo" \
               -o /etc/yum.repos.d/rasdaman.repo
     ```

   - **nightly:** updated nightly, so that they have the latest patches. It is not recommended to use these packages in a production installation as things could sometimes break.

     ```
     $ sudo curl "https://download.rasdaman.org/packages/rpm/
     ↪nightly/CentOS/7/x86_64/rasdaman.repo" \
               -o /etc/yum.repos.d/rasdaman.repo
     ```

Note: You may need to update the ca-certificates package to allow SSL-based applications (e.g. `yum update` or `wget/curl`) to check for the authenticity of SSL connections:

```
$ sudo yum install -y ca-certificates
```

2. The rasdaman packages should be available now via yum:

```
$ sudo yum clean all
$ sudo yum update
$ sudo yum search rasdaman
```

Output:

```
rasdaman.x86_64 : Rasdaman extends standard relational database␣
↪systems with the ability
                to store and retrieve multi-dimensional␣
↪raster data
```

3. Add the EPEL repository to yum (official page), needed for several dependencies of the rasdaman package:

```
$ sudo yum install epel-release
```

4. Install the rasdaman package:

```
$ sudo yum install rasdaman
```

You will find the rasdaman installation under `/opt/rasdaman/`. To make rasql available on the PATH for your system user:

```
$ source /etc/profile.d/rasdaman.sh
```

Note: If petascope has *problems* connecting to rasdaman, check this FAQ entry for some advice.

5. Check that the rasdaman server can answer queries:

```
$ rasql -q 'select c from RAS_COLLECTIONNAMES as c' --out string
```

Typical output:

```
rasql: rasdaman query tool v1.0, rasdaman v10.0.0 -- generated␣
↪on 26.02.2020 08:44:56.
opening database RASBASE at localhost:7001...ok
Executing retrieval query...ok
```

(continues on next page)

(continued from previous page)

```
Query result collection has 0 element(s):
rasql done.
```

6. Check that petascope is initialized properly, typically at this URL:

```
http://localhost:8080/rasdaman/ows
```

7. If SELinux is running then likely some extra configuration is needed to get petascope run properly. See *here* for more details.

Updating

The packages are updated whenever a new version of rasdaman is released. To download an update perform these steps:

```
$ sudo yum clean all
$ sudo service rasdaman stop
$ sudo yum update rasdaman
```

Note: You may need to update the ca-certificates package to allow SSL-based applications (e.g. `yum update` or `wget/curl`) to check for the authenticity of SSL connections:

```
$ sudo yum install -y ca-certificates
```

Customizing the package installation

When installing or updating rasdaman from the official packages, the process can be optionally customized with an installation profile (see example installer configuration).

- To customize when installing rasdaman for the first time, it is necessary to first download the package install profile from here.
- When updating an existing rasdaman installation, you can find the default package install profile in your installation at `/opt/rasdaman/share/rasdaman/installer/profiles/package/install.toml`.

Download / copy the `install.toml` file to some place, e.g. `$HOME/rasdaman_install.toml`, and make any desired changes to it before installing or updating rasdaman. Make sure that the `RAS_INSTALL_PATH` environment variable is set to point to the custom profile, e.g.

```
export RAS_INSTALL_PATH="$HOME/rasdaman_install.toml"
```

When you install or update rasdaman afterwards, the configuration process will take the custom profile into account instead of the default one.

Running rasdaman

A `rasdaman` service script allows to start/stop rasdaman, e.g.

```
$ service rasdaman start
$ service rasdaman stop
$ service rasdaman status
```

It can be similarly referenced with `systemctl`, e.g.

```
$ systemctl start rasdaman
$ systemctl stop rasdaman
$ systemctl status rasdaman
```

The service script can be customized by updating environment variables in `/etc/default/rasdaman` (create the file if it does not exist). The default settings can be seen below.

```
# rasdaman installation directory
RMANHOME=/opt/rasdaman
# local user running the rasdaman server
RMANUSER=rasdaman
# runuser, or sudo for older OS
RUNUSER=runuser
# login credentials for non-interactive rasdaman start/stop
RASLOGIN=rasadmin:d293a15562d3e70b6fdc5ee452eaed40
# port on which clients connect to rasdaman
RASMGR_PORT=7001
# options to be passed on to start_rasdaman.sh
START_RASDAMAN_OPTS="-p $RASMGR_PORT"
# options to be passed on to stop_rasdaman.sh
STOP_RASDAMAN_OPTS="-p $RASMGR_PORT"
# Java options to be passed on to embedded petascope
JAVA_OPTS="-Xmx4000m"
```

See also the dedicated pages on *configuration and log files* and *administration*.

Check *this section* on how to start / stop the petascope component of rasdaman.

2.2.2 Build From Source Guided

The *rasdaman installer* tool allows users to install rasdaman on a machine through a single script which wraps and hides all the *details of manual compilation* - it can't be easier than that! And it is safe: you can inspect the script and see what's happening. Plus, you retain full control over your configuration by simply editing a JSON file.

Currently, the following distributions are supported:

- Debian (9, 10)
- Ubuntu (16.04, 18.04, 20.04, 22.04)
- CentOS (7)

First-Time Installation

Download the installer and execute it:

```
$ wget https://download.rasdaman.org/installer/install.sh
$ bash install.sh
```

This creates a vanilla installation in `/opt/rasdaman` using reasonable default configurations from `/tmp/rasdaman-installer/profiles/installer/default.toml` (see the *installer configuration page* for more details).

Note that the script needs sudo rights for installing rasdaman into its proper system directory (`/opt/rasdaman`) and for installing package dependencies.

If SELinux is running then likely some extra configuration is needed to get petascope run properly after the installer has finished. Continue *here* for more details on this.

Updating an Existing Installation

Updating a rasdaman installation (if established with the installer) is just as easy:

```
$ update_rasdaman.sh
```

That is all, follow the instructions on the screen and you should be done.

Note: The updating feature of the installer has been introduced more recently. If you have an older installer on your system, please follow the steps for first-time installation after manually stopping rasdaman.

Creating Your Own Profile

The installer makes use of a configuration file, `installer_profile.toml`, created during first-time rasdaman installation and reused during updates. While reasonable defaults are built in, settings can be tweaked by editing the TOML file.

After establishing this file in e.g. `/opt/rasdaman/share/rasdaman/installer/install_profile.toml`, apply it through

```
$ ./install.sh -j /opt/rasdaman/share/rasdaman/installer/install_
↪profile.toml
```

Installer configuration

Default Installer configuration:

```
[general]
# The user running rasdaman
user = "rasdaman"
# Run the installation automatically without requiring any user␣
↪input
auto = true
# Build and install rasdaman? Updating an existing installation is␣
↪supported
install = true
# Set to true to uninstall rasdaman; if install is enabled as well␣
↪then
# any existing rasdaman installation is removed first.
# Important: review the [uninstall] section for customization of␣
↪the uninstall process.
uninstall = false

#
# Configure actions before rasdaman building and installation starts
#
[pre_install]
# Install third party dependencies needed to compile / run rasdaman␣
↪with
# apt-get/yum for packages available in the standard package␣
↪manager, or with
# pip for python packages.
# If this is set to false, the installer will just print the␣
↪package list and
# probably fail compilation or some further step if a package is␣
↪missing.
install_dependencies = true

#
# Configure how to install rasdaman
#
[install]
# Install rasdaman from: "source" or "package"
from = "source"
# Target installation directory ($RMANHOME)
install_path = "/opt/rasdaman/"
# Database backend to use for storing RASBASE: sqlite or␣
↪(deprecated) postgresql
database = "sqlite"
# Rasmgr port: best to stick to the default value of 7001, as␣
↪otherwise it has
# to be explicitly specified in clients that connect to rasdaman.
rasmgr_port = 7001
```

(continues on next page)

(continued from previous page)

```
[install.source]
# Rasdaman sources will be fetched from this repo
repository = "git://rasdaman.org/rasdaman.git"
# Rasdaman version to compile, e.g. master, v9.5.0, etc.
version = "master"
# Generate debug-ready binaries (slower performance)
debug = false
# Build in strict mode (compiler warnings terminate compilation)
strict = false
# Apply a particular patch before building; can be a URL or a path
patch = "https://rasdaman.org/patchmanager?
↪patchop=Download+Selected-{patch_id}"
# Whether to generate documentation
generate_docs = false

# Common servlet container settings for an externally deployed␣
↪petascope/SECORE.
[install.webapps]
# Install Java web applications (petascope, SECORE)
enable = true
# Deployment type: "external" (e.g. in Tomcat), or "standalone"
deployment = "external"
# The options below are only taken into account for "external"␣
↪deployment;
# If these settings are not specified the installer will try to␣
↪guess them: for
# supported distributions/versions this works well.
# Directory where Java web apps are deployed
webapps_path = "/var/lib/tomcat7/webapps/"
# Directory where the servlet container stores logs
webapps_logs = "/var/log/tomcat7/"

[install.webapps.petascope]
# petascope will use this port when deployment is "standalone"
standalone_port = 9009
# JDBC connection URL
petascopedb_url = "jdbc:postgresql://localhost:5432/petascopedb"
# Database username
petascopedb_username = "petauser"
# Database password; a random password will be generated if it is␣
↪empty
petascopedb_password = ""

[install.webapps.secore]
# SECORE will use this port when deployment is "standalone"
standalone_port = 9010

#
```

(continues on next page)

(continued from previous page)

```
# Configure what to do after rasdaman is installed and running
#
[post_install]
# Import demo collections (with rasdaman_insertdemo.sh) and demo␣
↪coverages
# (with petascope_insertdemo.sh)
insert_demo = true
# Run the systemtest
systemtest = false
# Generate a Linux package; if this is enabled then rasdaman will␣
↪not be
# configured to run but just compiled (dependencies needed to run␣
↪rasdaman
# will not be installed either)
generate_package = false

[post_install.package]
# Profile to be used after the package is installed to configure␣
↪rasdaman
profile_path = "profiles/package/deb/default.toml"
# Generated package details
name = "rasdaman"
description = """\
Rasdaman is the leading Array Database for flexible, scalable␣
↪analytics of massive \
multi-dimensional array (raster) data, such as spatio-temporal␣
↪datacubes."""
version = "9.5.0"
# Each new package of the same version should have a progressively␣
↪higher
# iteration number (starting from 1); the resulting package version␣
↪will
# typically be <version>-<iteration>
iteration = "{iteration}"
vendor = "rasdaman"
licence = "GPLv3"
category = "devel"
maintainer = "Dimitar Misev <misev@rasdaman.com>"
url = "https://rasdaman.org"

#
# Configure rasdaman uninstall; these settings are only valid if␣
↪uninstall
# is set to true in the [general] section.
#
[uninstall]
# Remove RASBASE and petascopedb?
remove_data = true
# Remove configuration files?
```

(continues on next page)

(continued from previous page)

```
remove_configs = true
```

2.2.3 Build From Source Manually

This section outlines the procedure for downloading and installing rasdaman from scratch.

Preparation

Create Dedicated User

While rasdaman can be installed and run under any operating system user, for security reasons it is strongly recommended to create a dedicated user to shield rasdaman activity (e.g., log files) from the rest of the system.

This user can be named `rasdaman`, but any other (pre-existing or newly established) user will do as well; in this case, adjust the commands listed in the sequel where necessary. In the sequel it will be assumed that a user account named `rasdaman` has been created, e.g. with

```
$ sudo adduser rasdaman
```

and that you are logged in as user `rasdaman`, e.g. with this command:

```
$ sudo -u rasdaman -i
```

Warning: The dollar sign ("$") symbolizes the command line prompt and is not to be typed in.

Note: As user `rasdaman` probably does not have sudo rights, make sure to execute the commands that require sudo with a user that has sudo rights.

Create Installation Directory

`$RMANHOME` is the target directory where rasdaman will be installed; by default this directory is `/opt/rasdaman`. Make sure it exists and the rasdaman user has write access to it:

```
$ export RMANHOME=/opt/rasdaman
$ mkdir -p $RMANHOME
$ chown rasdaman: $RMANHOME
```

Install Required Packages

build tools:

- *git* – needed to clone the rasdaman git repository
- *cmake* – for generating the makefiles needed to compile rasdaman
- *make*, *libtool*, *pkg-config* – general tools needed to configure and compile rasdaman
- *flex*, *bison*, *g++*, *libstdc++* – required for compilation of the C++ codebase
- *unzip*, *curl* – for compiling 3rd party dependencies of rasnet (grpc and protobuf)
- *maven2*, *OpenJDK 7+* – required for compilation of the Java code (Java client API, petascope OGC frontend, SECORE)

general libraries:

- *libssl-dev*, *libedit-dev*, *libreadline-dev*, *libboost-dev* (v1.48+), *libffi-dev* – required for various system tasks
- *libgdal-dev* – required for data format support (TIFF, JPEG, PNG, etc.)

database stuff: Pick one option below for rasdaman storage:

- *libsqlite*, *libsqlite-dev*, *sqlite3* – required for storing arrays in a filesystem directory and the rasdaman technical metadata in SQLite; see *details*; note that petascope currently requires PostgreSQL independently from the PostgreSQL / file system array decision - in other words: even if for the array engine you chose to not use PostgreSQL you currently still need to install it for storing the geo metadata making an array an OGC coverage)
- *libecpg-dev*, *postgresql* – required for PostgreSQL to hold rasdaman arrays and/or petascope geo metadata

optional packages:

- *libnetcdf-dev*, *python-netcdf4* – required for NetCDF support
- *libeccodes-dev*, *libgrib2c-dev* – for GRIB data support
- *libhdf4-dev* – required for HDF4 support
- *libtiff-dev*, *libjpeg-dev*, *ligpng-dev* - internal encoder/decoder implementations for TIFF, JPEG, or PNG formants.
- *libdw-dev* / *elfutils-devel* – for segfault stacktraces, useful in development
- *sphinx*, *sphinx_rtd_theme*, *latexmk*, *texlive* – main HTML / PDF documentation
- *doxygen* – generate C++ API documentation
- *r-base*, *r-base-dev* – required for *Initialize R support*, an R package providing database interface for rasdaman
- *performance boosters and additional service components* offered by rasdaman GmbH

geo data support (optional):

- Tomcat (or another suitable servlet container) – required for running the petascope and SECORE Java web applications, unless they are configured to start in standalone mode
- *python3* – Python 3.6+ to run *wcst_import <data-import>*, a tool for importing geo-referenced data into rasdaman / petascope
- *python3-pip*, *python3-setuptools*, *python3-wheel* – required to install Python dependencies for wcst_import
- *python-dateutil*, *lxml*, *numpy*, *netCDF4*, *GDAL*, *pygrib*, *jsonschema* – Python 3 dependencies for wcst_import, best installed with pip3

Installation commands for the packages is depending on the platform used, here is a guidance for some of the most frequently used.

Note: When installing the GDAL Python bindings with `pip3 install --user GDAL==...`, it is possible to come across an error similar to `cpl_port.h: No such file or directory`. To fix it, search for cpl_port.h on your system, e.g. `find / -name cpl_port.h`; normally it will be in `/usr/include/gdal`. Then retry the same pip3 command installing *only* GDAL, with additional `--global-option` arguments:

```
$ pip3 install --user --global-option=build_ext \
                      --global-option="-I/usr/include/gdal" GDAL==..
↪.
```

CentOS 7

```
# To build rasdaman
$ sudo yum install \
  make libtool autoconf bison flex flex-devel git curl gcc gcc-c++␣
↪unzip \
  boost-devel libstdc++-static boost-static libtiff-devel zlib-
↪devel \
  libedit-devel readline-devel libpng-devel netcdf-devel postgresql-
↪devel \
  eccodes-devel hdf-devel sqlite-devel openssl-devel libxml2-devel␣
↪elfutils-devel
# To build Java components
$ sudo yum install java-1.8.0-openjdk-devel maven ant

# CMake needs to be manually downloaded and installed as the system
# provided version is too outdated.

# To generate HTML documentation
$ sudo pip install sphinx sphinx_rtd_theme
# To generate PDF documentation (in addition to above)
$ sudo yum install python-pip texlive-cm texlive-ec texlive-ucs \
  texlive-metafont-bin texlive-fncychap texlive-pdftex-def texlive-
↪fancyhdr \
```

(continues on next page)

(continued from previous page)

```
  texlive-titlesec texlive-framed texlive-wrapfig texlive-parskip \
  texlive-upquote texlive-ifluatex texlive-cmap texlive-makeindex-
↪bin \
  texlive-times texlive-courier texlive-dvips texlive-helvetic␣
↪latexmk
# To generate C++ API documentation
$ sudo yum install doxygen

# To run rasdaman
$ sudo yum install \
  postgresql-server postgresql-contrib sqlite zlib elfutils netcdf␣
↪libtiff \
  libedit readline openssl libxml2 which python3-devel python3-pip \
  python3-setuptools python3-wheel eccodes hdf sysvinit-tools
# To run Java components
$ sudo yum install java-1.8.0-openjdk tomcat

# To run wcst_import.sh
$ sudo pip3 install jsonschema python-dateutil lxml \
  pyproj pygrib numpy netCDF4==1.2.7 pygrib

# To run rasdapy
$ pip3 install --user grpcio==1.9.0 protobuf==3.6.1

# To run systemtest
$ sudo apt-get install bc vim-common valgrind netcdf-bin libpython3-
↪dev
```

Debian 10 / Ubuntu 18.04 / Ubuntu 20.04

```
# To build rasdaman
$ sudo apt-get install --no-install-recommends \
  make libtool gawk autoconf automake bison flex git g++ unzip␣
↪libpng-dev \
  libjpeg-dev libboost-filesystem-dev libboost-thread-dev libboost-
↪system-dev \
  libtiff-dev libgdal-dev zlib1g-dev libffi-dev libboost-dev␣
↪libnetcdf-dev \
  libedit-dev libreadline-dev libdw-dev libsqlite3-dev libgrib2c-
↪dev curl \
  libssl-dev libeccodes-dev cmake ccache
# To build Java components
$ sudo apt-get install default-jdk-headless maven ant libgdal-java

# To generate HTML documentation
$ pip3 install --user sphinx sphinx_rtd_theme
# To generate PDF documentation (in addition to above)
```

(continues on next page)

(continued from previous page)

```
$ sudo apt-get install --no-install-recommends latexmk texlive-
↪latex-base \
  texlive-fonts-recommended texlive-latex-extra
# To generate C++ API documentation
$ sudo apt-get install --no-install-recommends doxygen

# To run rasdaman
$ sudo apt-get install \
  postgresql postgresql-contrib sqlite3 zlib1g libdw1 gdal-bin␣
↪debianutils \
  libedit-dev libnetcdf-dev python3-pip python3-setuptools python3-
↪wheel \
  libreadline-dev libssl1.1 libeccodes0
# To run Java components
$ sudo apt-get install default-jre-headless libgdal-java tomcat9

# To run wcst_import.sh; it is recommended to install Python 3.6
$ pip3 install --user jsonschema python-dateutil lxml \
  pyproj pygrib numpy netCDF4==1.2.7 GDAL==2.2.3
# To run rasdapy
$ pip3 install --user grpcio==1.9.0 protobuf==3.6.1

# To run systemtest
$ sudo apt-get install bc vim-common valgrind netcdf-bin libpython3-
↪dev
```

Ubuntu 22.04

```
# To build rasdaman
$ apt-get install --no-install-recommends make libtool gawk␣
↪autoconf automake \
  pkg-config bison flex git g++ unzip libpng-dev libjpeg-dev␣
↪libtiff-dev \
  libgdal-dev libnetcdf-dev libeccodes-dev libboost-filesystem-dev␣
↪libssl-dev \
  libboost-thread-dev libboost-system-dev libboost-dev zlib1g-dev␣
↪libffi-dev \
  libedit-dev libreadline-dev libdw-dev libsqlite3-dev libgrib2c-
↪dev curl
# To build Java components
$ apt-get install default-jdk-headless maven ant

# To generate HTML/PDF and C++ API documentation
$ apt-get install latexmk tex-gyre python3-sphinx python3-sphinx-
↪rtd-theme \
  texlive-latex-base texlive-fonts-recommended texlive-latex-extra␣
↪doxygen
```

(continues on next page)

(continued from previous page)

```
# To run rasdaman
$ apt-get install sqlite3 zlib1g libdw1 debianutils sudo libssl3␣
↪gdal-bin \
  libnetcdf-dev libgdal-dev libeccodes0 libreadline-dev libedit-dev␣
↪\
  python3-jsonschema python3-dateutil python3-lxml python3-grib␣
↪python3-numpy \
  python3-netcdf4 python3-pyproj
# To run Java components
$ apt-get install postgresql postgresql-contrib default-jre-headless

# To run systemtest
$ apt-get install bc vim-common valgrind netcdf-bin gdal-bin␣
↪python3-protobuf \
  python3-pip jq
$ pip3 install grpcio pylint==2.13.4
```

Note: Two files - *gdal.jar* and *libgdalalljni.so*, are absent in Ubuntu 22.04. You need to manually paste *gdal.jar* at `/usr/share/java` and *libgdalalljni.so* at `/usr/lib/jni/` for a successful build.

You can find these files here: https://download.rasdaman.org/installer/tpinstaller/ubuntu2204/

Download and Install rasdaman

Download

You can get a complete *rasdaman Community* distribution from www.rasdaman.org by executing the following command:

```
$ git clone git://rasdaman.org/rasdaman.git
```

This will create a sub-directory rasdaman in your current working directory.

Configure

Change into the newly cloned directory:

```
$ cd rasdaman
```

Optionally, select a tagged stable release. To activate a particular tagged version use its name prefixed with a "v", e.g:

```
$ git checkout v9.8.1
```

Note: You can list all tags with `git tag`.

The following commands will prepare for building on your system. First create a build directory:

```
$ mkdir -p build
$ cd build
```

In the build directory we next execute `cmake` to configure how rasdaman is compiled. A typical configuration looks like this:

```
$ cmake .. -DCMAKE_INSTALL_PREFIX=$RMANHOME
```

Any missing components will be reported; if this is the case, then install the missing packages and retry configuration. The `..` indicates the path to the rasdaman source tree, which is now the parent directory of the `build` directory in which the `cmake` command is executed.

The general *format* of invoking `cmake` on the command-line is as follows:

```
$ cmake /path/to/rasdaman/sources [ -D<option>... ]
```

Note: Alternatively, *ccmake* or *cmake-gui* can be used as graphical interfaces for this configuration step.

Configuration can be customized, Table 2.1 summarizes the options that can be specified with `-D<option>`, along with the default settings.

Note: To get a current list of all the custom options that can be passed to `cmake` on the command line, try `cmake -LH`.

Table 2.1: CMake options for configuring the installation

Option	Alternatives	Description
`CMAKE_INSTALL_PREFIX`	<path> (default /opt/rasdaman)	Installation directory.
`CMAKE_BUILD_TYPE`	**Release** / Debug	Specify build type, Release for production, Debug for development
`CMAKE_VERBOSE_OUTPUT`	ON / **OFF**	Enable this if you need detailed output from the make process.
`CMAKE_CXX_FLAGS`	<flags>	Specify additional compiler options, e.g. -DCMAKE_CXX_FLAGS=”-g3”
`DEFAULT_BASEDB`	**sqlite** / postgresql	Specify the DBMS that rasdaman uses for storing RASBASE.
`ENABLE_BENCHMARK`	ON / **OFF**	Generate binaries that contain extra code for benchmark output.
`ENABLE_PROFILING`	ON / **OFF**	Enable profiling of queries with google-perftools.
`ENABLE_DEBUG`	ON / **OFF**	Generate (slower) binaries that can be debugged / produce debug logs.
`ENABLE_ASAN`	ON / **OFF**	Compile with AddressSanitizer enabled (-fsanitize=address)
`ENABLE_STRICT`	ON / **OFF**	Enable compilation in strict mode (warnings terminate compilation).
`ENABLE_R`	ON / **OFF**	Enable compilation of R support.
`GENERATE_DOCS`	**ON** / OFF	Generate and install documentation (manuals, doxygen, javadoc).
`GENERATE_PIC`	**ON** / OFF	Generate position independent code (PIC).
`ENABLE_JAVA`	**ON** / OFF	Generate and install of Java-based components (rasj, petascope, secore).
`JAVA_SERVER`	**embedded** / external	Set the Java application deployment mode.
`ENABLE_STANDALONE_SECORE`	ON / **OFF**	Build SECORE as a standalone web application *.war*.
`USE_GDAL`	**ON** / OFF	Enable inclusion of GDAL library during installation. Further variables can be set to control the GDAL paths: `-DGDAL_INCLUDE_DIR`, `-DGDAL_LIBRARY`, `-DGDAL_JAVA_JAR_PATH`
`USE_GRIB`	ON / **OFF**	Enable inclusion of GRIB library during installation. Further variables allow controlling the GRIB library paths: `-DGRIB_LIBRARIES` and `-DGRIB_INCLUDE_DIR`
`USE_HDF4`	ON / **OFF**	Enable inclusion of HDF4 library during installation. Further variables allow controlling the HDF4 library paths:

Build

Next, execute `make` to compile and link rasdaman:

```
$ make -j2
```

Note: Compiling rasdaman can take awhile. `-j2` sets make to compile in parallel with 2 threads; it's recommended to increase this number to match the number of cores on your system (check with the `nproc` command).

To further improve the compilation speed, especially if you're recompiling rasdaman often, it can be helpful to install *ccache*.

Install

Install rasdaman to the directory specified before with `-DCMAKE_INSTALL_PREFIX`:

```
$ make install
```

Note: The user executing this command must have write access to the target directory specified. If `-DWAR_DIR` was specified, then it also needs to have write access to this directory. Information on enabling this without using sudo can be found in the *Preparation* Section.

As described in the previous section, the installation directory is chosen at compile time. Inside this installation directory we find the binary executable programs, development libraries, documentation, etc. (covered in more detail in Section *Installed Files and Data*). For your convenience you can add the executable path location to the `$PATH` definition, e.g:

```
$ export RMANHOME=/opt/rasdaman
$ export PATH=$RMANHOME/bin:$PATH
```

This allows to invoke `rasql` without specifying the full path `/opt/rasdaman/bin/rasql`.

Note: This only takes effect in the current terminal. To preserve them accross shell sessions, these settings can be appended to the `~/.bashrc` file.

Note: All paths *inside* rasdaman scripts and binaries are adjusted automatically during generation, so you do not need to edit any script.

Update rasdaman

In order to be able to update your working installation in future, it is best to keep the cloned rasdaman repository along with the build directory. Otherwise updating would require following the same steps from the *beginning*.

Skip to the *next section* if this is the first time your installing rasdaman. This section is only applicable if you already have a running, functional instance of rasdaman on your system.

To update, first change to the rasdaman source tree which was cloned in the first step, and run the following command:

```
$ git pull
```

If you haven't changed any source files, the command should execute successfully and download the latest changes in the rasdaman repository since the last time you cloned or updated the repository.

Next, the *build* and *install* steps need to be repeated. However, rasdaman should be stopped before, and started afterwards, so that the updated installation is fully reflected in the running system. In addition, the database schema of rasdaman may need to be updated with the `update_db.sh` command. In summary:

```
$ make -j2

$ stop_rasdaman.sh

$ make install
$ update_db.sh

$ start_rasdaman.sh
```

Initialize rasdaman

Create Relational Database

For the default SQLite based backend of rasdaman it is just necessary to make sure that the rasdaman user has read/write/executable access to the data directory specified with -DFILE_DATA_DIR or the environment variable `$RASDATA`.

For PostgreSQL it is necessary to make sure that rasdaman can login and is able to create databases and tables. Currently **ident-based authentication** is supported. A PostgreSQL user named as the operating system user under which rasdaman will be operated (e.g. `rasdaman` as recommended above) needs to be created, e.g:

```
$ sudo -u postgres createuser -s rasdaman
```

Database Initialization

The `create_db.sh` script creates and initializes a rasdaman database named `RASBASE` by instantiating a set of standard types in rasdaman. It has no parameters and is invoked as:

```
$ create_db.sh
```

Note: The rasdaman server should be stopped when running this command.

Server Configuration (Optional)

Rasdaman is a multi-server multi-user system. The server processes available must be configured initially, which is done in file `$RMANHOME/etc/rasmgr.conf`. For distribution, this configuration contains ten server processes going by a name like, for example, `N1`. If this is fine then you can just leave it as it is. If you want to change this by modifying server startup parameters or increasing the number of server processes available then see *rascontrol Invocation* for details on how to do this.

Server Start/Stop

Make sure that the ports rasdaman uses are not blocked in your system. These are 7001 for the scheduler (rasmrg) and 7002, 7003, etc. for each worker process. Ports used can be reconfigured, cf. *Server Manager and Server*.

Start rasdaman by invoking

```
$ start_rasdaman.sh
```

Note: Messages printed by `start_rasdaman.sh` will not always show the detailed system state. If, for example, the rasdaman servers fail to contact the base DBMS then nevertheless a message "Server started" may appear.

Workaround: use this to get the actual server state, as user `rasdaman`:

```
$ rascontrol -e -x "list srv -all"
```

Correspondingly, rasdaman can be stopped by invoking

```
$ stop_rasdaman.sh
```

Demo Database

The rasdaman distribution contains a demo database which serves as a first test of successful installation.

Inserting demo data into the fresh database is done through

```
$ rasdaman_insertdemo.sh localhost 7001 \
  $RMANHOME/share/rasdaman/examples/images rasadmin rasadmin
```

Note that repeated invocations are not harmful - each of the sample collection will simply receive additional objects made of the same images.

After successful completion, you can check whether the three rasdaman collections containing the example images have been created through:

```
$ rasql -q "select r from RAS_COLLECTIONNAMES as r" \
        --out string
```

This command shows a list of all collections existing in the database. There should be `mr`, `mr2`, and `rgb`.

Congratulations! At this point, if everything completed successfully, rasdaman is up and running and prepared for data definition, data import and retrieval, and any other suitable task.

Initialize geo service support

petascope

Petascope is the geo Web service frontend of rasdaman. It adds geo semantics on top of arrays, thereby enabling regular and irregular grids based on the OGC coverage standards.

Petascope gets installed automatically as `rasdaman.war` unless a `-DENABLE_JAVA=OFF` (cf. Table 2.1) is specified. The deployment directory of all war files can be set during the `configure` step with the `-DWAR_DIR=<DIR>` cmake option; by default this is `$RMANHOME/share/rasdaman/war`.

To implement the geo semantics, petascope uses a relational database for the geo-related metadata. Currently, PostgreSQL and H2 / HSQLDB are supported. When installing from packages, the package post-install script will automatically set up PostgreSQL for use by petascope. The steps approximately performed by the script are listed below.

PostgreSQL

PostgreSQL is automatically configured when rasdaman is installed, so doing the below is not usually necessary; we list the steps as documentation of how is PostgreSQL configured by default:

1. If postgres has not been initialized yet:

   ```
   $ sudo service postgresql initdb
   ```

If the output is 'Data directory is not empty!' then this step is skipped.

2. Trust-based access in PostgreSQL is enabled by adding the below configuration before the ident lines to `/etc/postgresql/9.4/main/pg_hba.conf` on Debian 8, or `/var/lib/pgsql/data/pg_hba.conf` on CentOS 7:

```
host     all    petauser    localhost       md5
host     all    petauser    127.0.0.1/32    md5
host     all    petauser    ::1/128         md5
```

3. Reload PostgreSQL so that the new configuration will take effect:

```
$ sudo service postgresql reload
```

4. Add a petascope user, for example `petauser`, to PostgreSQL:

```
$ sudo -u postgres createuser -s petauser -P
> enter password
```

In `$RMANHOME/etc/petascope.properties` set the `spring.datasource.username`/`spring.datasource.password` and `metadata_user`/`metadata_pass` options accordingly to this user / password. The password is randomly generated.

5. Copy `/opt/rasdaman/share/rasdaman/war/rasdaman.war` to the Tomcat webapps directory (`/var/lib/tomcat/webapps` on CentOS 7) and restart Tomcat.

Following successful deployment, petascope accepts OGC W*S requests at URL `http://localhost:8080/rasdaman/ows`.

H2 / HSQLDB

To alternatively set up H2 / HSQLDB for use by petascope instead of PostgreSQL:

1. Create a directory that will host petascopedb and the H2 driver:

```
$ mkdir /opt/rasdaman/geodb
```

2. Make sure the user running the webserver serving petascope can read/write to the folder above. For example, Tomcat webserver which uses *tomcat* user

```
$ sudo chown -R tomcat: /opt/rasdaman/geodb
```

However, if embedded deployment is enabled in petascope.properties, then the owner should be the `rasdaman` user which runs rasdaman

```
$ sudo chown -R rasdaman: /opt/rasdaman/geodb
```

3. Download the driver and place it in the created directory. For example, download a H2 driver

```
$ cd /opt/rasdaman/geodb
$ wget https://repo1.maven.org/maven2/com/h2database/h2/1.4.200/
↪h2-1.4.200.jar
```

(continues on next page)

(continued from previous page)

```
```

4. Configure database settings in petascope.properties file, see *details*.
5. Restart the webserver running petascope (or rasdaman if embedded tomcat).

SELinux configuration

If `SELinux` is enabled (result of `getenforce` is `enforcing`) then permissions for the `tomcat` user which is running petascope need to be configured properly if petascope is running in an external servlet container (as opposed to *embedded*):

- Allow to load the `gdal-java` native library (via JNI)
- Read / write files in `/tmp/rasdaman_*`
- Make HTTP requests to rasdaman and get back results on ports `7001-7010` (these are default, specified in `$RMANHOME/etc/rasmgr.conf`).

Before proceeding, a SELinux utility package needs to be installed on CentOS 7:

```
$ sudo yum install policycoreutils-python
```

There are two ways to configure SELinux in order to enable petascope:

1. Change from `enforcing` to `permissive` for Tomcat:

   ```
   $ semanage permissive -a tomcat_t
   ```

2. Create specific rules for the `tomcat` user and register with `SELinux`.

- Create a rule config file `tomcat_config.te` with this contents:

```
module tomcat_config 1.0;

require {
    type tomcat_t;
    type tomcat_var_lib_t;
    type usr_t;
    type tomcat_exec_t;
    type unconfined_service_t;
    type afs_pt_port_t;
    type tomcat_tmp_t;
    type tmpfs_t;
    type afs3_callback_port_t;
    class tcp_socket name_connect;
    class file {
        append create execute read relabelfrom rename write };
    class shm {
        associate getattr read unix_read unix_write write };
}

# ============= tomcat_t ==============
```

(continues on next page)

(continued from previous page)

```
allow tomcat_t afs3_callback_port_t:tcp_socket name_connect;
allow tomcat_t tmpfs_t:file { read write };
allow tomcat_t tomcat_tmp_t:file { execute relabelfrom };
allow tomcat_t tomcat_var_lib_t:file execute;
allow tomcat_t unconfined_service_t:shm {
    associate getattr read unix_read unix_write write  };
```

- Create a shell script `deployse.sh` to generate a binary package from this config file:

```
#!/bin/bash
set -e
MODULE=${1}
# this will create a .mod file
checkmodule -M -m -o ${MODULE}.mod ${MODULE}.te
# this will create a compiled semodule
semodule_package -m ${MODULE}.mod -o ${MODULE}.pp
# this will install the module
semodule -i ${MODULE}.pp
```

- Run the script to load the binary package module to `SELinux`:

```
$ sudo ./deployse.sh tomcat_config
```

Restart Tomcat with `sudo service tomcat restart`; now rasdaman should be able to import data to petascope via WCSTImport and get data from rasdaman via WCS / WMS / WCPS.

SSL/TLS configuration

Transport Layer Security (`TLS`) and its predecessor, Secure Sockets Layer (`SSL`), are technologies which allow web browsers and web servers to communicate over a secured connection. To configure it for `petascope` and `secore web` applications for `Tomcat`, check the official guide.

Initialize R support

`RRasdaman` is an `R` package providing database interface for rasdaman. This manual describes the installation process of the package.

Note: This package is still in beta. We are seeking contributors to finalize it and submit it to CRAN.

1. Install `R`:

```
$ sudo apt-get install r-base r-base-dev
```

2. Install needed R packages; from the R prompt:

   ```
   $ R --quiet
   > install.packages(c("rJava", "testthat"))
   ```

   In case an error "/usr/bin/ld: cannot find -lpcre (-llzma, -lbz2)" appears, install the following system packages needed for rJava:

   ```
   $ sudo apt-get install liblzma-dev libbz2-dev libpcre3-dev
   ```

3. Make sure that rasdaman was configured with -DENABLE_R=ON before proceeding.
4. Build and install the R package, in the rasdaman build directory:

   ```
   $ cd applications/RRasdaman
   $ make
   $ make install
   ```

5. Start rasdaman, then check from within an R session that everything works:

   ```
   $ R
   > library(RRasdaman)
   > conn <- dbConnect(Rasdaman())
   > dbListCollections(conn)
    [1] "mr"                          "rgb"
    [3] "mean_summer_airtemp"         "eobstest"
   > dbDisconnect(conn)
   ```

6. Optionally, run the package tests. This also requires the rasdaman up and running:

   ```
   $ cd applications/RRasdaman
   $ make check
   ```

2.2.4 Preconfigured Virtual Machines

rasdaman @ OSGeo Live

A complete VM with all OSGeo certified tools, including rasdaman, is available for download at live.osgeo.org. Be aware that this installation relies on the OSGeo release cycle and, therefore, will usually not reflect the latest software state.

rasdaman vagrant boxes

The following vagrant boxes can be used to quickly setup a rasdaman test environment with vagrant:

```
rasdaman/ubuntu1804
rasdaman/ubuntu2004
rasdaman/centos7e
rasdaman/centos7e_gdal2
```

rasdaman is not installed, this can be done by following the guides for installing rasdaman from a package or building from source. All packages needed for building rasdaman are preinstalled and the sources can be found in `/opt/rasdaman/source` (make sure to `git pull` to get the latest version). In `/opt/rasdaman/third_party` there is a cmake that can be used to configure and build rasdaman. To build and install rasdaman, you can use the rasdaman installer or do it from scratch.

It is not required to use the rasdaman-specific boxes, you can use any box published on the vagrant cloud such as `ubuntu/jammy64`.

Here is a sample `Vagrantfile` for the Ubuntu 20.04 box:

```
Vagrant.configure(2) do |config|
   config.vm.box = "rasdaman/ubuntu2004"
   config.vm.box_check_update = false
   config.vm.synced_folder ".", "/vagrant", type: "rsync"
   config.vm.provider "virtualbox" do |vb|
     # allow 6GB RAM
     vb.memory = "6000"
     # vb.cpus = 2
   end
   config.vm.provision "shell", inline: <<-SHELL
     # set the default locale
     echo 'LANGUAGE="en_US.UTF-8"' >> /etc/default/locale
     echo 'LC_ALL="en_US.UTF-8"' >> /etc/default/locale
   SHELL
end
```

To quickly get started:

```
$ sudo apt-get install vagrant
$ cd /location/of/Vagrantfile
$ vagrant up
$ vagrant ssh
```

Check the vagrant docs for further information.

2.3 Installed Files and Data

2.3.1 Top-level directories

As common with rasdaman, we refer to the installation location as `$RMANHOME` below; the default is `/opt/rasdaman`. The table below lists the top-level directories found in `$RMANHOME` after a fresh installation.

Directory	Description
`bin`	rasdaman executables, e.g. rasql, start_rasdaman.sh, …
`data`	Path where the server stores array tiles as files; this directory can get big, it is recommended to make it a link to a sufficiently large disk partition.
`etc`	Configuration files, e.g. rasmgr.conf
`include`	C++ API development headers.
`lib`	C++ and Java API libraries.
`log`	`rasmgr` and `rasserver` log files.
`share`	Various artefacts like documentation, python/javascript clients, example data, migration scripts, etc.

2.3.2 Executables

Rasdaman executables are found in `$RMANHOME/bin`; the table below lists the various binaries and scripts. More detailed information on these components is provided in the *Server Architecture* Section.

Executables	Description
`rasserver`	Client queries are evaluated by a `rasserver` worker process.
`rasmgr`	A manager process that controls `rasserver` processes and client/server pairing.
`rascontrol`	A command-line frontend for `rasmgr`.
`directql`	A rasserver that can execute queries directly, bypassing the client/server protocol; useful for debugging.
`rasql`	A command-line client for sending queries to a `rasserver` (as assigned by the `rasmgr`).
`start_rasdaman.sh`	Start `rasmgr` and the worker `rasservers` as configured in `$RMANHOME/etc/rasmgr.conf`. More details *here*.
`stop_rasdaman.sh`	Shutdown rasdaman, embedded petascope and embedded secore if enabled. More details *here*.
`create_db.sh`	Initialize the rasdaman metadata database (RASBASE).
`update_dh.sh`	Applies migration scripts to RASBASE.
`rasdaman_insertdemo.sh`	Insert three demo collections into rasdaman (used in the rasdaman Query Language Guide).
`petascope_insertdemo.sh`	Insert geo-referenced demo coverage in petascope.
`migrate_petascopedb.sh`	Applies database migrations on petascopedb. More details *here*.
`wcst_import.sh`	Tool for convenient and flexible import of geo-referenced data into petascope. More details *here*.
`prepare_issue_report.sh`	Helps preparing a report for an issue encountered while operating rasdaman. More details *here*.

start_rasdaman.sh

This script starts rasdaman. Normally rasdaman is installed from packages, and instead of executing this script directly one would execute `service rasdaman start`. Any options to be passed on to `start_rasdaman.sh` can be set in `/etc/default/rasdaman` in this case; see *more details*.

To start a specific service (rasdaman and *embedded petascope*) the `--service (core | petascope)` option can be used(`core` refers to `rasmgr` + `rasserver` only).

Since v10.0 the rasmgr port can be specified with `-p, --port`. Additionally, for security and usability reasons, `start_rasdaman.sh` will refuse running if executed with root user; this can be overriden if needed with the `--allow-root` option.

The script will use various environment variables, if they are set before it is executed:

- `RASMGR_PORT` - the port on which rasmgr will listen when started, and to which client applications will connect in order to send queries to rasdaman. This variable will be overrided by the value of option `--port`, if specified. By default if none are specified, the port is set to 7001.
- `RASLOGIN` - rasdaman admin credentials which will be used for starting rasmgr non-

interactively. See more details on the format and how is this setting used *here*. If not set, the script defaults to using rasadmin/rasadmin credentials; see here on how to change these defaults.

- `JAVA_OPTS` - options passed on to the `java` command when used to start the OGC frontend of rasdaman (petascope) if it is configured for *embedded deployment*. If not set, it defaults to `-Xmx4000m`

Check `-h, --help` for all details.

stop_rasdaman.sh

This script stops rasdaman. Normally rasdaman is installed from packages, and instead of executing this script directly one would execute `service rasdaman stop`. Any options to be passed on to `stop_rasdaman.sh` can be set in `/etc/default/rasdaman` in this case; see *more details*.

The script stops rasmgr, rasservers, rasfed, and petascope (if configured for embedded deployment) in the correct order with a regular TERM signal to each process; this ensures that the services exit properly. In some cases, a process may be hanging instead of exiting on the TERM signal; since rasdaman v10.0, `stop_rasdaman.sh` will detect and report such cases. It is prudent to then check the relevant process logs, and if it appears that there is no reason for the process hanging one can force-stop it with `stop_rasdaman.sh --force`, or manually do it by sending it a KILL signal (e.g. `kill -KILL <pid>`).

To stop a specific service the `--service (core | petascope )` option can be used. Since v10.0 the rasmgr port can be specified with `-p, --port`.

The script will use various environment variables, if they are set before it is executed:

- `RASMGR_PORT` - the port on which rasmgr was set to listen when it was started. This variable will be overrided by the value of option `--port`, if specified. By default if none are specified, the port is set to 7001.
- `RASLOGIN` - rasdaman admin credentials which will be used for stopping rasmgr non-interactively. See more details on the format and how is this setting used *here*. If not set, the script defaults to using rasadmin/rasadmin credentials; see here on how to change these defaults.

Check `-h, --help` for all details.

migrate_petascopedb.sh

This script is used to migrate coverages imported by wcst_import, OWS Service metadata and WMS 1.3 layers. For more details see *Meta Database Connectivity* and *Initialize geo service support*.

There are 2 types of migration:

1. Migrate petascopedb v9.4 or older to a newer rasdaman version. After the migration, the old petascopedb is backed up at petascope_94_backup.

2. Migrate petascopedb v9.5 or newer to a different database name or different database (e.g. PostgreSQL to HSQLDB).

Note: The petascope Web application must not be running (e.g in Tomcat) while migrating to a different database (type 2 above) to protect the existing data integrity.

2.3.3 Configuration files

Configurations are automatically loaded upon rasdaman start. After any modification a restarthas to be performed for the change to take effect.

Server rasdaman configuration files can be found in `$RMANHOME/etc`:

`rasmgr.conf`	allows fine-tunning the rasdaman servers, e.g. number of servers, names, database connection
`petascope.properties`	set petascope properties, e.g. backend/rasdaman connection details, CRS resolver URLs, features
`secore.properties`	secore configuration

Logging output of petascope and secore is configured in their respective config files, while logging output of rasdaman is controlled via the below configuration files:

`log-rasmgr.conf`	log output of rasmgr
`log-server.conf`	log output of rasserver worker processes
`log-client.conf`	log output of client applications, e.g., rasql

rasdaman uses the Easylogging++ library for logging in its C++ components. Log properties can be configured as documented on the EasyLogging GitHub page.

External potentially relevant configuration files are:

postgresql	**`/var/lib/pgsql/data/{postgresql.conf,pg_h`** `/etc/postgresql/9.X/{postgresql.conf,pg_hba.conf}`
tomcat	`/etc/tomcat/`, `/etc/default/tomcat`

2.3.4 Log files

rasdaman

rasdaman server logs are placed in `$RMANHOME/log/`. The server components feed the following files where `uid` represents a unique identifier of the process, and `pid` is a Linux process identifier:

`rasserver.<uid>.<pid>.log` `rasserver` worker logs: at any time there are several rasservers running (depending on the settings in `rasmgr.conf`) and each has a unique log file.

`rasmgr.<pid>.log` `rasmgr` log: there is only one `rasmgr` process running at any time.

Note: `ls -ltr` is a useful command to see the most recently modified log files at the bottom when debugging recently executed queries.

---

petascope & secore

It is highly recommended to set a specific log file however in the log4j configuration section in `petascope.properties` (e.g. `log4j.appender.rollingFile.File=/var/log/tomcatN/petascope.log`). Be careful that this location needs to be write accessible by the Tomcat user. The same can be set for SECORE in `secore.properties`.

2.3.5 Temporary files

Rasdaman stores various data temporarily in `/tmp/rasdaman\_*` directories, in particular:

- `/tmp/rasdaman\_conversion/` - format-encoded data, such as TIFF, NetCDF, etc., is in some cases temporarily stored here before decoding into rasdaman. This also happens always when encoding query processing results into some format for export. The intermediate data is quickly removed as soon as the encoding or decoding process is finished.

  Temporarily, however, this directory can get rather large: if you export array result that encodes into a 1GB TIFF file, then the directory will contain 1GB of data for some time; if 10 such queries run concurrently, then it may contain up to 10GB of data. For this reason we recommend to check the size of `/tmp` during installation, and make sure it is large enough. It is always recommended to make `/tmp` a separate partition, so as to prevent system-wide problems in case the filesystem is filled up with data.

- `/tmp/rasdaman\_petascope/` - contains small temporary files generated during data import with the wcst_import tool.

- `/tmp/rasdaman\_transaction\_locks/` - during query read/write transaction, rasdaman generates various empty lock files in this directory. As the files are empty, the size of this directory is minimal.

  While rasdaman is running this directory must not be removed, otherwise it may lead to data corruption.

2.3.6 Demo data & programs

Example database

A demonstration database is provided as part of the delivery package which contains the collections and images described in the *Query Language Guide*. To populate this database, first install the system as described here, and then invoke:

```
$ rasdaman_insertdemo.sh
```

The demo database occupies marginal disk space, and is a straightforward way to show that the rasdaman installation has been successfull.

Example programs

Several example programs are provided in the `c++` and `java` subdirectories of `$RMANHOME/share/rasdaman/examples`. Each directory contains a Makefile plus `.cc` and `.java` sources, resp.

Makefile

The `Makefile` helps to compile and link the sample C++ / Java sources files delivered. It is a good source for hints on the how-tos of compiler and linker flags.

Note: All programs, once compiled and linked, print a usage synopsis when invoked without parameter.

`query.cc`

Sends a hardwired query to a running rasdaman system:

In addition, it demonstrates how to work with the result set returned from rasdaman. The query can easily be changed, or made a parameter to the program.

`Query.java`

Sends the following hardwired query if one is not provided as a parameter:

`AvgCell.java`

This program computes the average cell value from all images of a given collection on client side. Note that it requires grayscale images. A good candidate collection is `mr` from the demo database.

2.4 Access Interfaces

Rasdaman services can be invoked in several ways: through command line, Web requests, and custom programs connecting via the C++ and Java APIs.

2.4.1 Command Line Tools

Queries can be submitted to the command line tool `rasql`. Complete control over the server is provided through several utilities, in particular `rasmgr`; see *rascontrol Invocation* for details. All tools can communicate with local and remote rasdaman servers.

2.4.2 Web Services

Several Web services are available with rasdaman. They are implemented as servlets, hence independent from the array engine and only available if started in a servlet container such as Tomcat or jetty. They can be accessed under the common context path `/rasdaman`.

- `/rasdaman/ows` exposes geo Web Services based on the interface standards of the Open Geospatial Consortium (OGC Web Services, OWS). Supported OGC standards are:
  - Web Coverage Service (WCS)
  - Web Coverage Processing Service (WCPS)
  - Web Map Service (WMS) suites
- `/rasdaman/def` provides access to a Coordinate Reference System (CRS) Resolver Service, SECORE. It is identical to the one deployed by OGC, where http://www.opengis.net/def/crs is the branch for CRS served by SECORE.
- `/rasdaman/rasql` provides support for submitting rasql queries and receiving results with standard HTTP requests. Requests must specify three mandatory parameters:

`username`	rasdaman login name under which the query will be executed
`password`	password corresponding to the login
`query`	rasql query string, properly encoded for URI embedding

  Example:

```
http://localhost:8080/rasdaman/rasql
    ?username=rasguest
    &password=rasguest
    &query=select%20encode%28mr2%2C%22png%22%29%20from%20mr
```

Note: rasql servlet also supports rasdaman user credentials in basic authentication header. In this case, `username` and `password` parameters are not required as the credentials are extracted from the header.

The diagram below illustrates the OGC service architecture of rasdaman:

```
clients              read:                          read:
+-----------------+
|                 |  GetCapabilities              select ...
|  +-----------+  |  DescribeCoverage
|  |3rd party  |  |
|  +-----------+  |  GetCoverage
|                 |  ProcessCoverage
|  +-----------+  |  GetMap
|  |wcs_client |  |               +-------------+          +-----
↪----+
|  +-----------+  | +-------------> |rasdaman-geo| +--------->␣
↪|rasserver|
|                 |               +-------------+          +-----
↪----+
|  +-----------+  |  write:                         write:
|  |wcst_import|  |
|  +-----------+  |  InsertCoverage                 create type/coll
|                 |  UpdateCoverage                 insert,update,
↪delete
+-----------------+  DeleteCoverage                 drop type/coll
```

2.4.3 APIs

Programmatic access is available through self-programmed code using the C++ and Java interfaces; see the C++ and Java Guide for details.

2.5 Server Architecture

The parallel server architecture of rasdaman offers a scalable, distributed environment to efficiently process even very large numbers of concurrent client requests. Yet, server administration is easy to accomplish, with only few things to do to have a smoothly running, highly performant installation. Moreover, the system is implemented in a special high availability technique where most server management operations can be done with the server up and running, limiting the need for a server shutdown to the absolute minimum.

In this Section the general rasdaman server architecture is outlined. It is recommended to study this section so as to understand server administration terminology used in the next Section.

2.5.1 Executables Overview

The following executables are provided in the `bin/` directory, among others:

- `rasmgr` is the central rasdaman request dispatcher;
- `rasserver` is the rasdaman server engine, it should not be generally invoked in a standalone manner;
- `rascontrol` allows to interactively control the rasdaman server by communicating with `rasmgr`;
- `rasql` is the command-line based query tool, explained in detail in the *rasdaman Query Language Guide*.

2.5.2 Server Manager and Server

Overview and Terminology

The rasdaman server configuration consists of one dispatcher process per computer, `rasmgr` (we will refer to it as *manager* in the sequel), and server processes, `rasserver` (referred to as *servers*), of which at a given time none, one, or several ones can be running. All server processes are under control of the manager. Server manager and rasdaman server(s) all run on the same physical hardware, the *rasdaman host*.

The servers resolve requests, thereby generating calls to the relational database system which in turn accesses its database files. For the purpose of this manual, the relational server together with the database it maintains are collectively called the *database*. The machine the relational database server runs on is referred to as *database host* (Figure 2.1).

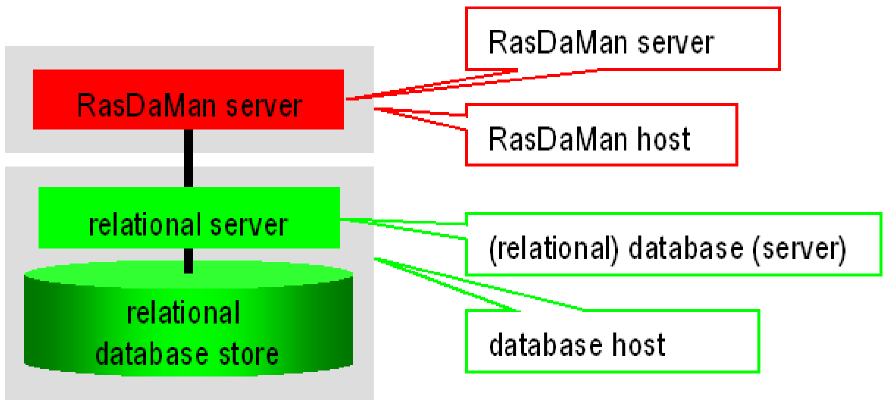


Figure 2.1: Overall server hierarchy, introducing the terminology for rasdaman hardware and software environment

Server Structure in General

The manager accepts client requests and assigns server instances to them, taking them from the pool of server processes it maintains. In distributed installations, it keeps contact to the managers on other machines to further dispatch client requests across all the rasdaman servers available. Whenever needed, the administrator can launch further server instances, or shut them down again.

Upon system configuration definition (see *rascontrol Invocation*), a unique name is assigned to each server identifying it to the manager.

Each rasdaman server is assigned to a relational database server, laid down in the manager configuration file. Databases can be registered and associated to particular rasdaman servers at any time.

rasdaman hosts and database hosts are identified by their resp. host name in common domain address form, e.g., `martini.rasdaman.com` or `199.198.197.50`.

`Rascontrol` is the interactive front-end to `rasmgr` and, as such, the main utility for user and system management. It provides the necessary functions to manage the whole system configuration, to add and remove user, to change their rights, and to obtain information about system activity.

The rasdaman server, i.e., `rasserver`, is controlled by the manager which starts and stops server instances. Hence, the `rasserver` executable should not (and actually cannot) be invoked directly.

Dynamic Server Assignment

The process of client/server communication and server scheduling is done as follows (see numbers in Figure 2.2).

1. The client starts every `OPENDB` and `BEGIN TRANSACTION` request with an HTTP call to the manager, providing the required service type (RPC, HTTP, etc.) and the database name, together with user name and password.
2. The manager's answer is the server ID of a free server, or an error message in case no server is available or access is denied for the given login.
3. Client-Server communication to perform the database requests.
4. Upon `CLOSEDB` and `ABORT/COMMIT TRANSACTION` the server informs the manager that it is available again. This is also done upon a client timeout.

These negotiation steps are performed between client library and server, hence transparent to the application.

The rasdaman server system is started by invoking the server manager `rasmgr` (see *Running the Manager*). If it finds a configuration file, them autopmatically all servers indicated will be started; alternatively, server configuration can be done directly through `rascontrol` (see *rascontrol Invocation*).

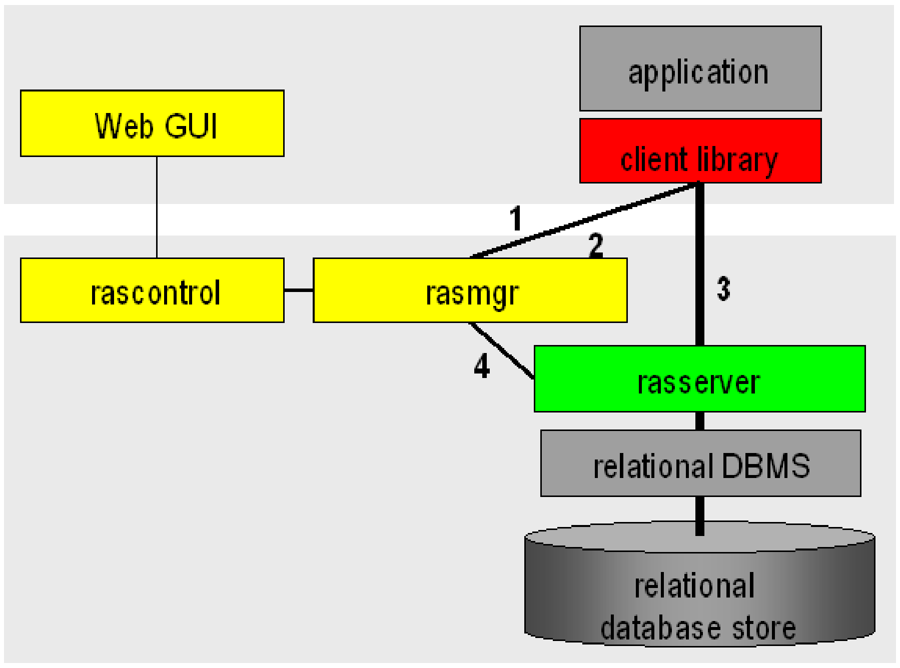


Figure 2.2: Internal server management

System Start-up

Invocation of the `rasmgr` executable must be done under the operating system login under which the rasdaman installation has been done, usually (and recommended) `rasdaman`. The service script `/etc/init.d/rasdaman` (when rasdaman is installed from the packages) automatically takes care of this.

Server Federation

rasdaman servers running on different computers can be coupled so as to form one single server network. To this end, the dispatcher processes, `rasmgr`, running on each node exploits knowledge about other nodes in the network. This is accomplished via `inpeer` and `outpeer` directives, best written into `rasmgr.conf`.

Whenever a local dispatcher finds that a new session cannot be served as there is no more free server process available currently it will attempt to acquire a free server from a peer `rasmgr`. Upon success, this server is transparently communicated to the client.

Any server in the network can forward requests this way (depending on the administrator controlled security policy on each node). Hence, there is **no single point of failure** in such a rasdaman peer network.

All peers in a rasdaman federation are assumed to access the same underlying database, or a database with identical contents.

Authentication

On every machine hosting rasdaman servers a separate manager has to run. The manager maintains an authorization file, `$RMANHOME/etc/rasmgr.auth`. It should not be changed by the administrator, as they are generated, maintained, and overwritten by the manager.

rasdaman Manager Defaults

The manager's default name is the `hostname` (the one reported by the UNIX command hostname), but it can be changed (see the `change` command). By default, it listens to port 7001 for incoming requests and uses port 7001 for outgoing requests:

Port Number Recommendations

To keep overview of the ports used, it is recommended to use the following schema (there is, however, no restriction preventing from choosing another schema - just keep an overview...):

- use port number 7001 for the server manager;
- use port numbers 7002 to 7999 for rasdaman servers.

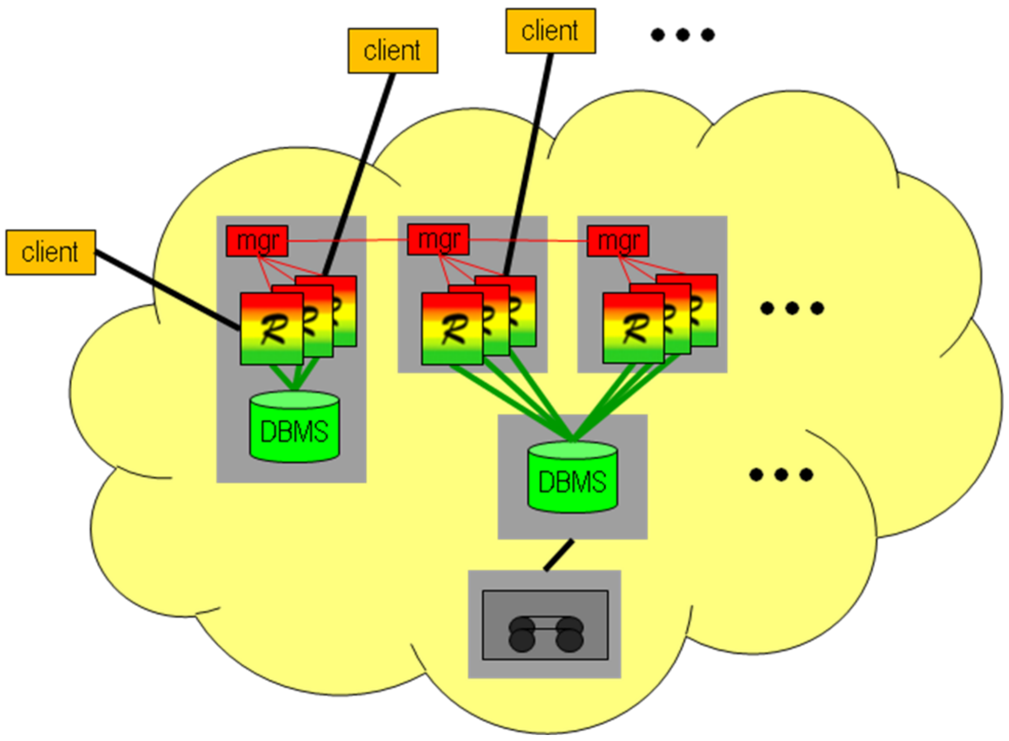


Figure 2.3: rasdaman federation

2.5.3 Storage backend

rasdaman can store array data in two different ways:

1. Arrays in a file system directory, array metadata in SQLite; this is default.
2. Everything in PostgreSQL: arrays in BLOBs, array metadata in tables.

Note: rasdaman enterprise additionally supports access to pre-existing archives of any structure.

The array storage variant can be chosen during the cmake configuration step (cf. Table 2.1) by setting `-DDEFAULT_BASEDB=sqlite|postgresql` when installing from source; it is fixed in the packages to `sqlite`, i.e. the default recommended option.

Storing arrays in a file system directory

In this storage variant, a particular directory gets designated to hold rasdaman arrays (maintained by rasdaman) and their metadata (maintained by an SQLite instance embedded in rasdaman).

The recommended directory location is `$RMANHOME/data/`; administrators may configure this to be a symbolic link to some other location, possibly another filesystem than where `$RMANHOME` resides (so as to keep programs and data separate). Alternatively, the path can be changed in the `-connect` option in `rasmgr.conf`.

The data directory will contain the named database. Currently only one database is supported, but this may change in future. Default database name, assumed by all tools, is `RASBASE`. While it can be changed this is not recommended as all tools will need to receive an extra parameter indicating the changed name.

The database name needs to be communicated to rasdaman in the `$RMANHOME/etc/rasmgr.conf` configuration file. Specifically, the connect string should be an absolute path to the `RASBASE` database (note that variables are not recognized in the script, therefore `$RMANHOME` has to be spelt out). Assuming the default values described above and a rasdaman installation directory of `$RMANHOME=/opt/rasdaman`, the corresponding configuration line might look like this:

```
define dbh rasdaman_host -connect /opt/rasdaman/data/RASBASE
```

Caution: For a customized data directory location it is recommended to use a symbolic link, rather than modify installation defaults.

Storing arrays in PostgreSQL BLOBs (deprecated)

In this storage variant, rasdaman arrays and their metadata are stored in a PostgreSQL database.

First, install and *configure PostgreSQL for use with rasdaman*. To deploy rasdaman for using PostgreSQL make sure to configure with `-DDEFAULT_BASEDB=postgresql`.

In `$RMANHOME/etc/rasmgr.conf` the connect string should be the name of the `RASBASE` database, e.g:

```
define dbh rasdaman_host -connect RASBASE
```

The `create_db.sh` script sets this automatically. It is recommended to keep this value because otherwise this name has to be changed in many places across multiple clients and scripts.

2.6 Server Administration

This Section explains on how to manage a rasdaman service on a *lower level*: start up and shut down individual server workers, as well as how to monitor and influence server state.

It is recommended to first study the previous section so as to understand server administration terminology used here.

2.6.1 General Procedure

`rasmgr` vs. `rascontrol`

It is important to distinguish between the manager, `rasmgr`, and its control front-end, `rascontrol`. The manager runs as a background process, supervising activity of local (and possibly remote) rasdaman servers. Interaction between user (i.e., administrator) and the manager takes place through the interactive control front end.

In the sequel, it is first described how to launch the manager `rasmgr`, then `rascontrol` commands are detailed.

Important Security Note

To remain compatible with older rasdaman versions, clients use login "rasguest" / password "rasguest" by default (i.e., when no user and password are explicitly set by the application). In the distribution configuration, this user is defined to have read-only access to the databases, so that users can access but not manipulate databases without authentication.

Therefore, the administrator is strongly urged to adapt authentication settings to the local security policy before switching databases online.

See *Users and Their Rights* to learn more about user management mechanisms.

2.6.2 Running the Manager

Manager Startup

Starting up the rasdaman system is done by invoking the rasdaman manager, `rasmgr`, from a shell under the `rasdaman` operating system login. Usually the manager will be sent to the background:

```
$ rasmgr &
```

Starting `rasmgr` is the only direct action to be done on it. Any further administration is performed using `rascontrol`.

Note that, unless a server configuration has been defined already, no rasdaman server is available just by starting the manager.

Invocation Synopsis

Manager invocation synopsis:

```
$ rasmgr [--help] [--hostname h] [--port p]
```

where

--help	print this help
--hostname h	host on which the manager process is running is accessible under name / IP address *h* (default: output of Unix command hostname)
--port p	manager will listen to port number *p* (default: 7001)

Examples

To start a manager which will listen at port 7001:

```
$ rasmgr --port 7001
```

2.6.3 `rascontrol` Invocation

The manager front end, rascontrol, is a command-line interface used for rasdaman administration. It allows to define the whole rasdaman system configuration, including start up and shut down of server instances and user logins and rights.

To secure access to the server administration facilities, rascontrol performs a login process requesting login name and password similar to the Unix rlogin command. User name must be one of the users defined in the rasdaman authentication list (see *Users and Their Rights*).

`rascontrol` Synopsis

```
$ rascontrol [-h|--help] [--host *h*] [--port *n*] [--prompt *n*]
             [--quiet]
             [--login|--interactive|--execute *cmd*|--testlogin]
```

where

--host h	name of the host where the manager runs (default: localhost)
-h, --help	this help
--port n	port number at which the manager listens to requests (default: 7001)
--prompt n	change rascontrol prompt as follows: • `0` - prompt '>'

- `1` - prompt '`rasc>`'
- `2` - prompt '`user:host>`'

(default: 2)

--quiet	quiet, don't print header (default for `--login` and `--testlogin`)
--login	print login and password, obtained from interactive input, to `stdout`, then exit (see *Script Use* below)
--interactive	read login and password from environment variable `RASLOGIN` instead of requesting it interactively
--execute cmd	execute single `*cmd*` and exit (batch mode); all text following `-x` until end of line is passed as `command`; this option implicitly assumes `-e`
--testlogin	just do a login and nothing else to check whether the login/password combination provided in the `RASLOGIN` variable is valid

Interactive Use

In interactive use, `rascontrol` will be invoked with the host parameter only. Following successful authentication, `rascontrol` accepts command line input from `stdin`.

Here is an example session (`mypasswd` will not be echoed on screen):

```
$ rascontrol
Login name: *mylogin*
Password: *mypasswd*
mylogin:localhost> define dbh h1 -connect /
mylogin:localhost> define db d1 -dbh h1
mylogin:localhost> define srv s1 -host localhost
-type h -dbh h1
mylogin:localhost> up srv s1
mylogin:localhost> save
mylogin:localhost> exit
$
```

Script Use

Alternatively to interactive login, user and password information can be taken from the environment variable `RASLOGIN`. This variant is suitable for batch scripting in conjunction with the `-x` option.

The following example shows how first the `RASLOGIN` is set appropriately:

```
$ export RASLOGIN=`rascontrol --login`
```

...and then a sample Unix shell script which starts all rasdaman servers defined in the system configuration, performing implicit login from the environment variable contents which has been obtained from the previous command and pasted into the shell script:

```
#!/bin/bash
export RASLOGIN=rasadmin:mytotallyencryptedpassword
rascontrol -x up srv -all
```

Comments in Scripts

To enhance legibility of scripts, `rascontrol` accepts comments in the usual shell syntax: Lines beginning with a hash sign '#' will be ignored, whatever they may contain. An example is usage in shell *here documents* (type `man sh` in your favourite shell for further information on this feature):

```
$ rascontrol <<EOF
# this is the command submitted to rascontrol:
list srv -all
# now terminate rascontrol:
exit
# the following line terminates rascontrol input:
EOF
$
```

2.6.4 rascontrol Command List

Command Synopsis

`help`	display information (general or about specific command)
`exit`	exit `rascontrol`
`list`	list info about the current status of the system
`up`	start server(s)
`down`	stop rasdaman server(s) or server manager(s)
`define`	define a new object
`remove`	remove an object
`change`	change parameters of objects
`save`	make configuration changes permanent

In the remainder of this section, commands are explained in detail, sorted by the targets they affect.

2.6.5 Server Hosts

Define Server Hosts

```
define host h -net n -port p
```

h symbolic host name

-net n set network host name to *n*

-port p port on which the rasdaman manager will listen

Change Server Host Settings

```
change host h [-name n] [-net x] [-port p]
            [-uselocalhost [on|off] ]
```

h host name whose entry is to be updated

-name n change host name to *n*

-net x change network name to *x*

-port p change port number to *p*

-uselocalhost [on|off] use domain name localhost (IP address 127.0.0.1) instead of regular network host name; usually this speeds up communication a little (default: `on`)

Note that it is not possible to change network name or port for a host while this server is running.

uselocalhost works only for the master manager and is on by default. This means that the servers running on manager master host should

Remove Server Host Definitions

```
remove host h
```

h host name whose entry is to be deleted

Remove host `h` from the definition table.

It is not possible to remove a host definition while the corresponding host has active servers.

Status Information

```
list host
```

List all hosts currently defined.

2.6.6 rasdaman Servers

Define rasdaman Servers

```
define srv s -host h -type t -port p -dbh d
    [-autorestart [on|off] [-countdown c]
    [-xp options]
```

s a unique, not yet used name for the server

-host h name of the host where the server will run

-type t communication type: `t` is `r` for RPC, `h` for http

-port p the RPC *program number* for RPC servers (recommended: 0x2999001 - 0x2999999), TCP/IP port for http servers (recommended: 7002 - 7999)

-dbh d database host where the relational database server to which the rasdaman server connects will run

-autorestart a for *a*=``on``: automatically restart rasdaman server after unanticipated termination for *a*=``off``: don't restart (default: *a*=``on``)

-countdown *c for *c*>``0``: restart rasdaman server after c requests for *c*=``0``: run rasdaman server indefinitely (default: *c*=``1000``)

-xp options pass option string *options* to server upon start (default: no options, i.e., empty string)

Option `-xp` must be the last option. Everything following "-xp" until end of line is considered to be "*options*" and will be passed, at startup time, to the server.

Change Server Settings

```
change srv s [-name n] -type t [-port p] [-dbh d]
        [-autorestart [on|off] [-countdown c]
        [-xp options]
```

s change settings for server *s*

-name n change server name to *n*

-port p change port number to *p*

-dbh d new database host where the relational database server runs to which the rasdaman server connects

-autorestart a for *a*=on: automatically restart rasdaman server after unanticipated termination for *a*=off: don't restart

-countdown c for *c*>0: restart rasdaman server after c requests for *c*=0: run rasdaman server indefinitely

-xp options pass option string *options* to server upon start

Option `-xp` must be the last option. Everything following "-xp" until end of line is considered to be "*options*" and will be passed, at startup time, to the server.

Restrictions:

- The server host cannot be changed.
- The server name cannot be changed while the server is up.
- The new settings will be used only next time the server starts.

Remove rasdaman Server Definitions

```
remove srv s
```

s server name whose entry is to be deleted

Remove server *s* from the definition table.

It is not possible to remove a server definition while the corresponding server is up and running

Status Information

```
list srv [ s | -host h | -all ] [-p]
```

s give information about server *s*

-host h give information about all servers running on host *h* information is requested

-all list information about all servers on all hosts (default)

-p additionally list configuration information

The first is variant prints status information of the currently defined server(s); if *s* is provided, then only server s is listed.

2.6.7 Database Hosts

Define Database Hosts

```
define dbh h [-connect c]
```

h a unique symbolic database host name, usually the host machine name

-connect c the connection string used to connect `rasserver` to the database server

-user u the user name (optional) used to connect `rasserver` to the base DBMS server; for PostrgreSQL, using this parameter automatically implies trust authentication.

-passwd p the password (optional) used to connect `rasserver` to the base DBMS server; for PostrgreSQL, using this parameter automatically implies trust authentication.

Change Database Host Settings

```
change dbh h [-name n] [-connect c]
```

h database host whose entry is to be changed

-name n change symbolic database host name to *n*

-connect c change connect string to *c*

-user u the user name used to connect `rasserver` to the base DBMS server; using this optional parameter automatically implies ident-based authentication.

-passwd p the password used to connect `rasserver` to the base DBMS server; using this optional parameter automatically implies ident-based authentication.

The connection parameters can be changed at any time, however the servers will get the information only when they are restarted.

Remove Database Host Definitions

```
remove dbh h
```

h database host name whose entry is to be deleted

Remove database host *h* from the definition table.

It is not possible to remove a database host definition while this database host has active servers connected to it.

Status Information

list dbh List all relational database hosts currently defined.

2.6.8 Databases

Databases represent the physical database itself, together with the relational database server accessing them. It is possible to have multiple database definitions in the rasdaman server environment which are distinguished by the database host; the interpretation, then, is that the same contents (be it the same physical database or a mirrored copy) is available through relational servers running on the different hosts mentioned. In other words, when a client opens a database, the server manager can freely choose any of the database hosts on which the database indicated is defined.

The pair (database,database host) must be unique.

Define Databases

```
define db d -dbh db
```

d define database with name *d*

-dbh db set database host name to *db*

Change Database Settings

```
change db d -name n
```

d database whose name is to be changed

-name n change to new database name *n*

Remove Database Definitions

```
remove db d -dbh db
```

d name of database to be removed

-dbh db host name of database to be removed

Remove definition of database *d* from the definition table. The database itself remains unchanged, it is not physically deleted.

It is not possible to remove a database definition while the corresponding database has open transactions.

Status Information

```
list db [ d | -dbh h | -all ]
```

d give information about servers connected to database *d*

-dbh h give information about all servers connected to database *d* via database host *h*

-all list information about all servers connected to any known database (default)

List relational database(s) defined.

2.6.9 Server Start-up and Shutdown

Server Start

```
up srv [ s | -host h | -all ]
```

s start only server *s*

-host s start all servers on host *h*; this requires that a manager has been started on this host previously.

-all start all servers defined; note that only those servers can be started on whose host a manager is currently running.

Look up the named server(s) in the definition list, and start the specified one(s) using the previously defined individual startup parameters.

At least one of the options *s*, -host *s*, and -all must be present.

Server Shutdown

```
down srv [ s | -host h | -all ] [-force] [-kill]
```

s name of the server to be stopped

-host s terminate all servers on host *h*

-all terminate all servers

-force *send SIGTERM* immediately, don't wait for transaction end

-kill *send SIGKILL* immediately, don't wait for transaction end

This command shuts down the indicated server(s). At least one of the options *s*, -host *s*, and -all must be present.

Without `-force` and `-kill`, the server is marked for shut down and will actually be terminated by sending `SIGTERM` after completing the current transaction. With `-force` and `-kill`, the server is terminated instantaneously; this should be handled with extreme caution, as experience shows that relational database systems react differently on such a situation: usually a running transaction is aborted (which is the desired behavior), but sometimes the running transaction is committed (most likely leaving the database in an inconsistent state). See a Unix manual for the difference between `SIGTERM` and `SIGKILL` signals.

The manager on host *h* is not terminated.

2.6.10 Users and Their Rights

Similarly to operating systems, rasdaman knows named users with access rights associated to them. Each rasdaman client must log in to the system under a specific login name using its specific password; this holds for database clients as well as for database administration. With each login name, a set of rights is associated which determines the set of actions admitted to the user under this login.

To this end, the rasdaman administrator manages user login names (user names) equipped with a password and rights to access the databases.

Attention: There is no way to retrieve a lost password!

The set of known logins as well as the associated rights all are under administrator control; the `define` and `remove` commands serve to add or delete user logins, the `change user` command allows to individually assign rights to a login.

In the rasdaman system's initial state after installation, user `rasadmin` is defined owning all possible rights (see below). A further user `rasguest` is defined which owns read-only access ("R") rights.

For both users, the password initially is identical with the user name. It is highly recommended to change this immediately (See *Change User Attributes*).

Define New User

```
define user u [-passwd p] [-rights r]
```

`u` login name, must be unique (i.e., not yet existing)

`-passwd p` set login password to pass

`-rights r` rights associated with this login

The user's password can be changed at any time (see *Change User Attributes*).

Remove User

```
remove user u
```

`u` login name to be removed

The user is removed from the login list and henceforth cannot login to the rasdaman system any more.

User Rights

User rights are indicated by upper case letters. They are divided into two categories: *system rights* and *database rights*. System rights apply to the whole system configuration of a server machine, whereas database rights can be specified individually for a database.

The following system rights are defined:

`C` user may change the system configuration

`A` access control: the user may perform user management

`S` start/stop right: the user may start and stop the system, in particular: rasdaman servers

`I` info retrieval: the user may retrieve server status information

The following database rights are defined:

`R` user is allowed read data (select...from...where) from rasdaman databases

W user is granted write access (update, insert, delete) to rasdaman databases

Notation of Rights

In the `change user` command used for user rights administration, a user's rights set is described by a *rights string*. It is built from letters denoting the rights to be granted.

To revoke a right, leave out the corresponding character. To grant no rights at all, use - (minus sign).

No blanks or other characters are allowed in a rights string.

Examples of valid rights strings are:

- grant all rights: `CASIRW`
- grant read access only: `R`
- grant no rights at all: `-`

These are examples for *invalid* rights strings:

- Blanks between rights: `CA SIR W`
- Invalid characters I: `AXYZS`
- Invalid characters II: `A_+S`

Change User Attributes

```
change user u [-name n | -passwd p | -rights r]
```

Options:

u user login to be updated

-name n change user name to *n*

-passwd p change password to *p*

-rights r change rights of user *u* according to rights string *r*

Change name of user, login password, or user rights.

Status Information

```
list user [-rights]
```

-rights additionally list rights assigned to each user

List all user names currently defined, optionally with their rights.

2.6.11 Server Control Options

The following options can be passed to the server when it is started by the server manager using the `up srv` command. Option settings are defined for a particular server using the `rascontrol` command `change srv -xp` which passes the rest of the line after `-xp` on to the server upon starting it (see *rasdaman Servers*).

--log logfile	print log to *logfile*. If *logfile* is stdout, then log output will be printed to standard output. (default: `$RMANHOME/log/rasserver.`*serverid.serverpid*.log)
--timeout t	client time out in seconds for sign-of-life signal. If no t indicated: 300 sec; if set to 0, no sign-of-life check is done.
--transbuffer b	set maximum size of transfer buffer to *b* bytes (default: 10 MB = 10000000 bytes)
--cachelimit c	specifies upper limit in bytes on using memory for caching (default: 0)
--enable-tilelocking	perform tile-level locking on insert / update / delete (default: whole database is locked)

2.6.12 Distributed Query Processing

Rasdaman can form a federation network for query answering. In such a setup, `rasmgrs` facing congestion (i.e., all `rasserver` worker processes busy) will try to acquire a free server from some other `rasmgr`'s holding in the federation.

Session-based server assignment

As always in rasdaman, acquisition and release of server processes is done on session level: when a client opens a new connection, it gets a server assigned; when it closes the connection, this server is released and put back into the pool of available processes. Hence, for optimal load balance clients should strive to have short-running sessions and not keep open connections unduly for a long time.

Federation network

The federation network is defined in a decentralized way: each `rasmgr` knows peers from which it accepts requests, and to which it can send requests. To this end, each `rasmgr` maintains an `inpeer` and `outpeer` list:

- The `inpeer` list contains those hosts from which this node's `rasmgr` will accept requests.
- The `outpeer` list contains those hosts which this node's `rasmgr` will ask for server processes on local session overflow.

By manipulating these two lists administrators can exercise fine-grain security policy in a rasdaman federation network.

Note that the federation connectivity graph is not necessarily symmetric: a `rasmgr` may send requests to some other `rasmgr`, but not accept requests, and vice versa, depending on the individual configuration.

Each host individually respects these statements, there is no global rasdaman federation configuration.

Federation node addressing

Addressing is based on hostnames, where a hostname in the sequel is one of

- a domain name, resolvable by this `rasmgr`'s host
- an IP address

All `inpeer` and `outpeer` statements accumulate so that host identifiers can be added and removed incrementally.

Security

A `rasmgr` request for a server process on another host is treated by the incoming host in the same way as any such incoming client request. The requesting `rasmgr` authenticates via the login and password which the originating client used for authenticating against rasdaman in the first place.

This implies that a client approaching such a federation must be known in all federation nodes. See *Users and Their Rights* for details on users and the various permissions they can have on a database.

If neither any `inpeer` nor any `outpeer` is defined (either interactively through `rascontrol` or by way of settings in `rasmgr.conf`) then this rasdaman instance will act completely standalone and will neither send nor accept peer requests.

Define peers

```
define inpeer hostname
```

`hostname` host from which requests for rasdaman server process assignment will be accepted by this rasmgr

```
define outpeer hostname [-port portnumber]
```

`hostname` host from which this rasmgr may request a rasdaman server process

`portnumber` port number at which the rasmgr on that host is listening (default: 7001)

List peers

```
list inpeer

list outpeer
```

These commands list all currently defined inpeers and outpeers, respectively.

Remove peers

```
remove inpeer hostname

remove outpeer hostname
```

These commands remove hostname *hostname* listed from the list of peers.

Examples

```
define inpeer www.acme.com

define inpeer 192.168.28.10
```

Caveat: fluctuating IPs

In cloud environments, IP addresses are maintained dynamically and can change for a given host between reboots. Hence, when growing a rasdaman federation by launching new VMs care must be taken that the in- and outpeers received the proper current IP address.

Restrictions

In the current version, the queries are distributed only if the receiving rasmgr has no locally assigned rasservers. This limitation will be removed in the next release.

2.6.13 Miscellaneous

Help

```
help
```

Display top level help page

```
help [command]
command help
```

Display information specific to *command*

(both syntax variants are equivalent)

Version Information

```
list version
```

version display rasdaman server version.

Save Changes to Disk

```
save
```

The `save` operation writes the current configuration and authorization values to disk. All changes done during the session thus become permanent.

`rascontrol` Termination

```
exit
```

terminates `rascontrol`.

2.7 Security

2.7.1 General

There are several security measures available, which should be considered seriously. Among them are the access right mechanisms found in Tomcat, rasdaman, and PostgreSQL. We highly recommend to make use of these.

For Tomcat and PostgreSQL refer to the pertaining documentation. The servlet is safe against SQL injection attacks - we are not aware of any means for the user to send custom queries to the PostgreSQL server or the rasdaman server. XSRF and XSS represent no danger to the service because there is no user generated content available.

For rasdaman, we recommend to change the default user passwords in rasdaman (rasguest/rasguest for read-only access, rasadmin/rasadmin for read-write and administrator access) to not run into the Oracle “Scott/tiger” trap. Even better, use additional separate, private users. The rasdaman service doesn’t use cookies.

2.8 Backup

Both software and hardware can fail, therefore it is prudent to establish regular backup procedures. For rasdaman in particular a couple of things should be considered for inclusion in a backup:

1. The rasdaman database, which normally can be found in `/opt/rasdaman/data`. The SQL database itself in this directory, `RASBASE`, is fairly small; the `TILES` subdirectory may be large as it contains all the array data, but if backup disk space is not scarce then it is definitely recommended to backup as well. Incremental backups of the `TILES` with rsync for example should work well without unnecessary duplicated data copying, unless existing data areas are often updated. Example with rsync:

   ```
   # backup small RASBASE to /backup/rasdamandb
   rsync -avz /opt/rasdaman/data/RASBASE /backup/rasdamandb/

   # backup potentially large TILES dir to /backup/rasdamandb
   rsync -avz /opt/rasdaman/data/TILES /backup/rasdamandb/
   ```

2. The petascopedb geo metadata database is usually small and worth backing up. By default it is stored in PostgreSQL and can be extracted into a small compressed archive as follows:

   ```
   # create backup in a gzip archive petascopedb.sql.gz
   sudo -u postgres pg_dump petascopedb | gzip > /backup/
   ↪petascopedb.sql.gz
   ```

   If necessary, it can be restored with

   ```
   # if a petascopedb already exists it needs to be renamed, as␣
   ↪otherwise
   # restoring over an existing petascopedb will corrupt it
   sudo -u postgres psql -c "ALTER DATABASE petascopedb RENAME TO␣
   ↪petascopedb_existing_backup"

   # create an empty petascopedb
   sudo -u postgres createdb petascopedb

   # restore backup petascopedb.sql.gz (use cat if it's not a gzip␣
   ↪archive)
   zcat petascopedb.sql.gz | sudo -u postgres psql -d petascopedb -
   ↪-quiet > /dev/null
   ```

   Alternatively, if the above fails for some reason, petascopedb can be backup with this command:

   ```
   # note that /backup/petascopedb_backup will contain a large␣
   ↪number of compressed files
   sudo -u postgres pg_dump -j 8 -Fd petascopedb -f /backup/
   ↪petascopedb_backup
   ```

If necessary, it can be restored with

```
sudo -u postgres pg_restore -j 8 -d petascopedb /backup/
↪petascopedb_backup
```

3. The rasdaman configuration files in `/opt/rasdaman/etc`, but also consider the `bin` and `share` directories which may be useful in case of package update problems, as well as maybe log files in the `log` directory.

```
# backup everything except the data dir, which is handled in␣
↪step 1. above
rsync -avz --exclude='data/' /opt/rasdaman /backup/
```

2.9 Migration

Sometimes it is necessary to migrate the installation from one machine (*OLD*) to another (*NEW*). This section outlines the steps on how to do this.

1. Make sure rasdaman is installed and functional on the NEW machine.
2. Stop rasdaman and an external tomcat if installed on both the OLD and NEW machine, e.g:

```
sudo service rasdaman stop
sudo service tomcat9 stop
```

3. Make a backup of the rasdaman and petascope databases on the OLD machine by following step 1. of the *backup guide* and copy the backup to the NEW machine.
4. Restore the database backups on the NEW machine by following step 2. of the *backup guide*.
5. Make a backup of the config files on the NEW machine (`/opt/rasdaman/etc`), then copy relevant configuration from the OLD to the NEW machine, in particular:
   - `rasmgr.conf` can probably copied as is, but check if the -host setting is correct for the NEW machine;
   - most settings from `petascope.properties` can be copied as is, except the ones for database configuration (spring.* and metadata*);
   - `/etc/default/rasdaman` can be copied as is usually;
7. Make sure that the `/opt/rasdaman` directory is owned by the `rasdaman` user, to avoid any permission issues caused by copying with other system users:

```
sudo chown -R rasdaman: /opt/rasdaman
```

8. Finally start rasdaman and tomcat:

```
sudo service rasdaman start
sudo service tomcat9 start
```

2.10 Uninstallation

When uninstalling rasdaman, you can execute the following commands to ensure that all installed files and services are fully removed from the system.

Note: These instructions are only applicable if rasdaman was installed from package or with the rasdaman installer.

```
# Stop rasdaman, disable service, and clear from failed list
$ sudo systemctl stop rasdaman
$ sudo systemctl disable rasdaman
$ sudo systemctl reset-failed

# Removes everything related to rasdaman (except dependencies)
# on Ubuntu/Debian systems; for CentOS use `yum erase rasdaman`
$ sudo apt-get purge rasdaman

# Remove the rasdaman installation directory
# WARNING: This removes all rasdaman data and configuration files!
$ sudo rm -r /opt/rasdaman

# Remove systemd service, init script, profile script
$ sudo rm /etc/systemd/system/rasdaman.service
$ sudo rm /etc/init.d/rasdaman
$ sudo rm /etc/profile.d/rasdaman.sh

# Remove rasdaman apt repository on Ubuntu/Debian
$ sudo rm /etc/apt/sources.list.d/rasdaman.list

# Remove dependencies of the rasdaman package which are no longer
# needed by other packages; it will remove postgres if no other
# package depends on it (Debian / Ubuntu)
$ sudo apt-get autoremove
```

2.11 Troubleshooting

2.11.1 General

The first step in troubleshooting problems should be to look into the *server logs*.

Start with checking the `rasmgr` and `rasserver` logs for any errors. If this does not provide any clues, check the `petascope.log` or `catalina.out`.

Next, investigate the status of rasdaman and external Tomcat if applicable with `systemctl rasdaman status` (and similar for Tomcat). Inspect the output of `ps aux | grep ras` to list details about the rasdaman processes, or `top` for CPU and memory usage.

It can be useful to double check the system memory usage with `free -m`, and disk space usage with `df -h`.

2.11.2 Manually stop rasdaman

If stopping rasdaman fails, it may be necessary to manually stop it:

```
# check the rasdaman processes still running on the system
$ ps aux | grep ras

# force kill any rasmgr process; <pid> is the number in the 2nd␣
↪column
# of the output from the previous command
$ kill -9 <pid>

# then try to kill any remaining rasserver processes
$ pkill -f rasserver

# if this fails (check with ps aux), force kill rasservers
$ pkill -9 -f rasserver
```

Checking the server logs could provide further information on why stopping rasdaman failed in the first place.

CHAPTER

THREE

CONTRIBUTING TO RASDAMAN

We'd like to encourage everyone to contribute to the project in whatever ways they can. This is a volunteer project; all help is greatly appreciated.

We don't have a rigid styleguide or set of rules for contributing to rasdaman; however, with the goal of making things easier for all developers, here are a few suggestions:

- **Get feedback from other people**

  This is what the Mailing Lists and this Trac project management system are for. It is always a good idea to talk to other devs on the mailing lists before submitting changes.

- **Use Trac tickets**

  This is important to track progress and activity. If you start working on an issue, accept the ticket. After you have finished, close the ticket, and add information about the changeset in the comment field. Provide also progress information when you make relevant changes as described in the UseOfTickets page

- **Write tests**

  Please write tests for any new functionality. See RasdamanTestSuites for instructions. We ask you for your understanding that patches are likely to get rejected if they do not contain adequate additions to the *systemtest*.

- **Stick to the Coding Standards**

  The rasdaman code guide is mandatory for all code. We ask you for your understanding that patches are likely to get rejected if they do not adhere to this guide.

- **Use meaningful commit messages and reference tickets**

  such messages help developers to understand what your goal and intent is. Further, it eases writing of release notes for new versions. Note that patches not starting with "ticket:nnn" will be automatically rejected.

3.1 Development Contributions

You developed a fix or some new functionality? We are grateful for your contribution. Of course we like any useful information, but best (and fastest) for inclusion is to be in sync with our development tools and processes. The following details are provided to help in this respect.

1. All our development is in Linux. Please consider this for your code.
2. We use *git* as a version management tool, so you may want do do that too. Check out from the repository using:

   ```
   $ git clone git://rasdaman.org/rasdaman.git
   $ git config --global user.name "Name Surname"
   $ git config --global user.email my_email@address.xyz
   ```

3. After ensuring the tests are successful (see *Adding Tests*), stage and commit your changes:

   ```
   $ git add <file1> <file2> <dir1/> <dir2>/*.java ...
   $ git commit -m "ticket:NNNN - [FIX|NEW] My brief explanation␣
   ↪of the patch"
   ```

   NNNN indicates the number of the ticket that is addressed with this patch, [FIX] should be set when the patch contributes a bug fix, or alternatively [NEW] should be used when the patch introduces a new feature.

4. Prepare your patch package through:

   ```
   $ git format-patch -n
   ```

   where `n` is the number of last commits that you want to create patch files for.

5. Upload your patch file (or a `.tar.gz` archive in case of several files) using Patch Manager to the default `master` branch. You will have to accept the Contributor Agreement. Without your stated consent we unfortunately cannot accept it, due to legal reasons.
6. If your patch is a [FIX], reupload it again by selecting the `release_X.Y` branch for the latest X.Y rasdaman version, e.g. `release_10.0`. It may be necessary to first rebase the commit to this branch in your local repository clone.

3.2 Documentation

Any changes to public interfaces likely require updating the rasdaman documentation. This is a short guide on how to do this.

3.2.1 Getting started

1. Install dependencies

```
$ sudo pip install -U sphinx sphinx_rtd_theme
```

2. Main documentation can be found in `doc/main` (`*.rst` files).
3. There are two ways to rebuild the docs when you have made changes to them.
   1. In the build directory of rasdaman (see relevant documentation), the documentation can be built if you have configured rasdaman with `-DGENERATE_DOCS=ON` argument for cmake:

      ```
      $ make doc        # generate all documentation
      $ make doc-html   # generate HTML documentation (requires␣
      ↪sphinx)
      $ make doc-pdf    # generate PDF documentation (requires␣
      ↪sphinx, latexmk, texlive)
      ```

      The generated documentation can be found in

      - HTML - `<build_dir>/doc/main/html/index.html`
      - PDF - `<build_dir>/doc/main/latex/rasdaman.pdf`

   2. Alternatively, in the rasdaman source tree:

      ```
      $  cd doc/main/
      $ ./build.sh       # generate all documentation
      $ ./build.sh html # generate HTML documentation (requires␣
      ↪sphinx)
      $ ./build.sh pdf   # generate PDF documentation (requires␣
      ↪sphinx, latexmk, texlive)
      ```

      Note that in this case you should have executed cmake in the build directory with `-DGENERATE_DOCS=ON` at least once (see relevant documentation); this will generate `doc/main/conf.py` which is required to run `./build.sh` above.

      The generated documentation can be found in

      - HTML - `<source_dir>/doc/main/_build/html/index.html`
      - PDF - `<source_dir>/doc/main/_build/latex/rasdaman.pdf`

Make changes

- Check the short intro below for the reST syntax
  - … but it should be fairly clear from looking at the docs sources
- Create a review request with *arc diff* before pushing changes.

3.2.2 Quick intro to reStructuredText

Section headers

In each case the underline or overline marker should be as long as the section header (use monospace font to do this correctly). From highest level to most granular section level:

1. `#` - Parts (overline and underline)
2. `*` - Chapters (overline and underline)
3. `=` - Sections (underline)
4. `-` - Subsections (underline)
5. `^` - Subsubsections (underline)

Example from the QL guide:

```
####################
Query Language Guide
####################

************
Introduction
************

Multidimensional Data
=====================

Subsection
----------

Subsubsection
^^^^^^^^^^^^^
```

Text formatting

```
*Italics*
**Bold**
``Code``
```

Cannot be nested, may not start/end with whitespace, and has to be separated from surrounding text with some non-word characters.

Lists

Bullet and number lists are supported:

```
* Bulleted list
* Item two

  * Nested list (note it has to have blank line before and after!)

- Bulleted list continues; you can use - instead of *

1. Numbered list
2. Item two

#. Automatically numbered list
#. Item two


term (single line)
    Definition of the term (indented on the next line)

    Definition continues with another paragraph (maintain␣
↪indentation)


| Line block
| line breaks are preserved
| and appear exactly like this (without the | characters)
```

Option lists (e.g. the output of `rasql -h`) can be simply copy pasted, you just need to make sure the options and their descriptions form two columns.

Source code

Any source code can go as an indented text after `::` (plus blank line). In the QL guide `::` automatically does rasql highlighting. For example:

```
::

    -- example query
    select avg_cells(c) from mr2 as c
```

renders as

```
-- example query
select avg_cells(c) from mr2 as c
```

For different highlighting you have to use the code-block directive indicating the language, e.g. java, cpp, xml, javascript, text, ini, etc. Example for java:

```
.. code-block:: java

    public static void main(...) {
        ...
    }
```

Furthermore, you can specify code blocks that are foldable, and hidden or shown by default (see extension documentation; starthidden is True by default and can be omitted):

```
.. hidden-code-block:: java
   :starthidden: False

    public static void main(...) {
        ...
    }
```

shows as

```
public static void main(...) {
    ...
}
```

You can see all lexers with `pygmentize -L lexers`; see also http://pygments.org/languages/

Images

If an image has no caption then use the image directive, e.g:

```
.. image:: media/logo_full.png
    :align: center
    :scale: 50%
```

If it has a caption then use the figure directive; the caption is added as an indented paragraph after a blank line:

```
.. _my-label:

.. figure:: media/logo_full.png
    :align: center
    :scale: 50%

    Caption for the figure.
```

Hyperlinks

To just have a URL as is nothing special needs to be done, just put as is:

```
http://rasdaman.org
```

To render the URL with alternative text, then the following form should be used:

```
`Link text <http://rasdaman.org>`_
```

Internal cross-referencing can be done by first setting up a label before a section header or a figure (see above this section Hyperlinks) and then using it to generate a link anywhere with

```
:ref:`my-label`
```

Instead of :ref: you can use :numref: to get automatic Figure number added to the link, e.g.

```
:numref:`my-label` -> Sec. 2
```

You can change the default text that :ref: generates like this:

```
:ref:`Custom text <my-label>`
```

Further information

- Specification: http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
- Sphinx guide: http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html

3.3 Git resources

- For extensive help on *git* see the online Git book.
- For info on git *conflicts* see Handling and Avoiding Conflicts in Git or, for a quick resolve conflict by discarding any local changes, this StackOverflow answer.

Further tips:

- Cleaning local history
- Dealing with rejected patches
- Git bundles
- . . .

3.3.1 Basic git for working on tickets

It is suggested to create a branch in your local working copy of the rasdaman git repo for each ticket/fix, so you will not mix up patches. (e.g: ticket:1450 -> branch ticket_1450, ticket:1451 -> branch ticket_1451, . . .)

Prerequisites

1. Checkout the newest source code from repository; suppose you did this in `/home/rasdaman/rasdaman` and you are in this directory in the terminal:

   ```
   $ pwd
   /home/rasdaman/rasdaman
   ```

2. List the branches in your local repository

   ```
   $ git branch
   ```

3. Switch to branch master - as this branch is the canonical branch for the rasdaman remote repository

```
$ git checkout master
```

4. Pull the newest patches if possible from remote repository (rasdaman.org) to your local repository

   ```
   $ git pull
   ```

5. Create a new branch from master branch for a particular fix or feature work:

```
$ git checkout -b "branch_name" # e.g: git checkout -b "ticket_
↪1451"

# check current branch, it should be ticket_1451
$ git branch
```

Work and commit changes

1. You changed some files in the source code directory (e.g: file1.java, file2.cc,…) and you want to create a commit; first *stage* the changed files:

```
$ git add file1.java file2.cc ..
```

Warning: Avoid doing `git add .`, i.e. adding all changed files automatically.

2. Now you are ready to commit the staged files:

```
$ git commit -m "ticket:1451 - fix some stuff"

# see details of your commit on top
$ git log
```

3. And create a patch from the commit, i.e. a file with extension `.patch` created from the last commit = `-1`, which contains all the changes you made:

```
$ git format-patch -1
# or for code review
$ arc diff
```

3. After your diff is reviewed and accepted, finish with this branch by uploading the patch to the patchmanager and switching to another ticket in a new branch, starting from master again.

Switch between pending patches

E.g you finished one ticket on ticket_1450 and uploaded to the patchmanager but the patch is rejected and needs to be updated, while you moved on to working on ticket_1460.

1. First, stage everything you are doing on ticket_1460; if you don't want to create a temporary commit, you can just stash everything in current branch.

```
$ git add <file1> <file2> ...

# or stash
$ git stash
# later can be retrieved with
$ git stash pop
```

2. Then commit it as your pending patch on this branch

```
$ git commit -m "ticket:1460 - fixed stuff"
```

3. Make sure your current branch is clear

```
# should report: "nothing to commit, working directory clean"
$ git status
```

4. Now switch to your failure patch (e.g: ticket_1450):

```
$ git checkout ticket_1450
```

5. Fix the issues here and stage the newly changed files:

```
$ git add <file 1> <file 2> ...
```

6. Commit it without changing the ticket's subject:

```
$ git commit --amend --no-edit
```

7. Create a patch from the updated commit:

```
$ git format-patch -1

# or for code review
$ arc diff
```

8. And upload it again to the patchmanager
9. Finally, you can switch back to the previous branch:

```
$ git checkout ticket_1460
```

Apply patches between branches

E.g you have 1 commit in ticket_1450 and 1 commit in ticket_1460) then you want to add this patch to ticket_1460)

1. Check current branch (should be ticket_1450)

```
$ git branch
```

2. Create a patch file (like "0001-ticket-1450-fix-some-issues.patch") from the last commit

```
$ git format-patch -1
```

3. Switch to other branch

```
$ git checkout ticket_1460
```

4. Apply your patch from ticket_1450

```
$ git am -3 0001-ticket-1451-fix-some-issues.patch
```

5. Check the newest commit (if the patch is applied successfully)

```
$ git log
```

If a patch cannot be applied

1. You made changes on files which the patch also changes, so you have to merge it manually:

```
$ git am -3 0001-ticket-1450-fix-some-issues.patch
# The patch is not applied, some conflict shows here
```

2. Please follow our git conflict resolution guide, or Steps 3 to 7 of this resolving merge conflicts guide.

3. Once resolved, mark as such:

```
$ git am --resolved
```

4. Check that your patch from ticket_1450 is now the last patch in ticket_1460 branch:

```
$ git log
```

3.4 C++ Guidelines

The rasdaman system is implemented in C++ 11; below are some guidelines.

3.4.1 IDE

Developing C++ code is no doubt a much better experience with good IDE support. Below we give some guidelines.

Qt Creator

Qt Creator is very well suited for hacking on rasdaman: it's free and open-source, very fast and low on resource usage compared to alternatives, has built-in debugging support, code formatter, and other useful features.

Below you see a video that walks through setting up Qt Creator on Ubuntu 18.04. A short transcript:

- install Qt Creator with `sudo apt install qtcreator`
- start Qt Creator

- we assume rasdaman was cloned into `~/rasdaman`, and create a build dir at `~/rasdaman-build`
  - run CMake to configure the build, see ref:*sec-system-install* for more info
  - note: it is recommended to create the build directory outside the source tree, to avoid confusing the IDE with build artefacts
- go to `File -> Open File or Project` in Qt Creator, navigate to the rasdaman source tree and select `CMakeLists.txt`
- this will lead to importing the `~/rasdaman-build` directory from which Qt Creator will figure out the build configuration; for development we normally want a Debug config
- with that the rasdaman project is now ready for development in Qt Creator
- some of the features are then shown in the video:
  - start building rasdaman (same as `make`) from within Qt Creator with `Ctrl+B`
  - navigate to a file with `Ctrl+k` (there are a lot more options for quick navigation)
  - `Ctrl+left mouse click` leads to the declaration of a symbol
  - `F4` will switch between header and source files
  - in `Projects (left sidebar) -> Run` you can configure the executable to be executed when running the project; normally we want this to be rasserver, and we specify the arguments like the query to be executed by rasserver
  - `F5` will then start the debugger on the configured executable; you can set up breakpoints as usual to stop the debugger at certain locations or conditions

Video: https://doc.rasdaman.org/_images/setup_qtcreator.gif

3.4.2 Debugging

The rasdaman code has facilities built in which aid debugging and benchmarking. On this page information is collected on how to use it. Target audience are experienced C++ programmers.

Important: It is best to configure rasdaman with `-DCMAKE_BUILD_TYPE=Debug` for debugging, and `-DCMAKE_BUILD_TYPE=Release` for benchmarking (and production deployment).

Debuging rasserver

In *rasnet* (the default network protocol), in order to attach to the rasserver process (with e.g. gdb -p <pid>) it is necessary to increase the values of SERVER_MANAGER_CLEANUP_INTERVAL and CLIENT_MANAGER_CLEANUP_INTERVAL in rasmgr/src/constants.hh to some large values; needless to say this requires recompiling and restarting rasdaman.

Once that is done, you can attach to a running rasserver process. First find the process id, second column in the output of

```
$ ps aux | grep rasserver
```

It's best to enable only one rasserver in rasmgr.conf or with rascontrol for this purpose. Then, attach to the pid:

```
$ gdb -p <pid>
```

Debugging directql

When not debugging the network protocol, it's recommended to use directql. directql has the same interface as rasql, with an important behind the scenes difference: it is a fully fledged rasserver itself actually, so it doesn't need to go through the client protocol. This makes it ideal for running tools like gdb, valgrind, etc.

When executing directql, use the same parameters as for rasql, but add -d /opt/rasdaman/data/RASBASE (or substitute that to whatever is the -connect value in rasmgr.conf).

Example with gdb:

```
$ gdb --args directql -q 'query that causes a segfault' \
                      --out file -d /opt/rasdaman/data/RASBASE
...
> run
...
# show a backtrace once the segfault has happened
> bt
```

Memory debugging with valgrind

Valgrind can be used to detect uninitialized values, memory errors, and memory leaks, e.g.

```
$ valgrind --leak-check=full --track-origins=yes \
           directql -q 'query that causes memory problems' \
                    --out file -d /opt/rasdaman/data/RASBASE
```

Memory debugging with AddressSanitizer

AddressSanitizer can be enabled during compilation with `-DENABLE_ASAN=ON`. This adds `-fsanitize=address` to the compiler flags. Please visit the ASAN page for more details.

Enabling extra output at compile time

In order to effect any extra output (besides standard logging) at all, the code must be compiled with the resp. option enabled. This is not default in production operation for at least two reasons: writing an abundance of lines into log files slows down performance somewhat, and, additionally, logging has a tendency to flood file systems; however, the option is available when needed.

If you are compiling with cmake, simply use `-DENABLE_DEBUG=ON` before doing make. Doing this includes the above cmake flags for debugging, and it also sets two other variables to enable more-verbose logging. E.g. in your build directory

```
$ cmake .. -DCMAKE_INSTALL_PREFIX=$RMANHOME -DCMAKE_BUILD_
↪TYPE=Debug -DENABLE_DEBUG=ON ...
$ make
$ make install
```

You may, optionally, alter settings in $RMANHOME/etc/log-client.conf and $RMANHOME/etc/log-server.conf to enable various other logging parameters, e.g. DEBUG and TRACE for extra verbose output in the logs.

3.4.3 Internal array representation

Internally in rasdaman, multidimensional arrays are handled as a 1-D array, linearized in column-major format. Column-major refers to matrices with rows and columns, indicating that first all cells of the first column are listed in order, then all cells of the second row, etc. Given that we are working with multidimensional arrays here, this notion needs to be generalized: the inner-most (last) axis is contiguous, and varies fastest, followed by the second last axis and so on.

For example, let's say we have an array with sdom `[5:10, -2:2, 0:5]`. The 1-D internal_array (in code) corresponds to external_array (in rasql):

```
linear_index := 0
for i := 5..10
  for j := -2..2
    for k := 0..5
      internal_array[linear_index] == external_array[i, j, k]
      linear_index += 1
```

3.5 Adding Tests

TODO: this is somewhat outdated and incomplete.

The rasdaman source tree comes with integration tests ("systemtest" for historical reasons) and unit tests (in each component directory `X` there is a subdirectory `X/test/`). To run the integration test:

```
$ cd systemtest
$ make check
```

After your patch submission, the patchmanager will automatically run the systemtest in a sandbox; the result will be flagged in the patchmanager table for each patch submitted. Allow some time (usually 1.5 hours) until the result gets visible. Patches which do not pass systemtest will be rejected without further inspection.

`make check` will automatically find all tests in the four test case directories, specifically, testcases_mandatory, testcases_petascope, testcases_fixed and testcases_open.

1. whenever a bug is found, a corresponding test should be created in the testcases_open directory;
2. when the bug is fixed, the test should be moved to the testcases_fixed directory;
3. testcases_services holds the test cases for petascope and secore;
4. testcases_mandatory holds the test cases for rasql typically.

Each test should have a folder which is inside one of the above mentioned directories, by convention named `test_X`, e.g. `test_select`. The test should be executed by a shell script inside the folder; its exit code indicates whether the test passed (0) or failed (non-0). Details of the test execution should be logged in the same folder. In `systemtest/util` there are various bash utility functions that can be used in the test scripts, e.g. for logging, checking result, etc.

3.5.1 Add a rasql test query

1. save the test query as `systemtest/test_mandatory/test_select/queries/<queryName>.rasql`
2. save the expected query result file in `systemtest/test_mandatory/test_select/oracle/<queryName>.oracle`

To generate a test oracle:

1. if the result is a scalar, run

   ```
   rasql -q  "<query>" --out string | grep Result > <queryName>.
   ↪oracle
   ```

2. if the result is an array, run

```
rasql -q  "<query>" --out file --outfile <queryName>.oracle
```

Make sure to validate the correctness of the oracle before adding to the systemtest.

If a query is *known to fail* and documented by a ticket, it can be marked in the systemtest, so that the result of that query is *SKIPPED*, rather than *FAILED*. To do this create a file `known_fails` (if not yet existing) in the corresponding test dir (next to the `test.sh`) and put each query file name in a single line in this file.

3.5.2 Add a petascope test

The scripts for WMS, WCS and WCPS testing can be found respectively in:

- `rasdaman/systemtest/testcases_services/test_wcps`
- `rasdaman/systemtest/testcases_services/test_wcs`
- `rasdaman/systemtest/testcases_services/test_wms`

To run a specific test (besides `make check` that runs the whole systemtest), go to the directory and execute

```
$ ./test.sh
```

Do **not** execute `sh test.sh` as the script is written for bash, and `sh` is often linked to a restricted version of bash like dash, or similar. Variables like Tomcat port, host, `rasdaman` connection details, etc. may need to be adapted before running the tests by editing `rasdaman/systemtest/conf/test.cfg`.

Testdata

Various coverages are inserted when running `make check` with the `wcst_import` test suite in `testcases_services/test_all_wcst_import/test.sh`. These are subsequently available in the WCPS, WCS, and WMS tests. At the testing end, they are removed by running `testcases_services/test_zero_cleanup/test.sh`.

Adding tests

To add new tests to the test suite, simply add new WCS or WCPS queries to the `queries` directory. Please adhere to the naming convention, continuing from the last number:

Type	File name format
WCS KVP	`number-meaningful_name.[error.]kvp`
WCS XML	`number-meaningful_name.[error.]xml`
WCS SOAP	`number-meaningful_name.[error.]soap`
WCS REST	`number-meaningful_name.[error.]rest`
WCPS	`number-meaningful_name.[error.]test`
WCPS XML	`number-meaningful_name.[error.]xml`
rasql	`number-meaningful_name.[error.]rasql`

Note: If the test is meant to raise an exception, add a further `.error` suffix to the file name before its extension, for both query and oracle.

The associated oracle (.oracle) files must also be added to the `oracle/` directory. The oracle can be automatically added by running the tests. In this case it can be more convenient to run the tests on the single new query by specifying the query file name as an argument of `test.sh` (e.g. `./test.sh 001-query.test`).

3.5.3 Templated System Test

The rasdaman query templating engine **rasqte** (currently found in `systemtest/testcases_manual/test_rasql`) allows to write template queries (in Jinja2 format) that focus on the operation that should be tested; a preprocessing step expands these templates into concrete valid queries targeting various data configurations that can be evaluated in rasdaman.

In Jinja2 templates we have:

- **Output markup** (surrounded in `{{` and `}}` which resolves to text; this supports some basic arithmetic, string functions, etc.
- **Tag markup** (surrounded in `{%` and `%}` which doesn't resolve to text, and can be used for loops, conditionals, etc.
- **Comments** - any text surrounded in `{#` and `#}`

Comprehensive documentation on Jinja2 templates can be found in the official template designer documentation.

The templating engine defines several global objects/variables that can be used in the query templates. The table below documents these objects; the Example column shows an example for 2-dimensional char data.

Important: the templating engine iterates over all dimensions in *dimension_list* and cell types in *cell_type_name_list*, and renders the template for each pair.

Variable	Description	Example	Default
Dimensionality			
dimension_max	Max dimension tested	4	4
dimension_list	All tested dimensions		[1, 2, 3, 4]
dimension	Curr. dimension	2	one of dimension_list
Cell type			
cell_type_name_list	All tested cell types		[boolean, octet, char, ushort, short, ulong, long, float, double, complex, complexd, char_char_char short_float]
cell_type_name	Curr. cell type	char	one of cell_type_name_list
cell_type_suffix_dic	Cell type -> constant suffix		{'octet': 'o', 'char': 'c', . . . }
cell_type_suffix	Curr. cell type suffix	c	cell_type_suffix_dic[cell_type_name
cell_type_min_dic	Cell type -> min value		{'octet': '-128', 'char': '0', . . . }
cell_type_min	Min for curr. cell type	0	cell_type_min_dic[cell_type_name]
cell_type_max_dic	Cell type -> max value		{'octet': '127', 'char': '255', . . . }
cell_type_max	Max for curr. cell type	255	cell_type_max_dic[cell_type_name]
cell_type_val_dic	Cell type -> non-edge value		{'octet': '-13', 'char': '33', . . . }
cell_type_val	Value for curr. cell type	33	cell_type_val_dic[cell_type_name]
cell_type_size_dic	Cell type -> cell size (B)		{'octet': 1, 'char': 1, . . . }
cell_type_size	Size for curr. cell type	1	cell_type_size_dic[cell_type_name]
cell_type_signed_dic	Cell type -> is signed		{'octet': True, 'char': False, . . . }
cell_type_signed	Is curr. cell type signed	False	cell_type_signed_dic[cell_type_name

continues on next page

Table 3.1 – continued from previous page

Variable	Description	Example	Default
cell_type_components_dic	Cell type -> comp. name/type pairs		{'char_char_char': [('b0','char') ('b1','char'),('b2','char')], ...}
cell_type_components	Components for curr. cell type	[]	cell_type_components_dic[cell_type_
Coll/Mdd type			
coll_name_dic	(dim,cell type) -> coll name		{(1,'octet'): 'test_1d_octet', (2,'char'): 'test_2d_char_set', ...}
coll_name	Curr. coll name	test_2d_char	'test_' + dimension + 'd_' + cell_type_name
coll_type_name_dic	(dim,cell type) -> coll type		{(1,'octet'): 'test_1d_octet_set', (2,'char'): 'test_2d_char_set', ...}
coll_type_name	Curr. coll type name	test_2d_char_set	coll_name + '_set'
mdd_type_name_dic	(dim,cell type) -> mdd type		{(1,'octet'): 'test_1d_octet_mdd', (2,'char'): 'test_2d_char_mdd', ...}
mdd_type_name	Curr. mdd type name	test_2d_char_mdd	coll_name + '_mdd'
MDD constants			
mdd_constant_cell_count	Number of cells in mdd constants	16	16
mdd_constant_extents_list	dimension -> sdom extents		[[], [16], [4,4], [2,2,4], [2,2,2,2]]
mdd_constant_extents	dim extents for curr. dim	[4,4]	mdd_constant_extents_list[dimensior

continues on next page

Table 3.1 – continued from previous page

Variable	Description	Example	Default
mdd_constant_sdom_list	dimension -> sdom		[‘’, ‘[0:15]’, ‘[0:3,0:3]’, ‘[0:1,0:1,0:3]’, ‘[0:1,0:1,0:1,0:1]’]
mdd_constant_sdom	sdom for curr. dim	‘[0:3,0:3]’	mdd_constant_sdom_list[dimension]
mdd_constant_cell_values_dic	cell type -> cell values		{‘char’: [‘0c’,’0c’,’255c’,’1c’, ‘99c’,’9c’,’109c’,’2c’,’5c’,’12c’, ‘23c’,’45c’,’123c’,’123c’, ‘234c’,’250c’], …}
mdd_constant_cell_values	cell values for curr. cell type	[‘0c’,’0c’, ‘255c’,’1c’, ‘99c’,’9c’, ‘109c’,’2c’, ‘5c’,’12c’, ‘23c’,’45c’, ‘123c’,’123c’, ‘234c’,’250c’]	mdd_constant_cell_values_dic[cell_t

mdd_constant_dic	(dim,cell type) -> mdd		{(2,’char’):
			‘<[0:3,0:3]
			0c,0c,255c,1c;
			99c,9c,109c,2c;5c,12c,23c,45c;
			123c,123c,234c,250c>’,
			…}

continues on next page

Table 3.1 – continued from previous page

Variable	Description	Example	Default
mdd_constant	mdd constant for curr. dim/cell type	‘<[0:3,0:3] 0c,0c,255c,1c; 99c,9c,10c,2c; 5c,12c,23c, 45c;123c,123c, 234c,250c>’	mdd_constant_dic[(dimension,cell_ty
Operations			
oper_induced_unary	Unary induced ops		[+,-,not]
oper_induced_unary_name	Unary induced op names		[plus,minus,not]
oper_induced_binary	Binary induced ops		[+,- ,*,/,overlay,is,and,or,xor, =,<,>,<=,>=,!=,]
oper_induced_binary_name	Binary induced op names		[plus,minus,multiplication,division, overlay,is,and,or,xor, equals,less,greater,lessorequal, greaterorequal,notequal]
oper_condense_op	Condense operators		[+,*,and,or,max,min]
oper_condense_name	Condense op names		[plus,multiplication,and,or,max,min]
Functions			
func_induced_unary	Unary induced functions		[sqrt,abs,exp,log,ln,sin,cos,tan, sinh,cosh,tanh,arcsin,asin, arccos,acos,arctan,atan]
func_induced_binary	Binary induced functions		[pow,power,mod,div,bit, max,min,complex]

continues on next page

Table 3.1 – continued from previous page

Variable	Description	Example	Default
func_condense	Condensers		[max_cells,min_cells,all_cells, some_cells,count_cells,add_cells, avg_cells,var_pop,var_samp, stddev_pop,stddev_samp]
Other			
separator	Instantiation sep.		'==='
template_name	Template file name		e.g. 'setup' (extension is removed)
test_id	Unique test id		template_name + "_" + dimension + "d_" + cell_type_name

In addition, the following functions can be used in the templates:

Function	Description	Example
is_atomic_cell_type(type)	return true if type is an atomic cell type	is_atomic_cell_type('char') -> True
is_complex_cell_type(type)	return true if type is a complex cell type	is_complex_cell_type('complexd') -> True
is_composite_cell_type(type)	return true if type is a composite cell type	is_composite_cell_type('short_float') -> True

Template instantiation

The template instantiation engine is a script `rasqte.py` that takes a template file as an input and produces a concrete output file.

```
usage: rasqte.py [-h] [-t TEMPLATE] [-d OUTDIR] [-s SEPARATOR] [-g]

rasql query template engine takes a Jinja2 template file as an␣
↪input and
renders it into a concrete output; various global variables and␣
↪functions are
available in the template (see option -g and the documentation).␣
↪The template
```

(continues on next page)

(continued from previous page)

```
is rendered multiple times for different variable configurations;␣
↪each output
is appended to the same output file in the directory specified with␣
↪-d,
separated by a line with a unique separator string (=== by default).

optional arguments:
  -h, --help            show this help message and exit
  -t TEMPLATE, --template TEMPLATE
                        Template file to be rendered; the output␣
↪should be
                        multiple lines of the form key:value, e.g.
                        query:SELECT version(); Consult the␣
↪documentation for
                        more details.
  -d OUTDIR, --outdir OUTDIR
                        Directory for output files ('.' by default).
  -s SEPARATOR, --separator SEPARATOR
                        Separator for different renderings of the␣
↪same
                        template ('===' by default).
  -g, --globals         Print all global variables/functions.
```

Rendered templates

The rendered **concrete file** will have many instantiations of one template. Each instantiation ends with a separator line (=== by default):

```
instantiated_query
===
instantiated_query
===
...
===
```

Each `instantiated_query` has this format: The `id` is used to compare the result of evaluating the query to an *oracle* file named `id`. `id` and `query` are mandatory, any other parameters are optional.

```
query: concrete rasql query (mandatory)
id: unique id (mandatory)
filter: python string or list of strings that remove matching
        lines from the output (usually output that contains random
        bits which cannot be compared to a fixed expected oracle)
disableoutfile: comment on why --out file should be removed from
                the generated rasql command (e.g. to skip comparing
                random output, or output that is really large)
```

(continues on next page)

(continued from previous page)

```
timeout: set how many seconds to wait for the test to finish,
         before killing the process; 60 seconds by default
knownfail: comment on why this query currently fails; once fixed,
           a line like this should be removed
skip: comment on why this query should be completely skipped during
      the test (i.e. not evaluated at all); should be removed once␣
↪fixed
...
key: value
# comment lines start with '#'

# empty lines (as above) are ignored as well
===
```

Concrete test evaluation

A systemtest script `test.py` then reads a concrete file and evaluates the tests, comparing to the expected oracle values.

```
usage: test.py [-h] [-d] [-t TESTSFILE]

rasql query systemtest evaluator; without arguments it evaluates␣
↪all tests in
the queries/ directory, starting with any setup tests and ending␣
↪with the
teardown tests.

optional arguments:
  -h, --help            show this help message and exit
  -d, --drop            Drop data (execute teardown queries) only␣
↪and exit.
  -t TESTSFILE, --testsfile TESTSFILE
                        Execute a specific tests file (with setup␣
↪before and
                        teardown after).
```

The following directories are used by the script:

- `queries` - contains the rendered templates (outputs of `rasqte.py`)
- `outputs` - results of evaluating tests in queries are saved in this directory. For rasql queries two file types are saved:
  1. any file outputs (produced by rasql as `--out file` is specified) in `template name.file*` files
  2. stdout, stderr, and exit code from running the program in a `template name` file
- `oracles` - similar structure as `outputs` directory, it contains the expected files against which the outputs are compared.

`setup` and `teardown` tests files in `queries` dir are treated specially: `setup` is evaluated first, before any others, and `teardown` is evaluated at the end. This allows to import data for the test, and drop it at the end, for example.

Examples

Create collection

Template:

```
CREATE COLLECTION {{ coll_name }} {{ coll_type_name }}
```

Engine instantiates 4 x 13 = 52 queries (4 for dimensions [1, 2, 3, 4] and 13 for cell types [boolean, char, octet, short, unsigned short, long, unsigned long, float, double, complex, complexd, char_char_char, short_float]:

```
rasql -q 'CREATE COLLECTION test_1d_boolean test_1d_boolean_set' --
↪out file
rasql -q 'CREATE COLLECTION test_1d_char test_1d_char_set' --out␣
↪file
rasql -q 'CREATE COLLECTION test_1d_octet test_1d_octet_set' --out␣
↪file
...
rasql -q 'CREATE COLLECTION test_4d_short_float test_4d_short_float_
↪set' --out file
```

Insert literal array

Template:

```
INSERT INTO {{ coll_name }} VALUES {{ mdd_constant }}
```

Instantiation:

```
rasql -q 'INSERT INTO test_1d_boolean VALUES ...' --out file
rasql -q 'INSERT INTO test_1d_char VALUES ...' --out file
...
rasql -q 'INSERT INTO test_3d_RGBPixel VALUES ...' --out file
```

The `mdd_constant` is constructed of 16 values that contain edge values (min/max) and other “interesting” values, like 0, nan, inf, etc.

Select: sin(array)

Template:

```
select sin(c) from {{coll_name}} as c
```

Instantiation:

```
rasql -q 'select sin(c) from test_2d_char as c' --out file
...
```

Select: sin(scalar)

Template:

```
select sin( {{cell_max}}{{cell_type_suffix}} )
```

Instantiation:

```
rasql -q 'select sin( 255c )' --out file
...
```

Select: all binary induced ops

Template:

```
{%- set dimension_other = [dimension + 1] if dimension < dimension_
↪max else [] -%}
{%- for dimension_right in [dimension] + dimension_other -%}
{%-   for cell_type_right in cell_type_name_list -%}
{%-     for op in oper_induced_binary -%}
{%-       set coll_name_right = coll_name_dic[(dimension_right,␣
↪cell_type_right)] -%}
SELECT a {{ op }} b FROM {{ coll_name }} AS a, {{ coll_name_right }}
↪ AS b
{%      endfor -%}
{%-   endfor -%}
{%- endfor -%}
```

Instantiation:

```
rasql -q 'select a + b from test_2d_char as a, test_2d_char as b' --
↪out file
rasql -q 'select a + b from test_2d_char as a, test_2d_octet as b' -
↪-out file
rasql -q 'select a + b from test_2d_char as a, test_2d_ushort as b'␣
↪--out file
...
```

Drop data

Template:

```
drop collection {{coll_name}}
```

Instantiation:

```
rasql -q 'drop collection test_2d_char as a' --out file
...
```

Insert encoded data

TODO

Template:

```
---
data.file: ../testdata/data_{{dimension}}d_{{cell_type_name}}.tif
data.dimension: [2]
---
insert into {{coll_name}} values decode($1)
```

Instantiation:

```
rasql -q 'insert into test_2d_char values decode($1)'
      -f ../testdata/data_2d_char.tif --out file
...
```

3.5.4 Running tests locally

Uploading a patch to the patchmanager will automatically trigger a jenkins build that applies the patch on master and runs the systemtest. The jenkins testing is done on several OS in parallel: CentOS 7, Ubuntu 16.04 and Ubuntu 18.04.

You can run the systemtest directly on your installation. But sometimes it can be useful to replicate the same environment as in the jenkins tests, as the patch may be causing some discrepancies across different OS. The steps below show how to replicate the jenkins test on your machine; required dependencies are rasdaman installer and vagrant.

```
# get rasdaman installer bootstrap script
wget https://download.rasdaman.org/installer/install.sh
# download the rasdaman installer in current dir
bash install.sh -d && mv /tmp/rasdaman-installer .
# in the test directory there are helper scripts and Vagrantfiles␣
↪for running tests
cd rasdaman-installer/test
```

(continues on next page)

(continued from previous page)

```
# the vm on which to test; can be centos7 or ubuntu1604 as well
vm=ubuntu1804
# the patch id to be fetched from the patchmanager
# download the patch beforehand to find out the id
patch_id=4000
branch=master

# run the systemtest; the last argument is the command which will␣
↪be executed
# in the /vagrant/rasdamaninstaller directory on the VM
./run_vm.sh "$vm" 1 "./ci_test.sh systemtest.toml $branch $patch_id"

# see local output
less "/tmp/rasdamaninstaller_test/$vm.log"

# see detailed installer output in the vm itself
cd "$vm"
vagrant ssh
less /tmp/rasdaman.install.log
```

The directory where the Vagrantfile sits is mounted in the VM at path `/vagrant`. The `./run_vm.sh` script copies the installer in this directory, so it's possible to update the installer code or `profiles/test/systemtest.toml` file to apply a local patch file for example instead of specifying a patch id to be downloaded from the patchmanager. After such changes, it is necessary to reload the VM in order to get the updated files in its `/vagrant` dir. This can be done with `vagrant reload --provision` in the VM directory (where the Vagrantfile is). Example follows below.

```
cd rasdaman-installer/test
vm=ubuntu1804
branch=master
patch_local="$HOME/patches/mypatch.patch"

# copy the local patch to the vm dir
cp "$patch_local" "$vm/"

# edit the toml file to set path to the vm patch, or do it with sed:
profile="../profiles/test/systemtest.toml"
patch_vm="/vagrant/mypatch.patch"
sed -i 's|patch = .*|patch = "'"$patch_vm"'"|' "$profile"

# reload the vm to make sure the updated profile and the patch␣
↪appear in /vagrant
cd "$vm"
./run.sh reload

# run the systemtest as usual, just don't specify a $patch_id now
# (we already set a path in the profile)
```

(continues on next page)

(continued from previous page)

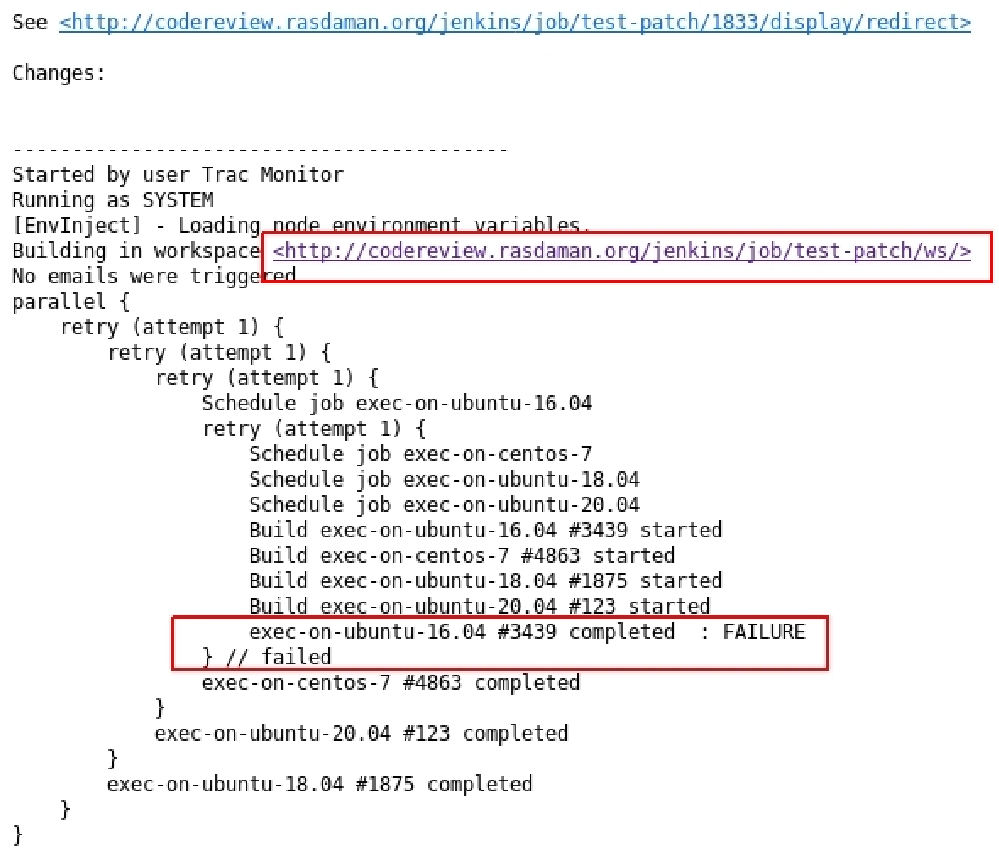
```
cd ..
./run_vm.sh "$vm" 1 "./ci_test.sh systemtest.toml $branch"
```

Instead of testing a single VM, it's possible to run the testing in parallel on all supported VMs (centos7, ubuntu1604, ubuntu1804). It's similar steps as before, except instead of `./run_vm.sh` we execute `./test_installer.sh`; the argument is the command which will be executed in the `/vagrant/rasdaninstaller` directory on each VM:

```
./test_installer.sh "./ci_test.sh systemtest.toml $branch $patch_id"
```

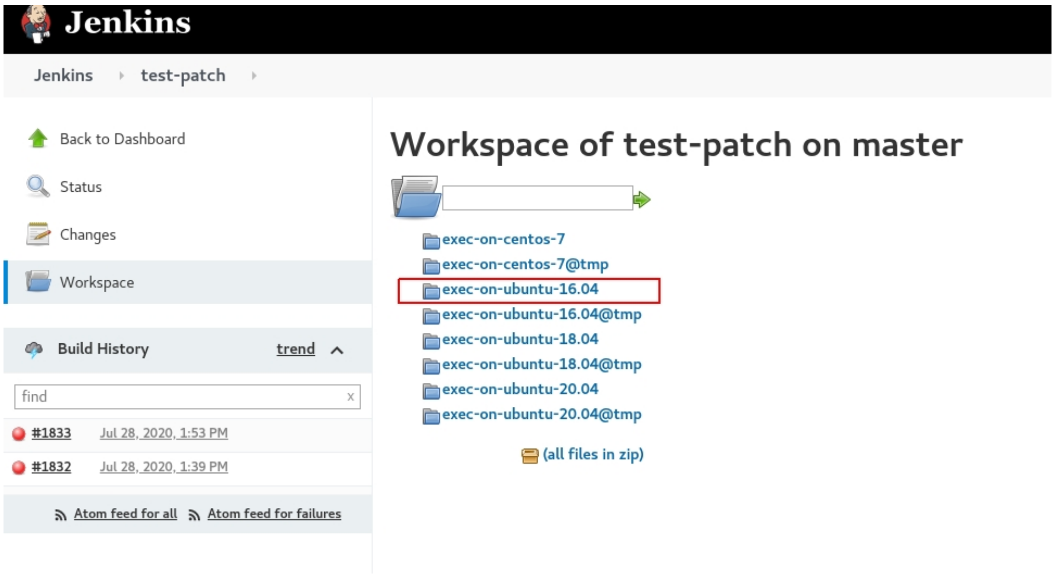
3.5.5 Check Jenkins log files

When you upload a patch to the patchmanager, the systemtest is automatically executed on several different virtual machines (VM) on Jenkins. If a patch failed on a VM, Jenkins will send an email to the patch's author, for example: the patch below failed on `Ubuntu 16.04 VM` with build number `3439`:

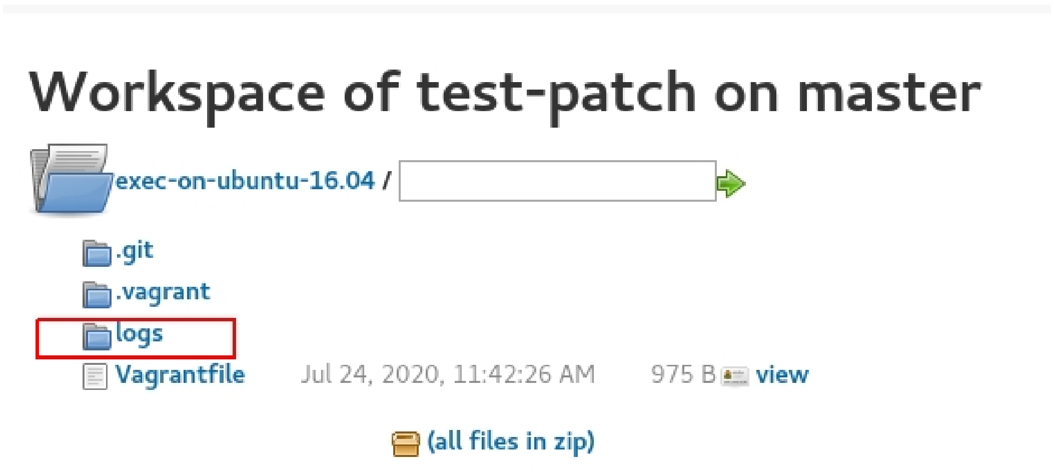


```
See <http://codereview.rasdaman.org/jenkins/job/test-patch/1833/display/redirect>

Changes:


------------------------------------------
Started by user Trac Monitor
Running as SYSTEM
[EnvInject] - Loading node environment variables.
Building in workspace <http://codereview.rasdaman.org/jenkins/job/test-patch/ws/>
No emails were triggered
parallel {
    retry (attempt 1) {
        retry (attempt 1) {
            retry (attempt 1) {
                Schedule job exec-on-ubuntu-16.04
                retry (attempt 1) {
                    Schedule job exec-on-centos-7
                    Schedule job exec-on-ubuntu-18.04
                    Schedule job exec-on-ubuntu-20.04
                    Build exec-on-ubuntu-16.04 #3439 started
                    Build exec-on-centos-7 #4863 started
                    Build exec-on-ubuntu-18.04 #1875 started
                    Build exec-on-ubuntu-20.04 #123 started
                    exec-on-ubuntu-16.04 #3439 completed  : FAILURE
                } // failed
                exec-on-centos-7 #4863 completed
            }
            exec-on-ubuntu-20.04 #123 completed
        }
        exec-on-ubuntu-18.04 #1875 completed
    }
}
```

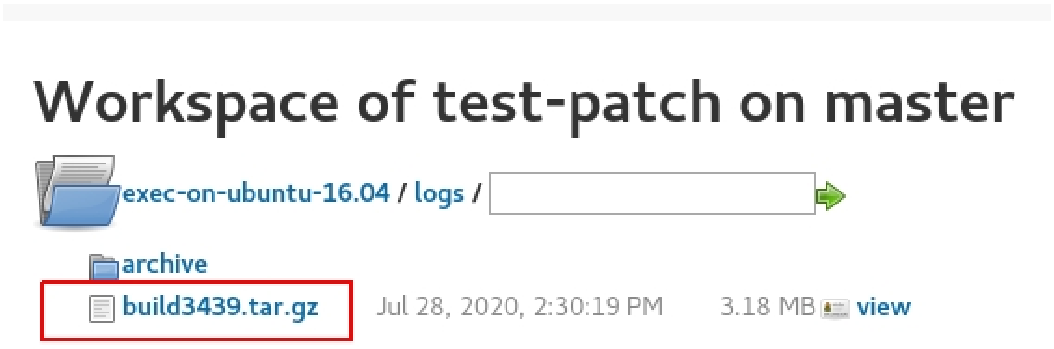
Then, one needs to access Jenkins at the link in the email (`http://codereview.rasdaman.org/jenkins/job/test-patch/ws/`) and click on the failed `Ubuntu 16.04 VM`:



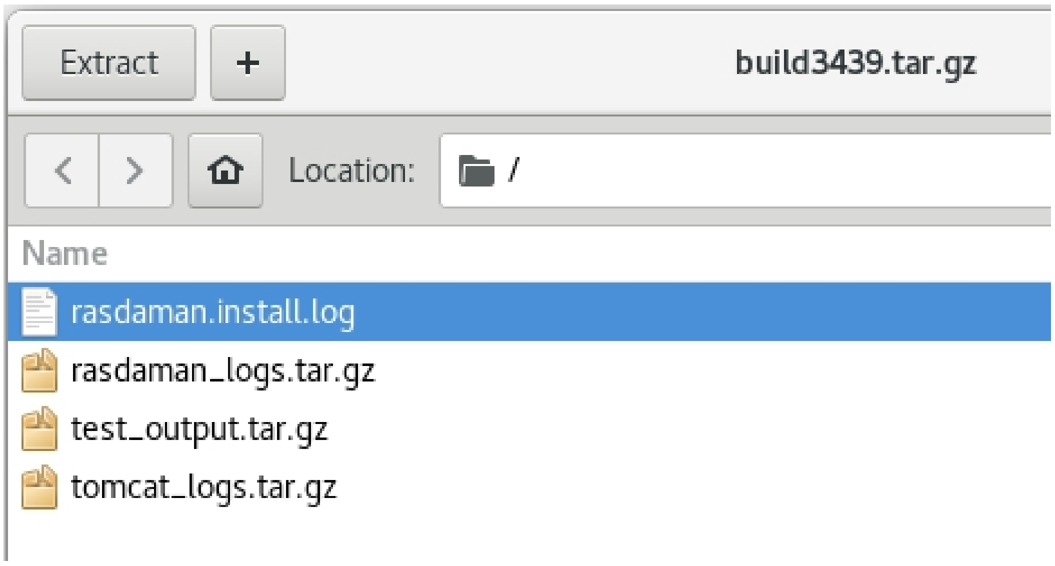
Select the `logs` folder:


Workspace of test-patch on master
exec-on-ubuntu-16.04 /
.git
.vagrant
logs
Vagrantfile Jul 24, 2020, 11:42:26 AM 975 B view
(all files in zip)

and download the compressed file with the failed build number `3439` to the local system:

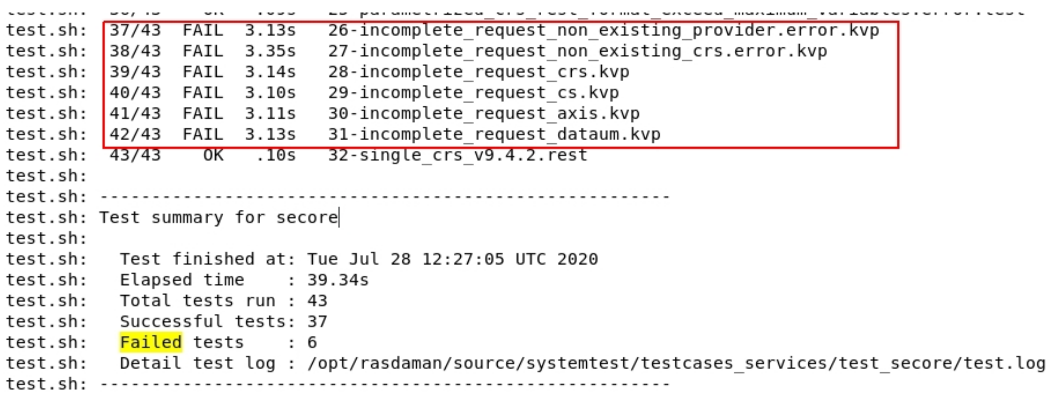

Workspace of test-patch on master
exec-on-ubuntu-16.04 / logs /
archive
build3439.tar.gz Jul 28, 2020, 2:30:19 PM 3.18 MB view

Extract the downloaded archive (`build3439.tar.gz`) and check the `rasdaman.install.log` file:



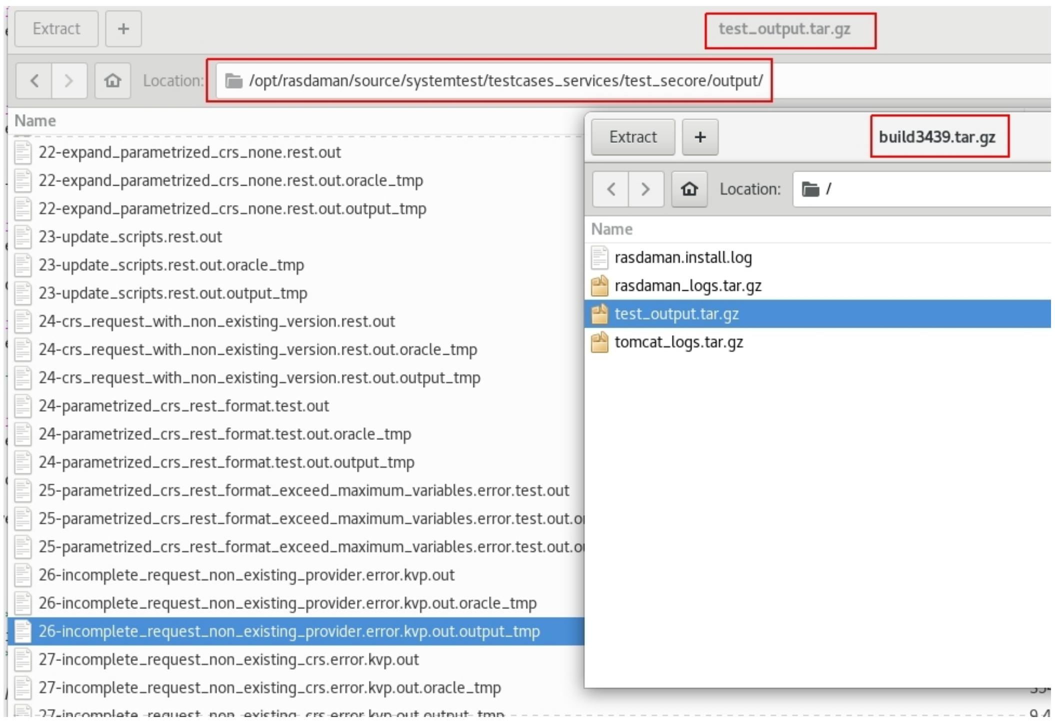
```
rasdaman.install.log
rasdaman_logs.tar.gz
test_output.tar.gz
tomcat_logs.tar.gz
```

Search for `Fail` test cases in `rasdaman.install.log` file, e.g:



```
test.sh:  37/43  FAIL  3.13s   26-incomplete_request_non_existing_provider.error.kvp
test.sh:  38/43  FAIL  3.35s   27-incomplete_request_non_existing_crs.error.kvp
test.sh:  39/43  FAIL  3.14s   28-incomplete_request_crs.kvp
test.sh:  40/43  FAIL  3.10s   29-incomplete_request_cs.kvp
test.sh:  41/43  FAIL  3.11s   30-incomplete_request_axis.kvp
test.sh:  42/43  FAIL  3.13s   31-incomplete_request_datum.kvp
test.sh:  43/43    OK   .10s   32-single_crs_v9.4.2.rest
test.sh:
test.sh: -------------------------------------------------------
test.sh: Test summary for secore
test.sh:
test.sh:   Test finished at: Tue Jul 28 12:27:05 UTC 2020
test.sh:   Elapsed time    : 39.34s
test.sh:   Total tests run : 43
test.sh:   Successful tests: 37
test.sh:   Failed tests    : 6
test.sh:   Detail test log : /opt/rasdaman/source/systemtest/testcases_services/test_secore/test.log
test.sh: -------------------------------------------------------
```

Then, check the ouput of these failed tests in `test_output.tar.gz` archive which is in `build3439.tar.gz` to compare the differences between oracle files and output files:


Extract
+
test_output.tar.gz
Location: /opt/rasdaman/source/systemtest/testcases_services/test_secore/output/
Name
Extract
+
build3439.tar.gz
Location: /
Name

```
22-expand_parametrized_crs_none.rest.out
22-expand_parametrized_crs_none.rest.out.oracle_tmp
22-expand_parametrized_crs_none.rest.out.output_tmp
23-update_scripts.rest.out
23-update_scripts.rest.out.oracle_tmp
23-update_scripts.rest.out.output_tmp
24-crs_request_with_non_existing_version.rest.out
24-crs_request_with_non_existing_version.rest.out.oracle_tmp
24-crs_request_with_non_existing_version.rest.out.output_tmp
24-parametrized_crs_rest_format.test.out
24-parametrized_crs_rest_format.test.out.oracle_tmp
24-parametrized_crs_rest_format.test.out.output_tmp
25-parametrized_crs_rest_format_exceed_maximum_variables.error.test.out
25-parametrized_crs_rest_format_exceed_maximum_variables.error.test.out.o
25-parametrized_crs_rest_format_exceed_maximum_variables.error.test.out.o
26-incomplete_request_non_existing_provider.error.kvp.out
26-incomplete_request_non_existing_provider.error.kvp.out.oracle_tmp
26-incomplete_request_non_existing_provider.error.kvp.out.output_tmp
27-incomplete_request_non_existing_crs.error.kvp.out
27-incomplete_request_non_existing_crs.error.kvp.out.oracle_tmp
```

```
rasdaman.install.log
rasdaman_logs.tar.gz
test_output.tar.gz
tomcat_logs.tar.gz
```

3.6 rasdaman Code Guide

Don't expect others to clean up your code

An open-source project is fun, but it requires a great deal of discipline to make all the code seamless that is coming from the developers worldwide. If everybody just follow their individual coding style - no matter how ingenious the code is - then the whole project will soon become unmaintainable.

To avoid this, rasdaman provides this code guide - don't worry, it contains as few rules as possible, just enough to achieve overall coherence. Although written for C++, *mutatis mutandis* it applies to Java, Javascript, and even scripts.

- *Rules* that have to be fulfilled strictly.
- *Recommendations* which serve as suggestions for a 'better' coding style.
- *Examples* to show how code should be written according to the guidelines.

Please understand that, while we always highly appreciate your contributions, we may have to reject your patch if it breaks this code guide. Your successors looking at the code will be most grateful for your efforts.

Credits: This code guide has been established by the rasdaman team based on the codeguide originally developed by Roland Ritsch who in turn has crafted it along the style guide of ELLEMTEL/Norway. Any eventual error is ours, of course.

3.6.1 Rules

Rule 0: Every time a rule is broken, this must be clearly documented.

—

Rule 1: Include files in C++ must have a file name extension *.hh*.

Rule 2: Implementation files in C++ must have a file name extension *.cc*.

Rule 3: Inline definition files must have a file name extension *.icc*.

Rule 4: Every file must include information about its purpose, contents, and copyright. For this purpose, the several standard headers are provided *here*. Adjust the copyright to your name / instituion as deemed adequate. All code must use a GPL header, except for files in the raslib/, rasodmg/, and rasj/ directories, which must use an LGPL header.

Rule 5: All method definitions must start with a description of their functionality using the standard method header.

Rule 6: All comments must be written in English.

—

Rule 7: Every include file must contain a mechanism that prevents multiple inclusions of the file.

Rule 8: Never use path name in `#include` directives. Only use relative paths and the parent path (..) is not allowed.

Rule 9: Never have indirect inclusion of a function. Collective include files are allowed.

—

Rule 10: The names of variables and functions must begin with a lowercase letter. Multiple words must be written together, and each word that follows the first starts with an uppercase letter (Camel Casing).

Rule 11: The names of constants must be all uppercase letters, words must be separated by underscores ("_").

Rule 12: The names of abstract data types, structures, typedefs, and enumerated types must begin with an uppercase letter. Multiple words are written together and each word that follows the first is begun with an uppercase letter (Camel Casing).

—

Rule 13: The public, protected, and private sections of a class must be declared in that order (the public section is declared before the protected section which is declared before the private section). See the standard class definition for details.

Rule 14: No member functions within the class definition include file. The only exception are inline functions.

Rule 15: No public or protected member data in a class. Use public inline methods (`setVariable()` and `getVariable()`) to access private member data.

Rule 16: A member function that does not affect the state of an object (its instance variables) must be declared const.

Rule 17: If the behavior of an object is dependent on data outside the object, this data must not be modified by const member functions.

—

Rule 18: A class which uses `new` to allocate instances managed by the class must define a copy constructor.

Rule 19: All classes which are used as base classes and which have virtual function, must define a virtual destructor.

Rule 20: A class which uses `new` to allocate instances managed by the class must define an assignment operator.

Rule 21: An assignment operator which performs a destructive action must be protected from performing this action on the object upon which it is operating.

—

Rule 22: A public member function must never return a non-`const` reference or pointer to member data.

Rule 23: A public member function must never return a non-`const` reference or pointer to data outside an object, unless the object shares the data with other objects.

—

Rule 24: Do not use unspecified function arguments (ellipsis notation).

Rule 25: The names of formal arguments to functions must be specified and are to be the same both in the function declaration and in the function definition.

—

Rule 26: Always specify the return type of a function explicitly. If no value is returned then the return type is void.

—

Rule 27: A function must never return a reference or a pointer to a local variable.

Rule 28: Do not use the preprocessor directive `#define` to obtain more efficient code; instead, use inline functions.

—

Rule 29: Constants must be defined using const or enum; never use `#define`.

Rule 30: Do not use numeric values directly in the code; use symbolic values instead (Use constants for default values). Always document the meaning of the value.

—

Rule 31: Variables must be declared with the smallest possible scope. Do not use global variables.

Rule 32: Never declare multiple variables in the same line.

Rule 33: Every variable that is declared must be given a value before it is used.

Rule 34: Don't use implicit type conversions.

Rule 35: Never cast an object to a virtual class.

Rule 36: Never convert a `const` to a non-`const`.

—

Rule 37: The code following a `case` label must always be terminated by a `break` statement.

Rule 38: A `switch` statement must always contain a `default` branch which handles unexpected cases.

Rule 44: Never use `goto`.

—

Rule 45: Do not use `malloc`, `realloc` or `free`, but use new and `delete`. In general, use C++, not C code.

Rule 47: Always provide empty brackets (`[]`) for `delete` when deallocating arrays.

Rule 48: Use C++ exception handling (try/catch) for every possible failure situation.

—

Rule 49: When submitting a patch, describe concisely in the commit message what has been accomplished in the patch. In case of a fix, include in the message the ticket# fixed and place a comment in the source file at the location the fix was done mentioning the ticket (best by its URL).

3.6.2 Recommendations

Recommendation 1: Optimize code only if you know that you have a performance problem. Think twice before you begin.

Recommendation 2: Eliminate all warnings generated by the compiler.

Recommendation 3: An include file should not contain more than one class declaration.

Recommendation 4: Place machine-dependent code in a special file so that it may be easily located when porting code from one machine to another.

Recommendation 5: Always give a file a name that is unique in as large a context as possible.

Recommendation 6: An include file for a class should have a file name of the form + .hh. Use all lowercase letters.

Recommendation 7: Use the directive #include "filename.hh" for user-prepared include files.

Recommendation 8: Use the directive #include for include files from system libraries.

Recommendation 9: Choose names that suggest the usage. Don't give generic names to variables.

Recommendation 10: Encapsulate global variables and constants, enumerated types, and typedefs in a class.

Recommendation 11: Always provide the return type of a function explicitly on a separate line, together with template or inline specifiers.

Recommendation 12: When declaring functions, the leading parenthesis and the first argument (if any) are to be written on the same line as the function name. If space permits, other arguments and the closing parenthesis may also be written on the same line as the function name. Otherwise, each additional argument is to be written on a separate line (with the closing parenthesis directly after the last argument).

Recommendation 13: Always write the left parenthesis directly after a function name (no blanks). Use 'astyle –style=allman -c -n' for autoformatting your code.

Recommendation 14: Braces (`{ }`) which enclose a block are to be placed in the same column as the outer block, on separate lines directly before and after the block. Use indentation of four spaces and don't use tab stops. Use `astyle --style=allman -c -n` for autoformatting your code.

Recommendation 15: The reference operator * and the address-of operator & should be directly connected with the type names in declarations and definitions. Use `astyle --style=allman -c -n` for autoformatting your code.

Recommendation 16: Do not use spaces around `.` or `->`, nor between unary operators and operands. Use `astyle --style=allman -c -n` for autoformatting your code. Got it? ;-)

Recommendation 17: An assignment operator should return a const reference.

Recommendation 18: Use references instead of pointers whenever possible.

Recommendation 19: Use constant references (const &) instead of call-by-value, unless using a pre-defined data type or a pointer.

Recommendation 20: Avoid long and complex functions.

Recommendation 21: Avoid pointers to functions.

Recommendation 22: Pointers to pointers should be avoided whenever possible.

Recommendation 23: Use a typedef to simplify program syntax when declaring function pointers.

Recommendation 24: Always use unsigned for variables which cannot reasonably have negative values.

Recommendation 25: Always use inclusive lower limits and exclusive upper limits.

Recommendation 26: Avoid the use of continue.

Recommendation 27: Do not write logical expressions of the type `if (test)` or `if (! test)` when test is a pointer.

Recommendation 28: Use parentheses to clarify the order of evaluation for operators in expressions.

Recommendation 29: Do not allocate memory and expect that someone else will deallocate it later.

Recommendation 30: Always assign NULL to a pointer after deallocating memory.

Recommendation 31: Check the return codes from library functions even if these functions seem foolproof.

Recommendation 32: If possible, always use initialization instead of assignment. To declare a variable that has been initialized in another file, the keyword extern is always used.

Recommendation 33: Avoid implicit type conversions (casts).

Recommendation 34: Use all flavors of const as often as possible.

3.6.3 Examples

Standard Include Header

```
/*
* This file is part of rasdaman community.
*
* Rasdaman community is free software: you can redistribute it and/
↪or modify
* it under the terms of the GNU General Public License as published␣
↪by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Rasdaman community is distributed in the hope that it will be␣
↪useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
*
* Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
*
* For more information please see <http://www.rasdaman.org>
* or contact Peter Baumann via <baumann@rasdaman.com>.
*/
/*************************************************************
 *
 * PURPOSE:
 *
 * COMMENTS:
 *
 * BUGS:
 *
 ************************************************************/
```

Standard Include Header (LGPL)

```
/*
* This file is part of rasdaman community.
*
* Rasdaman community is free software: you can redistribute it and/
↪or modify
* it under the terms of the GNU Lesser General Public License as␣
↪published by
```

(continues on next page)

(continued from previous page)

```
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Rasdaman community is distributed in the hope that it will be␣
↪useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public␣
↪License
* along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
*
* Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
*
* For more information please see <http://www.rasdaman.org>
* or contact Peter Baumann via <baumann@rasdaman.com>.
*/
/*************************************************************
 *
 * PURPOSE:
 *
 * COMMENTS:
 *
 * BUGS:
 *
 ************************************************************/
```

Standard Source Headers

```
/*
* This file is part of rasdaman community.
*
* Rasdaman community is free software: you can redistribute it and/
↪or modify
* it under the terms of the GNU General Public License as published␣
↪by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Rasdaman community is distributed in the hope that it will be␣
↪useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
```

(continues on next page)

(continued from previous page)

```
* You should have received a copy of the GNU General Public License
* along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
*
* Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
*
* For more information please see <http://www.rasdaman.org>
* or contact Peter Baumann via <baumann@rasdaman.com>.
*/
/************************************************************
 *
 * PURPOSE:
 *
 * COMMENTS:
 *
 * BUGS:
 *
 ************************************************************/
```

Standard Source Header (LGPL)

```
/*
* This file is part of rasdaman community.
*
* Rasdaman community is free software: you can redistribute it and/
↪or modify
* it under the terms of the GNU Lesser General Public License as␣
↪published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Rasdaman community is distributed in the hope that it will be␣
↪useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public␣
↪License
* along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
*
* Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
*
* For more information please see <http://www.rasdaman.org>
* or contact Peter Baumann via <baumann@rasdaman.com>.
*/
```

(continues on next page)

(continued from previous page)

```
/*************************************************************
 *
 * PURPOSE:
 *
 * COMMENTS:
 *
 * BUGS:
 *
 ************************************************************/
```

Standard Inline Header

```
/*
* This file is part of rasdaman community.
*
* Rasdaman community is free software: you can redistribute it and/
↪or modify
* it under the terms of the GNU General Public License as published␣
↪by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* Rasdaman community is distributed in the hope that it will be␣
↪useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
*
* Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
*
* For more information please see <http://www.rasdaman.org>
* or contact Peter Baumann via <baumann@rasdaman.com>.
/
/**
 * INLINE SOURCE:
 *
 * MODULE:
 * CLASS:
 *
 * COMMENTS:
 *
*/
```

Standard Script / Make Header

```
#
# MAKEFILE FOR:
#
# This file is part of rasdaman community.
#
# Rasdaman community is free software: you can redistribute it and/
↪or modify
# it under the terms of the GNU General Public License as published␣
↪by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# Rasdaman community is distributed in the hope that it will be␣
↪useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
#
# Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
#
# For more information please see <http://www.rasdaman.org>
# or contact Peter Baumann via <baumann@rasdaman.com>.
# Top Level makefile. This points to the various modules that have␣
↪to be build
# and/or deployed
#
#
# COMMENTS:
#
##################################################################
```

Standard Script / Make Header (LGPL)

```
#
# MAKEFILE FOR:
#
# This file is part of rasdaman community.
#
# Rasdaman community is free software: you can redistribute it and/
↪or modify
# it under the terms of the GNU Lesser General Public License as␣
↪published by
```

(continues on next page)

(continued from previous page)

```
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# Rasdaman community is distributed in the hope that it will be␣
↪useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public␣
↪License
# along with rasdaman community.  If not, see <http://www.gnu.org/
↪licenses/>.
#
# Copyright 2003 - 2018 Peter Baumann / rasdaman GmbH.
#
# For more information please see <http://www.rasdaman.org>
# or contact Peter Baumann via <baumann@rasdaman.com>.
#
#
# COMMENTS:
#
##################################################################
```

Recomendation 12

Correct:

```
inline int
getLenght()
{
    ...
}
```

Wrong:

```
inline int getLenght()
{
    ...
}
```

Macros vs inline functions

Wrong:

```
#define SQUARE(x) ((x)*(x))          // wrong
int a = 2
int b = SQUARE(a++)                  // a == 6
```

Right:

```
inline int
square( int x );                     // right
{
  return (x*x)
}
int c = 2;
int d = square(c++);                 // d == 4
```

Constants vs Standalone Values

Wrong:

```
if (iterations <= 0)
    iterations = 5;
```

Correct:

```
// Default number of iterations in units
const int defaultIterationsNumber = 5;

...

if (iterations <= 0)
    iterations = defaultIterationsNumber;
```

Macros vs const variables

```
#define BUFSIZE 7              // no type checking

const int bufSize = 7          // type checking takes place

enum  size { BufSize = 7 };    // type checking takes place
```

Standard Method Declaration

```
/**
* Description of addNumbers
* @param n1 the first argument.
* @param n2 the second argument.
* @return The return value
*/
template <class P>
int
addNumbers(int n1, int n2)
{
    ...
}
```

Case statement

```
switch(tag)
{
  case A:
    // do something
    // break is missing and foo() is also called in case A    //␣
↪wrong

  case B:
    foo();
    // do something else
    break;

  default:
    // if no match in above cases, this is executed
    break;
}
```

Dynamic array allocation and deallocation

```
int n = 7
T* myT = new T[n];  // T is type with defined constructors and␣
↪destructors

//........

delete myT;         // No! Destructor only called for first object␣
↪in array a.
delete [10] myT ;   // No! Destructor called on memory out of␣
↪bounds in array a.
delete [] myT ;     // OK, and always safe.
```

Standard Class Definition

Example class definitions in accordance with the style rules

```
class String : private Object
{
public:
    String();
    String(const String&);
    unsigned getLenght() const;
    inline Encoding getEncoding() const;
    inline void setEncoding(Encoding newEncoding);

protected:
    int checkIndex( unsigned index ) const;

private:
    unsigned noOfChars;
    Encoding encoding;

};
```

Wrong:

```
class String
{
  public:
    int getLength() const // No !!
    {
      return length;
    };

  private:
    int length;
};
```

Correct:

```
class String
{
  public:
    int getLength() const;

  private:
    int length;
};

inline int
String::getLength() const
{
```

(continues on next page)

(continued from previous page)

```
  return len ;
}
```

Classes with dynamic member data

Declaration examples of the assignment operator:

```
MySpezialClass&
MySpezialClass::operator= (const MySpezialClass msp);      // no

void
MySpezialClass::operator= (const MySpezialClass msp);      // well

const MySpezialClass&
MySpezialClass::operator= (const MySpezialClass msp);      //␣
↪recommanded

Class definition

class DangerousBlob
{
  public:
    const DangerousBlob& operator=(const DangerousBlob& dbr);

  private:
    char* cp;
};
```

Definition of assignment operator:

```
const DangerousBlob&
DangerousBlob::operator=(const Dangerous& dbr)
{
  if ( this != &dbr )           // Guard against assigning to the
↪"this" pointer
  {
    // ...
    delete cp;                  // Disastrous if this == &dbr
    // ...
  }
}
```

Constant references as return types:

```
class Account
{
  public:
    Account ( int myMoney ): moneyAmount(myMoney) { };
```

(continues on next page)

(continued from previous page)

```
    const int& getSafemoney()  const { return moneyAmount;};
    int&       getRiskyMoney() const { return moneyAmount;};   // no

  private:
     int moneyAmount;
};

Account myAcc(10);
myAcc.getSafeMoney()  += 100000;  // compilation error: assignment␣
↪to constant
myAcc.getRiskyMoney() += 1000000; // myAcc::moneyAmount = 1000010 !!
```

Note: Method definition within the class definition is forbidden by rule.

Parameter declaration

```
int setPoint( int, int )      // wrong
int setPoint( int x, int y )

int
setPoint( int x, int y )
{
  //....
}
```

Return type

```
int
calculate ( int j )
{
  return 2*j;
}

void
noReturnType( char* xData, char* yFile)
{
  //....
}
```

Include directive

```
// file is PrintData.cc

#include "PrintData.hh"     // user include file

#include <iostream.h>       // include file of the system library
```

Avoid global data

```
class globale
{
  public:
    //........

  protected:
    const char* functionTitle = "good style";

    int   constGlobal;
    char* varGlobal;
}
```

Formating of functions

```
void foo (); // no
void foo();  // better

// right
int
myComplicateFunction( unsigned unsignedValue,
                      int intValue
                      char* charPointerValue );

// wrong
int myComplicateFunction (unsigned unsignedValue, int intValue␣
↪char* charPointerValue);
```

Formating of pointer and reference types

```
char*
object::asString()
{
  // something
};
```

(continues on next page)

(continued from previous page)

```
char* userName = 0;
int   sfBlock  = 42;
int&  anIntRef = sfBlock;
```

Assignment operator

```
MySpezialClass&
MySpezialClass::operator=( const MySpezialClass& msp ); // no

const MySpezialClass&
MySpezialClass::operator=( const MySpezialClass& msp ); //␣
↪recommended
```

Reference vs pointer

```
// Unnecessarily complicated use of pointers
void addOneComplicated ( int* integerPointer )
{
  *integerPointer += 1:
}
addOneComplicated (&j)


// Write this way instead
void addOneEasy ( int& integerReference )
{
  integerReference +=1:
}
addOneEasy(i);
```

Call-by-value vs call-by-constant-reference

```
// this may lead to very inefficient code.
void foo( string s );
string a;
foo(a)                // call-by-value

// the actual argument is used by the function
// but it connot be modified by the function.
void foo( const string& s );
string c;
foo(c);               // call-by-constant-reference
```

Avoid continue

```
while ( /* something */ )
{
  if (/* something */)
  {
    // do something
    continue;                    // Wrong!
  }
  // do something
}

// By using an extern 'else' clause, continue is avoided and the␣
↪code
// is easier to understand

while ( /* something */ )
{
  if (/* something */)
  {
    // do something
  }
  else
  {
    // do something
  }
}
```

Parentheses

```
// Interpreted as (a<b)<c, not (a<b) && (b<c)
if (a<b<c)
{
  //...
}

// Interpreted as a & (b<8), (a&b) <8
if (a & b<8)
{
  //..
}

// when parentheses are recommended
int i = a>=b && c < d && e+f <= g+h;          // no
int j = (a>=b)&&(c<d) && (( e+f) <= (g+h)); // better
```

Include Files

Include file for the class `PackableString`:

```
#ifndef PACKABLESTRING_HH
#define PACKABLESTRING_HH

#include "string.hh".
#include "packable.hh".

/**
 * A test class with elaborate description.
/*

class Buffer:public String:public Packable
{
  public:
    class PackableString (const String& s);
    class Buffer* put (class Buffer* outbuffer);
    //.......
};

#endif
```

Implementation file for the class `PackableString`:

```
// PackableString.cc
// not recommanded <../include/iostream.h> Wrong

#include <iostream.h> // Right
#include "PackableString.hh"
// to be able to use Buffer instances, buffer.hh must be included.
#include "buffer.hh"

Buffer*
PackableString::put(Buffer* outbuffer)
{
    //......
}
```

3.7 Geo services

3.7.1 Petascope Developer's Documentation

Introduction

This page serves as an introduction to the petascope component from a developer's perspective (see also *Geo Services Guide*).

Petascope is built on the **Spring Boot Framework** with **Hibernate** as object relational mapping data model for backend-communication with petascopedb; Implements support for the Coverage Schema Implementation (CIS version 1.0: *GridCoverage*, *RectifiedGridCoverage* and *ReferenceableGridCoverage* and CIS version 1.1: *GeneralGridCoverage* which is the unified class for coverage types in CIS 1.0).

Petascope can be deployed on more backend DBMS beside PostgreSQL like HSQLDB, H2, etc. Postgresql is still the most stable database for deploying petascope, but the user can switch to other databases by changing the configuration in petascope.properties.

The Spring Boot Framework provides many utilities that aid in quicker development of petascope. Petascope can now start as an embedded web application with an internal embedded Tomcat (i.e: no need to deploy to external Tomcat).

Code organization

Petascope is divided in 3 applications:

- core contains the classes to generate petascopedb's tables by **Hibernate** with **Liquibase** and other utilities classes. This is the core library used by other petascope's applications.
- main contains the classes to handle WCS, WCPS, WMS, WCST-T requests and generates rasql queries for rasdaman. This is the **rasdaman.war** application to be deployed to external **Tomcat** or started in embedded mode with `java -jar rasdaman.war`.
- migration handles petascopedb migration (**must need when updating from v9.4 to v9.5+**) using Liquibase; it can also migrates petascopedb from Postgresql to another DBMS like H2 or HSQLDB.

Database migration

Schema migration

To support different kinds of databases, we use **Liquibase**, which creates the changes for each update in XML and uses that to generate the SQL statements for the target database (e.g: Postgresql, HSQLDB, H2, etc). To further understand how **Liquibase** works to populate database tables, see comments in the liquibase.properties config file. and list of existing schema versions files.

Data migration

This feature is invoked in some cases, for example: existing data need to be corrected or newly data need to be populated in newly created tables by **Liquibase**.

- For each data migration version, there must have one java class to handle, these classes are added in `org.rasdaman.datamigration` package and extends `AbstractDataMigrationHandler` abstract class. Each handler class should have an incremental `migrationVersion` property (1-based) and an unique `handlerId` (generated by `uuid` command tool).

- When petascope starts, from the list of all handlers, for each handler, it checks if the `handlerId` exists in column `applied_migration`. If it does not exists, this handler will run the migration and insert its `handlerId` as a new row in table `applied_migration`.

CRS management

Petascope relies on a **SECORE Coordinate Reference System (CRS)** resolver that can provide proper metadata on a coverage's native CRS. One can either deploy a local SECORE instance, or use the official OGC SECORE resolver at http://www.opengis.net/def/.

It currently keeps a few internal caches, especially for SECORE CRS resources and responses: the gain is both on performance and on robustness against network latencies. Caching information about CRSs is safe as CRSs can be considered static resources - normally they do not change (and with the CRS versioning recently introduced by OGC a particular CRS version never will change indeed).

It is suggested to run a *WCS GetCapabilities* after a fresh new deployment, so that the CRS definitions of all the offered coverages are cached: after that single request, mainly almost all the CRS-related information has already been cached.

The **CrsUtil** class serves several purposes:

- CRS definitions: the relevant information parsed from a GML CRS definition is stored as a CrsDefinition object. This includes both spatial and temporal reference systems;
- CRS equivalence tests: thanks to the `/equal` endpoint of SECORE, effective equivalence (no simple string comparison) between two reference systems can be verified. This operation is required when checking if a CRS has been cached or not: as an example, KVP notation of a CRS URI is independent of the order of key/value pairs, so that http://www.opengis.net/def/crs?authority=EPSG&version=0&code=32633 and http://www.opengis.net/def/crs?version=0&authority=EPSG&code=32633 are equivalent despite their different URI identifier.

Testing

The systemtest/testcase_services covers all the possible cases for WCS, WCPS, WMS and WCS-T. The easiest way to understand how Petascope works is by running some tests and debug it with your IDE (e.g: NetBeans, IntelliJ IDEA,. . .).

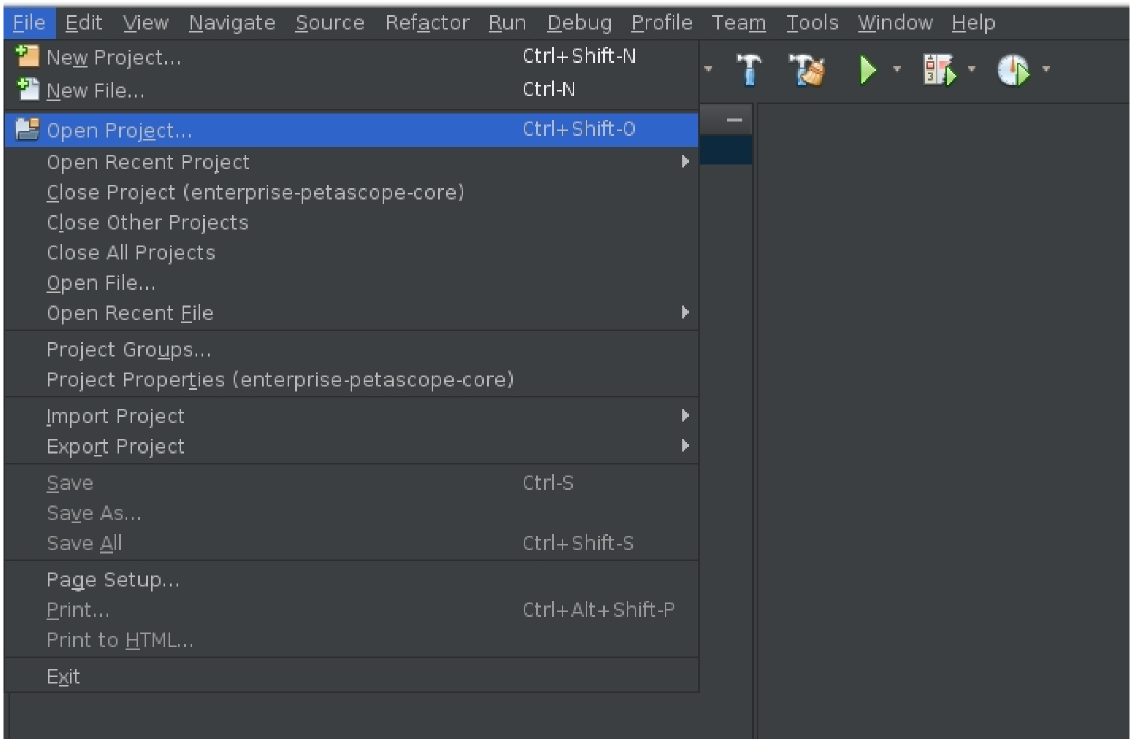
For instance: send this request in Web Browser with deployed petascope in Tomcat: http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1&request=GetCapabilities. Then you can set a debug in class `petascope.controller.PetascopeController` of **petascope-main** application, then follow all following classes when debugging to understand how the request is handled inside petascope.

Development & Debugging

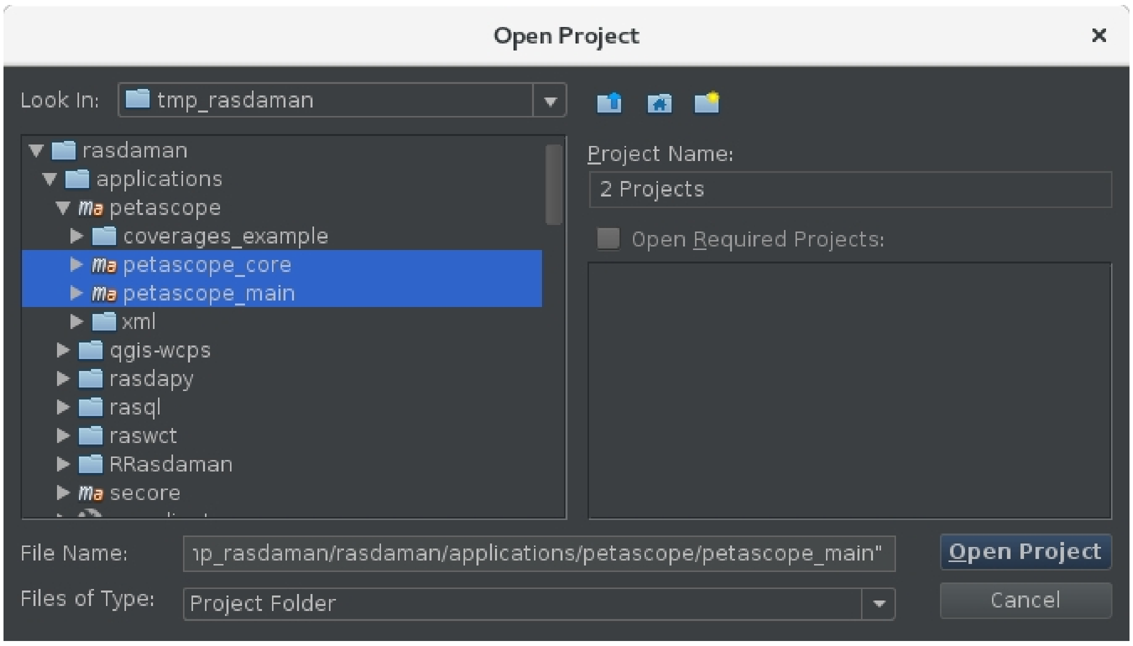
After one has downloaded rasdaman source code and compiled it successfully (see *detail*), you can load the petascope code in an IDE like NetBeans and run/debug it during development.

In the below examples, NetBeans IDE version 8.2 will be used for demonstration.

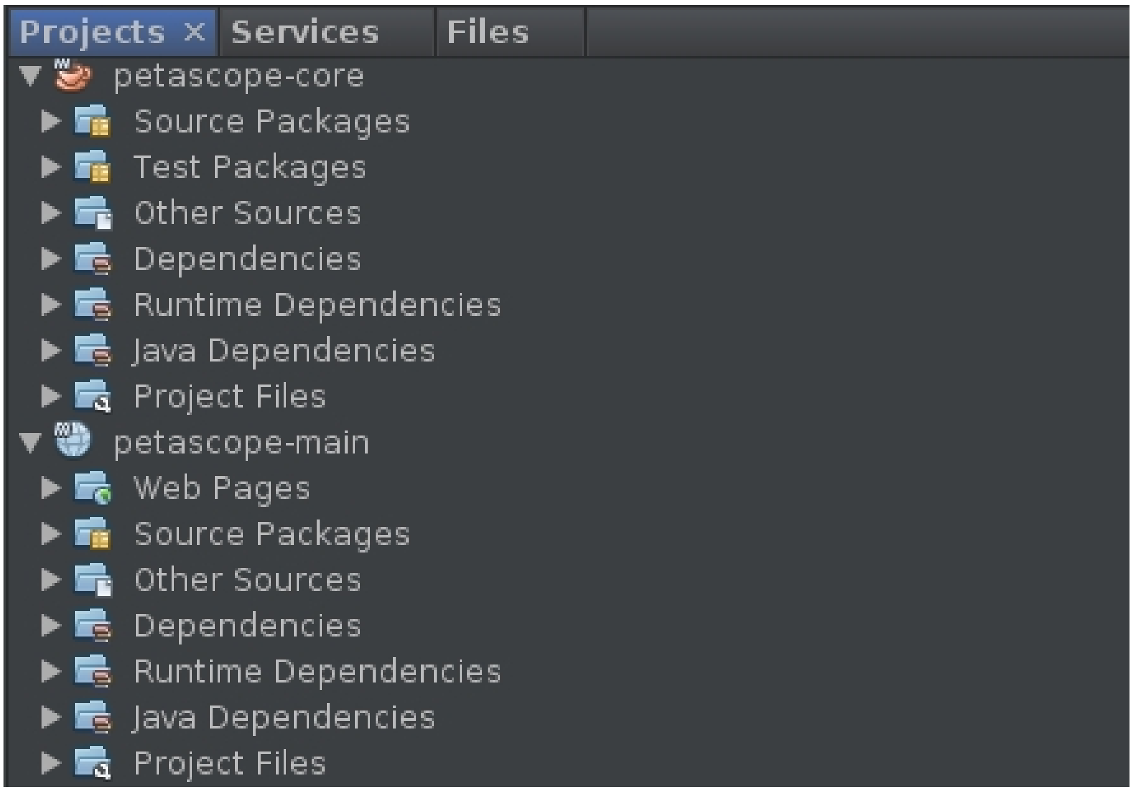
- Open the NetBeans IDE and locate the source code of petascope from the rasdaman source tree in `applications/petascope`:



- Select the two petascope sub-projects: `petascope_core` and `petascope_main`:



Now the projects should display on the left panel:



```
▼ petascope-core
  ▶ Source Packages
  ▶ Test Packages
  ▶ Other Sources
  ▶ Dependencies
  ▶ Runtime Dependencies
  ▶ Java Dependencies
  ▶ Project Files
▼ petascope-main
  ▶ Web Pages
  ▶ Source Packages
  ▶ Other Sources
  ▶ Dependencies
  ▶ Runtime Dependencies
  ▶ Java Dependencies
  ▶ Project Files
```

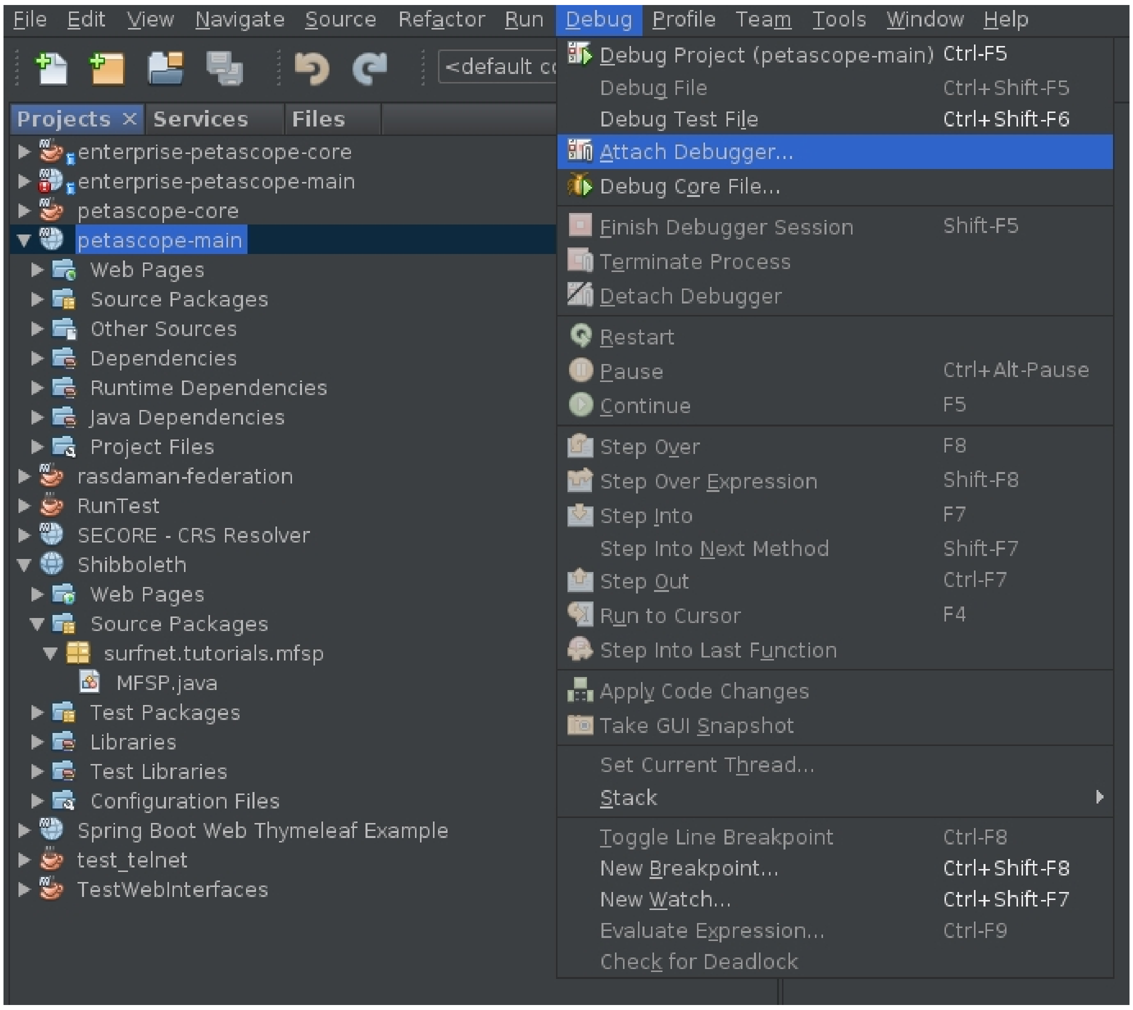
- Next, change directory on terminal to `petascope_main` source code location:

```
cd applications/petascope/petascope_main
```

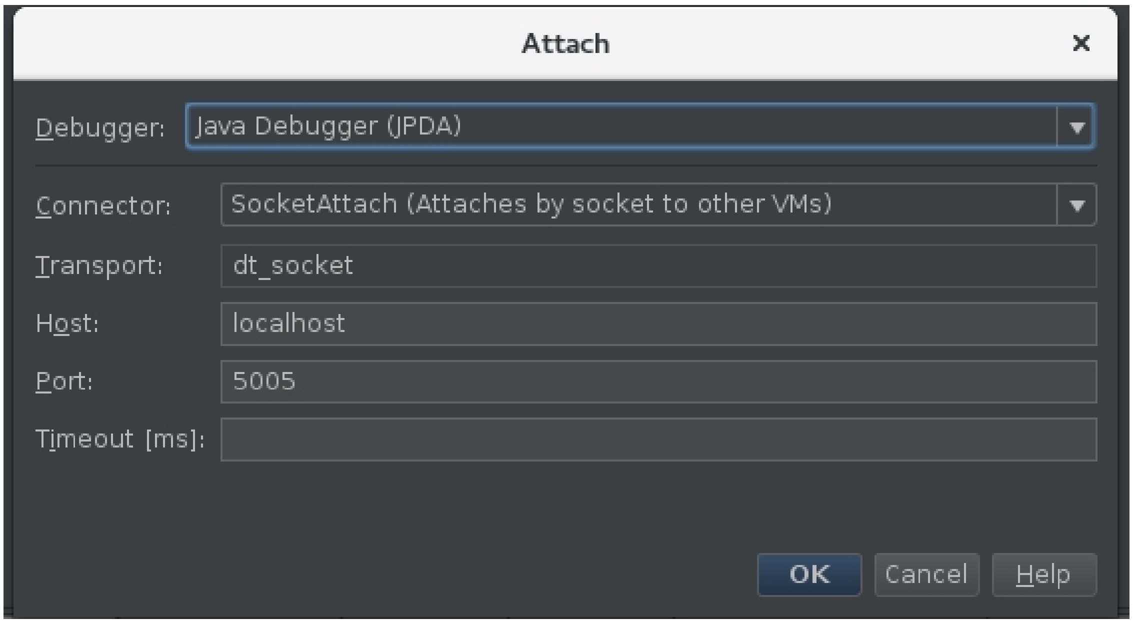
- Run petascope application as a **standalone web application** with **embedded Tomcat** and Tomcat will listen on port `5005` for *debugging*. Petascope will start at the port configured in `/opt/rasdaman/etc/petascope.properties`, setting `server.port` (e.g `8090`).

```
mvn spring-boot:run -Drun.jvmArguments="-Xdebug -
↪Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005"
```

- Next, attach the debugger from NetBeans to the embedded Tomcat:



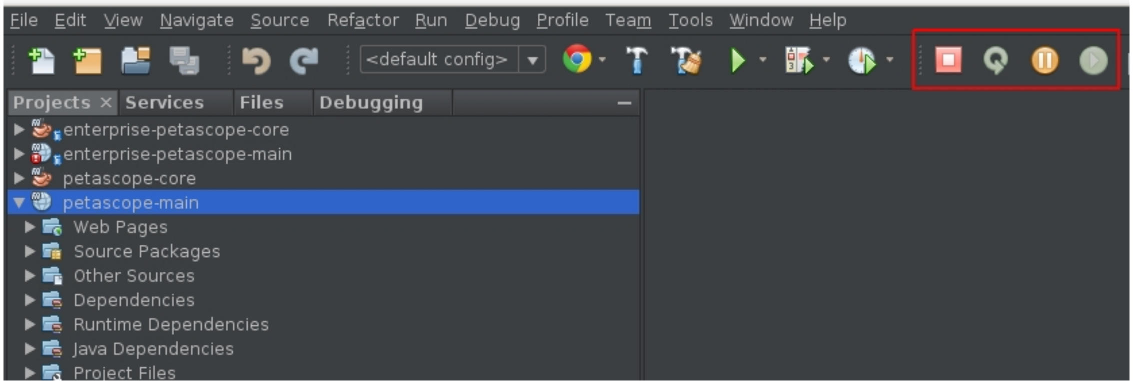
- Input the debugging port `5005` which was used to start embedded Tomcat above:



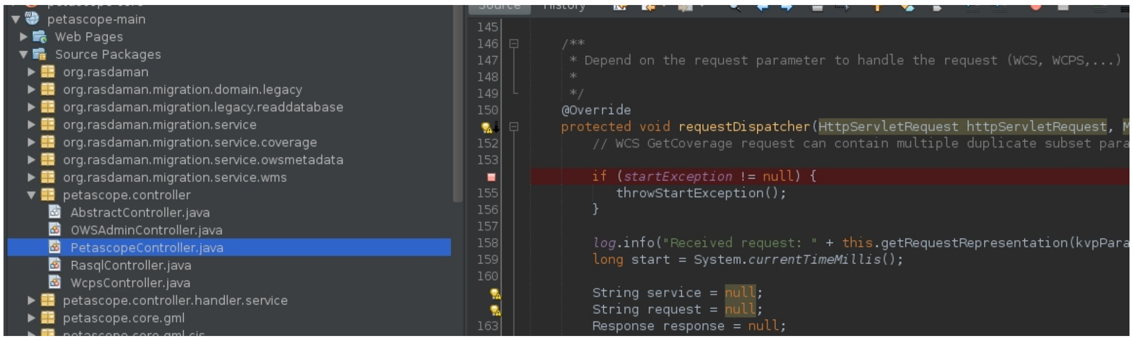
Attach

Debugger:	Java Debugger (JPDA)
Connector:	SocketAttach (Attaches by socket to other VMs)
Transport:	dt_socket
Host:	localhost
Port:	5005
Timeout [ms]:	

- After that, NetBeans will show the panel with control buttons for debugging:

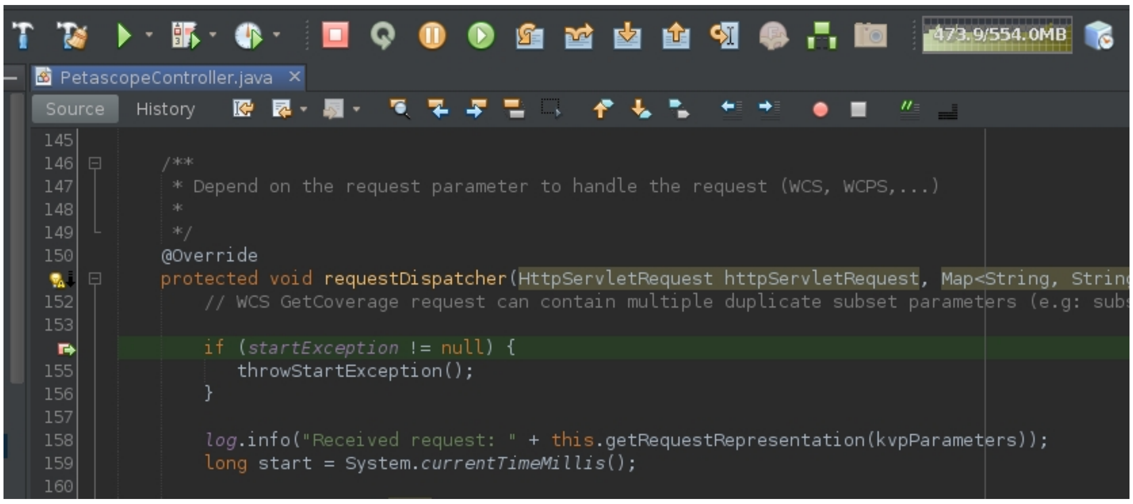


- Add a test break point by clicking on the number in the middle panel, for example: in `petascope-main` project, class `PetascopeController`, line `154`:


petascope-main
Web Pages
Source Packages
org.rasdaman
org.rasdaman.migration.domain.legacy
org.rasdaman.migration.legacy.readdatabase
org.rasdaman.migration.service
org.rasdaman.migration.service.coverage
org.rasdaman.migration.service.owsmetadata
org.rasdaman.migration.service.wms
petascope.controller
AbstractController.java
OWSAdminController.java
PetascopeController.java
RasqlController.java
WcpsController.java
petascope.controller.handler.service
petascope.core.gml

```
145
146     /**
147      * Depend on the request parameter to handle the request (WCS, WCPS,...)
148      *
149      */
150     @Override
        protected void requestDispatcher(HttpServletRequest httpServletRequest, M
152         // WCS GetCoverage request can contain multiple duplicate subset para
153
            if (startException != null) {
155             throwStartException();
156         }
157
158         log.info("Received request: " + this.getRequestRepresentation(kvpPara
159         long start = System.currentTimeMillis();
160
            String service = null;
            String request = null;
163         Response response = null;
```

- Then, open this embedded petascope endpoint `http://localhost:8090/rasdaman/ows` on web browser and see NetBeans stops petascope application at the selected line above for inspecting and debugging:



```
145
146     /**
147      * Depend on the request parameter to handle the request (WCS, WCPS,...)
148      *
149      */
150     @Override
        protected void requestDispatcher(HttpServletRequest httpServletRequest, Map<String, Strin
152         // WCS GetCoverage request can contain multiple duplicate subset parameters (e.g: subs
153
            if (startException != null) {
155             throwStartException();
156         }
157
158         log.info("Received request: " + this.getRequestRepresentation(kvpParameters));
159         long start = System.currentTimeMillis();
160
```

- Furthermore, you can step with the debugger, see variable values, continue execution, etc.

Warnings

Don't create `BigDecimal` directly from a `double` variable, rather from `double.toString()`. E.g. `BigDecimal a = new BigDecimal(0.2356d)` will result with random fraction numbers after the real value of double (*0.2356534858345843959290904239049023490239042903490239 04*); subsequently this would lead to wrong coefficient calculation in petascope.

3.7.2 WSClient Developer's Documentation

Introduction

WSClient is a frontend Web application which facilitates interactions from users to petascope (OGC WCS/WCPS/WMS standards implementation). It it based on the AngularJS framework version 1.4 with other libraries like CSS Bootstrap and WebWorldWind to make a single page application.

When building petascope, WSClient is added to *rasdaman.war*, which is then deployed to Tomcat. For example, in Tomcat 9 / Ubuntu 18.04 the WSClient can be found in this directory:

```
/var/lib/tomcat9/webapps/rasdaman/WEB-INF/classes/public/ows/
```

Code

WSClient uses TypeScript rather JavaScript directly. To compile WSClient, the following dependencies are necessary:

- *npm* - Node package manger:

```
# CentOS
$ sudo yum install npm
# Debian / Ubuntu
$ sudo apt-get install npm
```

- *tsc* - Used for compiling TypeScript .ts files to JavaScript .js:

```
$ sudo npm install -g tsc
```

Everytime a new feature/fix is added, one needs to compile from TypeScript to JavaScript to work in Web Browsers with the following command in WSClient source folder (`application/wcs-client/app`):

```
$ tsc
```

This will generate two important files in `application/wcs-client/app/ows`: `main.js` and `main.js.map`. They need to be included in the patch besides other added/updated files.

3.7.3 SECORE Developer's Documentation

Introduction

SECORE (Semantic Cordinate Reference System Resolver) is a server which resolves CRS URLs into full CRS definitions represented in GML 3.2.1. Offical SECORE of rasdaman is hosted at: http://www.opengis.net/def.

Same as Petascope, SECORE builds on Spring framework. However, as it is an XML database resolver (mainly all CRSs are occupied from EPSG releases), hence it does not rely on any relational database as petascopedb.

Code

SECORE stores and queries XML data in a BaseX XML database. On the disk this database is stored in `$CATALINA_HOME/webapps/secoredb` (e.g: `/var/lib/tomcat/webapps`), this is the directory where external Tomcat process will typically have write access. The database is created and maintained automatically, so no action by the user is required regarding this.

In SECORE, there are 2 types of GML Database (*UserDictionary.xml* and *GmlDictionary.xml*). User will *add/update/delete* CRSs **only** in *UserDictionary.xml* when *GmlDictionary.xml* comming from EPSG releases are intact.

SECORE database tree can be viewed and (upon login) modified via graphical web interface at "http://your.server/def/index.jsp".

More generally, any folder and definition can turn to EDIT mode by appending a **/browse.jsp** to its URI: e.g. "http://your.server/def/uom/EPSG/0/9001/browse.jsp" will let you *view/edit* EPSG:9001 unit of measure, whereas "http://your.server/def/uom/EPSG/0/browse.jsp" will let you either *remove* EPSG UoM definitions or *add a new one*, not necessarily under the EPSG branch: the **gml:identifier** of the new definition will determine its position in the tree.

As explained in the `related publication <http://link.springer.com/chapter/10.1007%2F978-3-642-29247-7_5>`_, SECORE supports *parametrization of CRSs* as well: with this regard, you should mind that relative XPaths are not allowed (either start with */* or *//* when selecting nodes); non-numeric parameters must be embraced by single or double quotes both when setting optional default values in the definition or when setting custom values in the URI.

Update new EPSG version

When EPSG announces a new release, one can download the new GML dictionary file from this link: http://www.epsg-registry.org.

From the downloaded .zip file, extract *GmlDictionary.xml* file inside and add it to SECORE secore database under a folder with version name (e.g: `9.4.2/GmlDictionary.xml`).

After that, build SECORE normally to have a new web application *def.war* and redeploy it to Tomcat server. Finally, check if a new EPSG version is added from http://your.server/def/EPSG/. Example:

```
<identifiers xmlns="http://www.opengis.net/crs-nts/1.0"
   xmlns:gco="http://www.isotc211.org/2005/gco"
   xmlns:gmd="http://www.isotc211.org/2005/gmd"
   at="http://localhost:8080/def/crs/EPSG/">
   <identifier>http://localhost:8080/def/crs/EPSG/0</identifier>
   <identifier>http://localhost:8080/def/crs/EPSG/8.5</identifier>
   <identifier>http://localhost:8080/def/crs/EPSG/8.9.2</identifier>
   <identifier>http://localhost:8080/def/crs/EPSG/9.4.2</identifier>
</identifiers>
```

3.7.4 wcst_import Developer's Documentation

Development is best done by opening `applications/wcst_import/` in the PyCharm IDE. It is recommended to install pylint as follows:

```
pip3 install pylint=2.13.4
```

Then it will be automatically executed when you do `make` to build rasdaman, or manually do `make pylint.check` in `applications/wcst_import/`. It uses the configuration file `applications/wcst_import/pylint.cfg` to customize the execution.

In pycharm one can install the pylint plugin, and set the `applications/wcst_import/pylint.cfg` file as configuration file. If the plugin does not work as expected, pylint can be setup as an External Tool and manually invoked as needed.

CHAPTER

FOUR

QUERY LANGUAGE GUIDE

4.1 Preface

4.1.1 Overview

This guide provides information about how to use the rasdaman database management system (in short: rasdaman). The document explains usage of the rasdaman Query Language.

Follow the instructions in this guide as you develop your application which makes use of rasdaman services. Explanations detail how to create type definitions and instances; how to retrieve information from databases; how to insert, manipulate, and delete data within databases.

4.1.2 Audience

The information in this manual is intended primarily for application developers; additionally, it can be useful for advanced users of rasdaman applications and for database administrators.

4.1.3 Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as raswct, release notes, and additional information on the rasdaman wiki.

4.2 Introduction

4.2.1 Multidimensional Data

In principle, any natural phenomenon becomes spatio-temporal array data of some specific dimensionality once it is sampled and quantised for storage and manipulation in a computer system; additionally, a variety of artificial sources such as simulators, image renderers, and data warehouse population tools generate array data. The common characteristic they all share is that a large set of large multidimensional arrays has to be maintained. We call such arrays multidimensional discrete data (or short: MDD) expressing the variety of dimensions and separating them from the conceptually different multidimensional vectorial data appearing in geo databases.

rasdaman is a domain-independent database management system (DBMS) which supports multidimensional arrays of any size and dimension and over freely definable cell types. Versatile interfaces allow rapid application deployment while a set of cutting-edge intelligent optimization techniques in the rasdaman server ensures fast, efficient access to large data sets, particularly in networked environments.

4.2.2 Rasdaman Overall Architecture

The rasdaman client/server DBMS has been designed using internationally approved standards wherever possible. The system follows a two-tier client/server architecture with query processing completely done in the server. Internally and invisible to the application, arrays are decomposed into smaller units which are maintained in a conventional DBMS, for our purposes called the *base DBMS*.

On the other hand, the base DBMS usually will hold alphanumeric data (such as metadata) besides the array data. Rasdaman offers means to establish references between arrays and alphanumeric data in both directions.

Hence, all multidimensional data go into the same physical database as the alphanumeric data, thereby considerably easing database maintenance (consistency, backup, etc.).

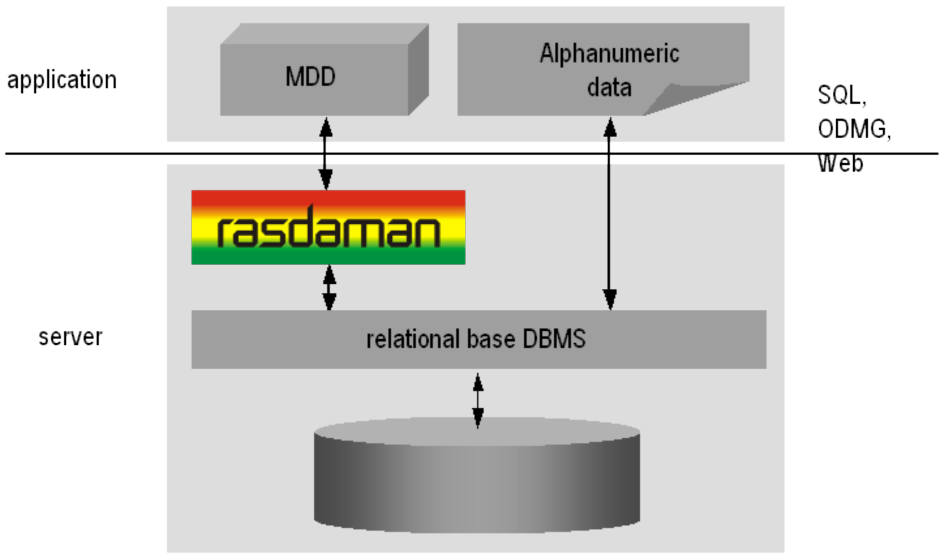


Figure 4.1: Embedding of rasdaman in IT infrastructure

Further information on application program interfacing, administration, and related topics is available in the other components of the rasdaman documentation set.

4.2.3 Interfaces

The syntactical elements explained in this document comprise the rasql language interface to rasdaman. There are several ways to actually enter such statements into the rasdaman system:

- By using the rasql command-line tool to send queries to rasdaman and get back the results.
- By developing an application program which uses the raslib/rasj function `oql_execute()` to forward query strings to the rasdaman server and get back the results.

Developing applications using the client API is the subject of this document. Please refer to the *C++ Developers Guide* or *Java Developers Guide* of the rasdaman documentation set for further information.

4.2.4 rasql and Standard SQL

The declarative interface to the rasdaman system consists of the *rasdaman Query Language,* rasql, which supports retrieval, manipulation, and data definition.

Moreover, the rasdaman query language, rasql, is very similar - and in fact embeds into - standard SQL. With only slight adaptations, rasql has been standardized by ISO as 9075 SQL Part 15: MDA (Multi-Dimensional Arrays). Hence, if you are familiar with SQL, you will quickly be able to use rasql. Otherwise you may want to consult the introductory literature referenced at the end of this chapter.

4.2.5 Notational Conventions

The following notational conventions are used in this manual:

Program text (under this we also subsume queries in the document on hand) is printed in a `monotype font`. Such text is further differentiated into keywords and syntactic variables. Keywords like struct are printed in boldface; they have to be typed in as is.

An optional clause is enclosed in brackets; an arbitrary repetition is indicated through brackets and an ellipsis. Grammar alternatives can be grouped in parentheses separated by a `|` symbol.

Example

```
select resultList
from namedCollection [ [ as ] collIterator ]
     [ , namedCollection [ [ as ] collIterator ] ]...
[ where booleanExp ]
```

It is important not to mix the regular brackets `[` and `]` denoting array access, trimming, etc., with the grammar brackets `[` and `]` denoting optional clauses and repetition; in grammar excerpts the first case is in double quotes. The same applies to parentheses.

Italics are used in the text to draw attention to the first instance of a defined term in the text. In this case, the font is the same as in the running text, not Courier as in code pieces.

4.3 Terminology

4.3.1 An Intuitive Definition

An array is a set of elements which are ordered in space. The space considered here is discretized, i.e., only integer coordinates are admitted. The number of integers needed to identify a particular position in this space is called the *dimension* (sometimes also referred to as *dimensionality*). Each array element, which is referred to as *cell*, is positioned in space through its *coordinates*.

A cell can contain a single value (such as an intensity value in case of grayscale images) or a composite value (such as integer triples for the red, green, and blue component of a color image). All cells share the same structure which is referred to as the *array cell type* or *array base type*.

Implicitly a neighborhood is defined among cells through their coordinates: incrementing or decrementing any component of a coordinate will lead to another point in space. However, not all points of this (infinite) space will actually house a cell. For each dimension, there is a *lower* and *upper bound*, and only within these limits array cells are allowed; we call this area the *spatial domain* of an array. In the end, arrays look like multidimensional rectangles with limits parallel to the coordinate axes. The database developer defines both spatial domain and cell type in the *array type definition*. Not all bounds have to be fixed during type definition time, though: It is possible to leave bounds open so that the array can dynamically grow and shrink over its lifetime.

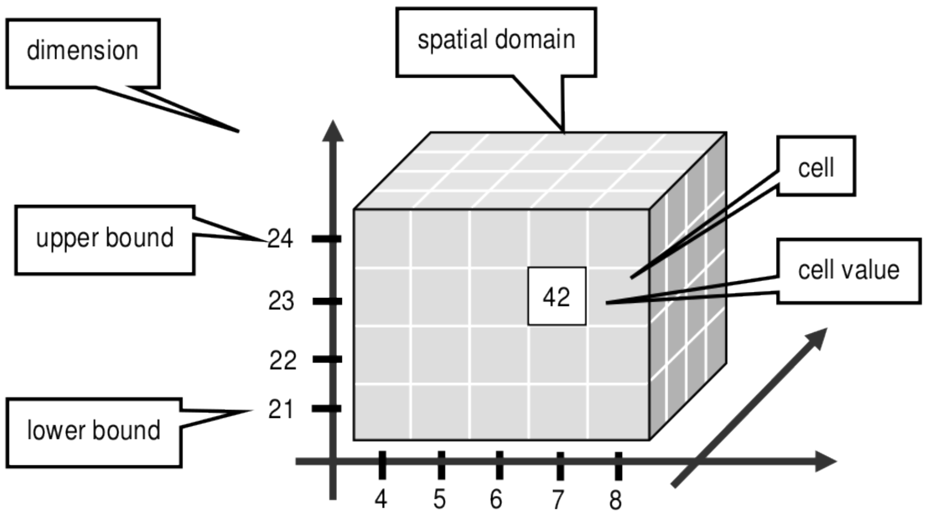


Figure 4.2: Constituents of an array

Synonyms for the term array are *multidimensional array* / *MDA*, *multidimensional data* / *MDD*, *raster data*, *gridded data*. They are used interchangeably in the rasdaman documentation.

In rasdaman databases, arrays are grouped into collections. All elements of a collection share the same array type definition (for the remaining degrees of freedom see *Array types*). Collections form the basis for array handling, just as tables do in relational database technology.

4.3.2 A Technical Definition

Programmers who are familiar with the concept of arrays in programming languages maybe prefer this more technical definition:

An array is a mapping from integer coordinates, the spatial domain, to some data type, the cell type. An array's spatial domain, which is always finite, is described by a pair of lower bounds and upper bounds for each dimension, resp. Arrays, therefore, always cover a finite, axis-parallel subset of Euclidean space.

Cell types can be any of the base types and composite types defined in the ODMG standard and known, for example from C/C++. In fact, most admissible C/C++ types are admissible in the rasdaman type system, too.

In rasdaman, arrays are strictly typed wrt. spatial domain and cell type. Type checking is done at query evaluation time. Type checking can be disabled selectively for an arbitrary number of lower and upper bounds of an array, thereby allowing for arrays whose spatial domains vary over the array lifetime.

4.4 Sample Database

4.4.1 Collection mr

This section introduces sample collections used later in this manual. The sample database which is shipped together with the system contains the schema and the instances outlined in the sequel.

Collection `mr` consists of three images (see Figure 4.3) taken from the same patient using magnetic resonance tomography. Images are 8 bit grayscale with pixel values between 0 and 255 and a size of 256x211.

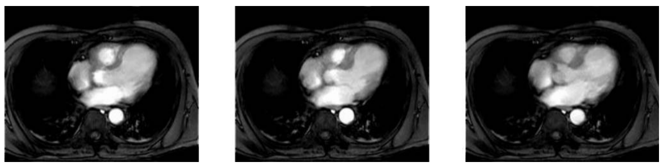


Figure 4.3: Sample collection `mr`

4.4.2 Collection mr2

Collection `mr2` consists of only one image, namely the first image of collection `mr` (Figure 4.4). Hence, it is also 8 bit grayscale with size 256x211.

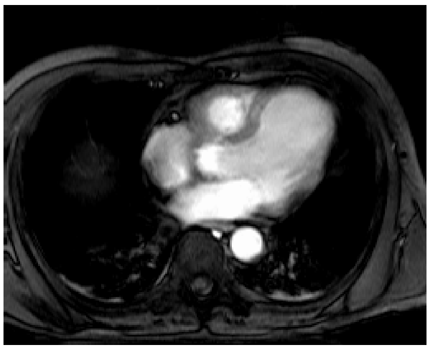


Figure 4.4: Sample collection `mr2`

4.4.3 Collection rgb

The last example collection, `rgb`, contains one item, a picture of the anthur flower (Figure 4.5). It is an RGB image of size 400x344 where each pixel is composed of three 8 bit integer components for the red, green, and blue component, resp.

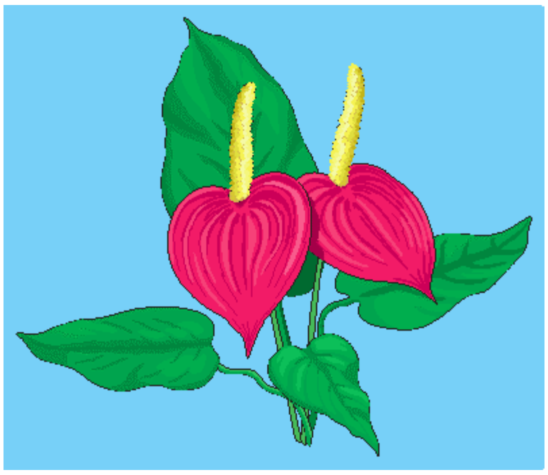


Figure 4.5: The collection `rgb`

4.5 Type Definition Using rasql

4.5.1 Overview

Every instance within a database is described by its *data type* (i.e., there is exactly one data type to which an instance belongs; conversely, one data type can serve to describe an arbitrary number of instances). Each database contains a self-contained set of such type definitions; no other type information, external to a database, is needed for database access.

Types in rasdaman establish a 3-level hierarchy:

- *Cell types* can be atomic base types (such as char or float) or composite ("struct") types such as red / green / blue color pixels.

- *Array types* define arrays over some atomic or struct cell type and a spatial domain.
- *Set types* describe sets of arrays of some particular array type.

Types are identified by their name which must be unique within a database and not exceed length of 200 characters. Like any other identifier in rasql queries, type names are case-sensitive, consist of only letters, digits, or underscore, and must start with a letter.

4.5.2 Cell types

Atomic types

The set of standard atomic types, which is generated during creation of a database, materializes the base types defined in the ODMG standard (cf. Table 4.1).

Table 4.1: rasdaman atomic cell types

type name	size	description
`bool`	1 bit[2]	true (nonzero value), false (zero value)
`octet`	8 bit	signed integer
`char`	8 bit	unsigned integer
`short`	16 bit	signed integer
`unsigned short` / `ushort`	16 bit	unsigned integer
`long`	32 bit	signed integer
`unsigned long` / `ulong`	32 bit	unsigned integer
`float`	32 bit	single precision floating point
`double`	64 bit	double precision floating point
`CInt16`	32 bit	complex of 16 bit signed integers
`CInt32`	64 bit	complex of 32 bit signed integers
`CFloat32`	64 bit	single precision floating point complex
`CFloat64`	128 bit	double precision floating point complex

Composite types

More complex, composite cell types can be defined arbitrarily, based on the system-defined atomic types. The syntax is as follows:

```
create type typeName
as (
  attrName_1 atomicType_1,
  ...
  attrName_n atomicType_n
)
```

Attribute names must be unique within a composite type, otherwise an exception is thrown. No other type with the name `typeName` may pre-exist already.

[2] memory usage is one byte per pixel

Example

An RGB pixel type can be defined as

```
create type RGBPixel
as (
  red char,
  green char,
  blue char
)
```

4.5.3 Array types

An **marray** ("multidimensional array") type defines an array type through its cell type (see *Cell types*) and a spatial domain.

Syntax

The syntax for creating an marray type is as below. There are two variants, corresponding to the dimensionality specification alternatives described above:

```
create type typeName
as baseTypeName mdarray domainSpec
```

where `baseTypeName` is the name of a defined cell type (atomic or composite) and `domainSpec` is a multidimensional interval specification as described in the following section.

Alternatively, a composite cell type can be indicated in-place:

```
create type typeName
as (
  attrName_1 atomicType_1,
  ...
  attrName_n atomicType_n
) mdarray domainSpec
```

No type (of any kind) with name `typeName` may pre-exist already, otherwise an exception is thrown.

Attribute names must be unique within a composite type, otherwise an exception is thrown.

Spatial domain

Dimensions and their extents are specified by providing an axis name for each dimension and, optionally, a lower and upper bound:

```
[ a_1 ( lo_1 : hi_1 ), ... , a_d ( lo_d : hi_d ) ]

[ a_1 , ... , a_d ]
```

where `d` is a positive integer number, `a_i` are identifiers, and `lo_1` and `hi_1` are integers such that `lo_1` $\leq$ `hi_1`. Both `lo_1` and `hi_1` can be an asterisk ($\star$) instead of a number, in which case no limit in the particular direction of the axis will be enforced. If the bounds `lo_1` and `hi_1` on a particular axis are not specified, they are assumed to be equivalent to $\star$.

Axis names must be unique within a domain specification, otherwise an exception is thrown.

Currently axis names are ignored and cannot be used in queries yet.

Examples

The following statement defines a 2-D RGB image, based on the definition of `RGBPixel` as shown above:

```
create type RGBImage
as RGBPixel mdarray [ x ( 0:1023 ), y ( 0:767 ) ]
```

An 2-D image without any extent limitation can be defined through:

```
create type UnboundedImage
as RGBPixel mdarray [ x, y ]
```

which is equivalent to

```
create type UnboundedImage
as RGBPixel mdarray [ x ( *:* ), y ( *:* ) ]
```

Selectively we can also limit only the bounds on the x axis for example:

```
create type PartiallyBoundedImage
as RGBPixel mdarray [ x ( 0 : 1023 ), y ]
```

4.5.4 Set types

A **set** type defines a collection of arrays sharing the same marray type. Additionally, a collection can also have null values which are used in order to characterise sparse arrays. A sparse array is an array where some of the elements have a null value.

Syntax

```
create type typeName
as set ( marrayTypeName [ nullValues ] )
```

where `marrayTypeName` is the name of a defined marray type and `nullValues` is an optional specification of a set of values to be treated as nulls; for semantics in operations refer to *Null Values*.

No type with the name *typeName* may pre-exist already.

Null Values

The optional `nullValues` clause in a set type definition is a set of null value intervals:

```
null values [ nullInterval, ... ]
```

Each `nullInterval` can be a pair of lower and upper limits (1, 2, 3), or a single (double) value (1):

```
lo : hi      (1)
 * : hi      (2)
lo : *       (3)

nullValue    (4)
```

In case of an interval, the three variants are interpreted as follows:

1. Both `lo` and `hi` are double values such that `lo` $\leq$ `hi`;
2. `lo` is $*$ and `hi` is a double value, indicating that all values lower than `hi` are null values;
3. `lo` is a double value and `hi` is $*$, indicating that all values greater than `lo` are null values.

For floating-point data it is recommended to always specify small intervals instead of single numbers with variant (4).

Limitation

Currently, only atomic null values can be indicated. They apply to all components of a composite cell simultaneously. In future it may become possible to indicate null values individually per struct component.

Examples

The following statement defines a set type of 2-D RGB images, based on the definition of RGBImage:

```
create type RGBSet
as set ( RGBImage )
```

If values 0, 253, 254, and 255 are to be considered null values, this can be specified as follows:

```
create type RGBSet
as set ( RGBImage null values [ 0, 253 : 255 ] )
```

Note that these null values will apply equally to every band. It is not possible to separate null values per band.

As the cell type in this case is char (possible values between 0 and 255), the type can be equivalently specified like this:

```
create type RGBSet
as set ( RGBImage null values [ 0, 253 : * ] )
```

With the set type below, values which are nan are null values (nanf is the float constant, while nan is the double constant):

```
create type FloatSetNanNullValue
as set ( FloatImage null values [nanf] )

create type DoubleSetNanNullValue
as set ( DoubleImage null values [nan] )
```

4.5.5 Drop type

A type definition can be dropped (i.e., deleted from the database) if it is not in use. This is the case if both of the following conditions hold:

- The type is not used in any other type definition.
- There are no array instances existing which are based, directly or indirectly, on the type on hand.

Further, atomic base types (such as char) cannot be deleted.

Drop type syntax

```
drop type typeName
```

4.5.6 List available types

A list of all types defined in the database can be obtained in textual form, adhering to the rasql type definition syntax. This is done by querying virtual collections (similar to the virtual collection `RAS_COLLECTIONNAMES`).

Technically, the output of such a query is a list of 1-D `char` arrays, each one containing one type definition.

Syntax

```
select typeColl from typeColl
```

where *typeColl* is one of

- `RAS_STRUCT_TYPES` for struct types
- `RAS_MARRAY_TYPES` for array types
- `RAS_SET_TYPES` for set types
- `RAS_TYPES` for union of all types

Note: Collection aliases can be used, such as:

```
select t from RAS_STRUCT_TYPES as t
```

No operations can be performed on the output array.

Example output

A struct types result may look like this when printed:

```
create type RGBPixel
as ( red char, green char, blue char )

create type TestPixel
as ( band1 char, band2 char, band3 char )

create type GeostatPredictionPixel
as ( prediction float, variance float )
```

An marray types result may look like this when printed:

```
create type GreyImage
as char mdarray [ x, y ]

create type RGBCube
as RGBPixel mdarray [ x, y, z ]

create type XGAImage
as RGBPixel mdarray [ x ( 0 : 1023 ), y ( 0 : 767 ) ]
```

A set types result may look like this when printed:

```
create type GreySet
as set ( GreyImage )

create type NullValueTestSet
as set ( NullValueArrayTest null values [5:7] )
```

An all types result will print combination of all struct types, marray types, and set types results.

4.5.7 Changing types

The type of an existing collection can be changed to another type through the `alter` statement.

The new collection type must be compatible with the old one, which means:

- same cell type
- same dimensionality
- no domain shrinking

Changes are allowed, for example, in the null values.

Alter type syntax

```
alter collection collName
set type collType
```

where

- *collName* is the name of an existing collection
- *collType* is the name of an existing collection type

Usage notes

The collection does not need to be empty, i.e. it may contain array objects.

Currently, only set (i.e., collection) types can be modified.

Example

Update the set type of a collection `Bathymetry` to a new set type that specifies null values:

```
alter collection Bathymetry
set type BathymetryWithNullValues
```

4.6 Query Execution with rasql

The rasdaman toolkit offers essentially a couple of ways to communicate with a database through queries:

- By submitting queries via command line using rasql; this tool is covered in this section.
- By writing a C++, Java, or Python application that uses the rasdaman APIs (raslib, rasj, or rasdapy3 respectively). See the rasdaman API guides for further details.

The rasql tool accepts a query string (which can be parametrised as explained in the API guides), sends it to the server for evaluation, and receives the result set. Results can be displayed in alphanumeric mode, or they can be stored in files.

4.6.1 Examples

For the user who is familiar with command line tools in general and the rasql query language, we give a brief introduction by way of examples. They outline the basic principles through common tasks.

- Create a collection `test` of type `GreySet` (note the explicit setting of user `rasadmin`; rasql's default user `rasguest` by default cannot write):

  ```
  rasql -q "create collection test GreySet" \
        --user rasadmin --passwd rasadmin
  ```

- Print the names of all existing collections:

  ```
  rasql -q "select r from RAS_COLLECTIONNAMES as r" \
        --out string
  ```

- Export demo collection `mr` into TIFF files rasql_1.tif, rasql_2.tif, rasql_3.tif (note the escaped double-quotes as required by shell):

  ```
  rasql -q "select encode(m, \"tiff\") from mr as m"
        --out file
  ```

- Import TIFF file *myfile* into collection `mr` as new image (note the different query string delimiters to preserve the `$` character!):

  ```
  rasql -q 'insert into mr values decode($1)' \
        -f myfile --user rasadmin --passwd rasadmin
  ```

- Put a grey square into every mr image:

```
rasql -q "update mr as m set m[0:10,0:10] \
          assign marray x in [0:10,0:10] values 127c" \
      --user rasadmin --passwd rasadmin
```

- Verify result of update query by displaying pixel values as hex numbers:

```
rasql -q "select m[0:10,0:10] from mr as m" --out hex
```

4.6.2 Invocation syntax

Rasql is invoked as a command with the query string as parameter. Additional parameters guide detailed behavior, such as authentication and result display.

Any errors or other diagnostic output encountered are printed; transactions are aborted upon errors.

Usage:

```
rasql [--query q|-q q] [options]
```

Options:

-h, --help show command line switches

-q, --query q query string to be sent to the rasdaman server for execution

-f, --file f file name for upload through $i parameters within queries; each $i needs its own file parameter, in proper sequence[4]. Requires –mdddomain and –mddtype

--content display result, if any (see also –out and –type for output formatting)

--out t use display method t for cell values of result MDDs where t is one of

- none: do not display result item contents
- file: write each result MDD into a separate file
- string: print result MDD contents as char string (only for 1D arrays of type char)
- hex: print result MDD cells as a sequence of space-separated hex values
- formatted: reserved, not yet supported

Option –out implies –content; default: none

--outfile of file name template for storing result images (ignored for scalar results). Use '%d' to indicate auto numbering position, like with

[4] Currently only one -f argument is supported (i.e., only $1).

printf(1). For well-known file types, a proper suffix is appended to the resulting file name. Implies –out file. (default: rasql_%d)

--mdddomain d	MDD domain, format: '[x0:x1,y0:y1]'; required only if –file specified and file is in data format r_Array; if input file format is some standard data exchange format and the query uses a convertor, such as encode($1,"tiff"), then domain information can be obtained from the file header.
--mddtype t	input MDD type (must be a type defined in the database); required only if –file specified and file is in data format r_Array; if input file format is some standard data exchange format and the query uses a convertor, such as decode($1,"tiff"), then type information can be obtained from the file header.
--type	display type information for results
-s, --server h	rasdaman server name or address (default: localhost)
-p, --port p	rasdaman port number (default: 7001)
-d, --database db	name of database (default: RASBASE)
--user u	name of user (default: rasguest)
--passwd p	password of user (default: rasguest)
--quiet	print no ornament messages, only results and errors

4.7 Overview: General Query Format

4.7.1 Basic Query Mechanism

rasql provides declarative query functionality on collections (i.e., sets) of MDD stored in a rasdaman database. The query language is based on the SQL-92 standard and extends the language with high-level multidimensional operators.

The general query structure is best explained by means of an example. Consider the following query:

```
select mr[100:150,40:80] / 2
from mr
where some_cells( mr[120:160, 55:75] > 250 )
```

In the **from** clause, mr is specified as the working collection on which all evaluation will take place. This name, which serves as an "iterator variable" over this collection, can be used in other parts of the query for referencing the particular collection element under inspection.

Optionally, an alias name can be given to the collection (see syntax below) - however, in most cases this is not necessary.

In the **where** clause, a condition is phrased. Each collection element in turn is probed, and upon fulfillment of the condition the item is added to the query result set. In the example query, part of the image is tested against a threshold value.

Elements in the query result set, finally, can be "post-processed" in the **select** clause by applying further operations. In the case on hand, a spatial extraction is done combined with an intensity reduction on the extracted image part.

In summary, a rasql query returns a set fulfilling some search condition just as is the case with conventional SQL and OQL. The difference lies in the operations which are available in the **select** and **where** clause: SQL does not support expressions containing multidimensional operators, whereas rasql does.

Syntax

```
select resultList
from collName [ as collIterator ]
[ , collName [ as collIterator ] ] ...
[ where booleanExp ]
```

The complete rasql query syntax can be found in the Appendix.

4.7.2 Select Clause: Result Preparation

Type and format of the query result are specified in the **select** part of the query. The query result type can be multidimensional, a struct or atomic (i.e., scalar), or a spatial domain / interval. The select clause can reference the collection iteration variable defined in the from clause; each array in the collection will be assigned to this iteration variable successively.

Example

Images from collection mr, with pixel intensity reduced by a factor 2:

```
select mr / 2
from mr
```

4.7.3 From Clause: Collection Specification

In the **from** clause, the list of collections to be inspected is specified, optionally together with a variable name which is associated to each collection. For query evaluation the cross product between all participating collections is built which means that every possible combination of elements from all collections is evaluated. For instance in case of two collections, each MDD of the first collection is combined with each MDD of the second collection. Hence, combining a collection with n elements with a collection containing m elements results in n*m combinations. This is important for estimating query response time.

Example

The following example subtracts each MDD of collection mr2 from each MDD of collection mr (the binary induced operation used in this example is explained in *Binary Induction*).

```
select mr - mr2
from mr, mr2
```

Using alias variables a and b bound to collections mr and mr2, resp., the same query looks as follows:

```
select a - b
from mr as a, mr2 as b
```

Cross products

As in SQL, multiple collections in a from clause such as

```
from c1, c2, ..., ck
```

are evaluated to a *cross product*. This means that the select clause is evaluated for a virtual collection that has n1 * n2 * … * nk elements if c1 contains n1 elements, c2 contains n2 elements, and so forth.

Warning: This holds regardless of the select expression - even if you mention only say c1 in the select clause, the number of result elements will be the product of *all* collection sizes!

4.7.4 Where Clause: Conditions

In the **where** clause, conditions are specified which members of the query result set must fulfil. Like in SQL, predicates are built as boolean expressions using comparison, parenthesis, functions, etc. Unlike SQL, however, rasql offers mechanisms to express selection criteria on multidimensional items.

Example

We want to restrict the previous result to those images where at least one difference pixel value is greater than 50 (see *Binary Induction*):

```
select mr - mr2
from mr, mr2
where some_cells( mr - mr2 > 50 )
```

4.7.5 Comments in Queries

Comments are texts which are not evaluated by the rasdaman server in any way. However, they are useful - and should be used freely - for documentation purposes; in particular for stored queries it is important that its meaning will be clear to later readers.

Syntax

```
-- any text, delimited by end of line
```

Example

```
select mr -- this comment text is ignored by rasdaman
from mr   -- for comments spanning several lines,
          -- every line needs a separate '--' starter
```

4.8 Constants

4.8.1 Atomic Constants

Atomic constants are written in standard C/C++ style. If necessary constants are augmented with a one or two letter postfix to unambiguously determine its data type (Table 4.2).

The default for integer constants is `l`, and for floating-point it is `d`. Specifiers are case insensitive.

Example

```
25c
-1700L
.4e-5D
```

Note: Boolean constants true and false are unique, so they do not need a type specifier.

Table 4.2: Data type specifiers

postfix	type
o	octet
c	char
s	short
us	unsigned short
l	long
ul	unsigned long
f	float
d	double

Additionally, the following special floating-point constants are supported as well:

Table 4.3: Special floating-point constants corresponding to IEEE 754 NaN and Inf.

Constant	Type
NaN	double
NaNf	float
Inf	double
Inff	float

Complex numbers

Special built-in types are `CFloat32` and `CFloat64` for single and double precision complex numbers, resp, as well as `CInt16` and `CInt32` for signed integer complex numbers.

Syntax

```
complex( re, im )
```

where *re* and *im* are integer or floating point expressions. The resulting constant type is summarized on the table below. Further re/im type combinations are not supported.

Table 4.4: Complex constant type depends on the type of the re and im arguments.

type of re	type of im	type of complex constant
short	short	CInt16
long	long	CInt32
float	float	CFloat32
double	double	CFloat64

Example

```
complex( .35d, 16.0d )   -- CFloat64
complex( .35f, 16.0f )   -- CFloat32
complex( 5s, 16s )       -- CInt16
complex( 5, 16 )         -- CInt32
```

Component access

The complex parts can be extracted with `.re` and `.im`; more details can be found in the *Induced Operations* section.

4.8.2 Composite Constants

Composite constants resemble records ("structs") over atomic constants or other records. Notation is as follows.

Syntax

```
struct { const_0, ..., const_n }
```

where *const_i* must be of atomic or complex type, i.e. nested structs are not supported.

Example

```
struct{ 0c, 0c, 0c }  -- black pixel in an RGB image, for example
struct{ 1l, true }    -- mixed component types
```

Component access

See *Struct Component Selection* for details on how to extract the constituents from a composite value.

4.8.3 Array Constants

Small array constants can be indicated literally. An array constant consists of the spatial domain specification (see *Spatial Domain*) followed by the cell values whereby value sequencing is as follow. The array is linearized in a way that the lowest dimension[5] is the "outermost" dimension and the highest dimension[6] is the "innermost" one. Within each dimension, elements are listed sequentially, starting with the lower bound and proceeding until the upper bound. List elements for the innermost dimension are separated by comma ",", all others by semicolon ";".

The exact number of values as specified in the leading spatial domain expression must be provided. All constants must have the same type; this will be the result array's base type.

Syntax

```
< mintervalExp
  scalarList_0 ; ... ; scalarList_n ; >
```

where *scalarList* is defined as a comma separated list of literals:

```
scalar_0, scalar_1, ... scalar_n ;
```

Example

```
< [-1:1,-2:2] 0, 1, 2, 3, 4;
              1, 2, 3, 4, 5;
              2, 3, 4, 5, 6 >
```

This constant expression defines the following matrix:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

4.8.4 Object identifier (OID) Constants

OIDs serve to uniquely identify arrays (see *Linking MDD with Other Data*). Within a database, the OID of an array is an integer number. To use an OID outside the context of a particular database, it must be fully qualified with the system name where the database resides, the name of the database containing the array, and the local array OID.

The worldwide unique array identifiers, i.e., OIDs, consist of three components:

- A string containing the system where the database resides (system name),

[5] the dimension which is the *leftmost* in the spatial domain specification

[6] the dimension which is the *rightmost* in the spatial domain specification

- A string containing the database ("base name"), and
- A number containing the local object id within the database.

The full OID is enclosed in '<' and '>' characters, the three name components are separated by a vertical bar '|'.

System and database names obey the naming rules of the underlying operating system and base DBMS, i.e., usually they are made up of lower and upper case characters, underscores, and digits, with digits not as first character. Any additional white space (space, tab, or newline characters) inbetween is assumed to be part of the name, so this should be avoided.

The local OID is an integer number.

Syntax

```
< systemName | baseName | objectID >
objectID
```

where *systemName* and *baseName* are string literals and *objectID* is an *integerExp*.

Example

```
< acme.com | RASBASE | 42 >
42
```

4.8.5 String constants

A sequence of characters delimited by double quotes is a string.

Syntax

```
"..."
```

Example

```
SELECT encode(coll, "png") FROM coll
```

4.8.6 Collection Names

Collections are named containers for sets of MDD objects (see *Linking MDD with Other Data*). A collection name is made up of lower and upper case characters, underscores, and digits. Depending on the underlying base DBMS, names may be limited in length, and some systems (rare though) may not distinguish upper and lower case letters.

Operations available on name constants are string equality "=" and inequality "!=".

4.9 Spatial Domain Operations

4.9.1 One-Dimensional Intervals

One-dimensional (1D) intervals describe non-empty, consecutive sets of integer numbers, described by integer-valued lower and upper bound, resp.; negative values are admissible for both bounds. Intervals are specified by indicating lower and upper bound through integer-valued expressions according to the following syntax:

The lower and upper bounds of an interval can be extracted using the functions .lo and .hi.

Syntax

```
integerExp_1 : integerExp_2
intervalExp.lo
intervalExp.hi
```

A one-dimensional interval with *integerExp_1* as lower bound and *integerExp_2* as upper bound is constructed. The lower bound must be less or equal to the upper bound. Lower and upper bound extractors return the integer-valued bounds.

Examples

An interval ranging from -17 up to 245 is written as:

```
-17 : 245
```

Conversely, the following expression evaluates to 245; note the parenthesis to enforce the desired evaluation sequence:

```
(-17 : 245).hi
```

4.9.2 Multidimensional Intervals

Multidimensional intervals (*m-intervals*) describe areas in space, or better said: point sets. These point sets form rectangular and axis-parallel "cubes" of some dimension. An m-interval's dimension is given by the number of 1D intervals it needs to be described; the bounds of the "cube" are indicated by the lower and upper bound of the respective 1D interval in each dimension.

From an m-interval, the intervals describing a particular dimension can be extracted by indexing the m-interval with the number of the desired dimension using the operator `[]`.

Dimension counting in an m-interval expression runs from left to right, starting with lowest dimension number 0.

Syntax

```
[ intervalExp_0 , ... , intervalExp_n ]
[ intervalExp_0 , ... , intervalExp_n ] [integerExp ]
```

An (n+1)-dimensional m-interval with the specified *intervalExp_i* is built where the first dimension is described by *intervalExp_0*, etc., until the last dimension described by *intervalExp_n*.

Example

A 2-dimensional m-interval ranging from -17 to 245 in dimension 1 and from 42 to 227 in dimension 2 can be denoted as

```
[ -17 : 245, 42 : 227 ]
```

The expression below evaluates to [42:227].

```
[ -17 : 245, 42 : 227 ] [1]
```

...whereas here the result is 42:

```
[ -17 : 245, 42 : 227 ] [1].lo
```

4.10 Array Operations

As we have seen in the last Section, *intervals* and *m-intervals* describe n-dimensional regions in space.

Next, we are going to place information into the regular grid established by the m-intervals so that, at the position of every integer-valued coordinate, a value can be stored. Each such value container addressed by an n-dimensional coordinate will be referred to as a *cell*. The set of all the cells described by a particular m-interval and with cells over a particular base type, then, forms the *array*.

As before with intervals, we introduce means to describe arrays through expressions, i.e., to derive new arrays from existing ones. Such operations can change an arrays shape and dimension (sometimes called geometric operations), or the cell values (referred to as value-changing operations), or both. In extreme cases, both array dimension, size, and base type can change completely, for example in the case of a histogram computation.

First, we describe the means to query and manipulate an array's spatial domain (so-called geometric operations), then we introduce the means to query and manipulate an array's cell values (value-changing operations).

Note that some operations are restricted in the operand domains they accept, as is common in arithmetics in programming languages; division by zero is a common example. *Arithmetic Errors and Other Exception Situations* contains information about possible error conditions, how to deal with them, and how to prevent them.

4.10.1 Spatial Domain

The m-interval covered by an array is called the array's *spatial domain*. Function sdom() allows to retrieve an array's current spatial domain. The *current domain* of an array is the minimal axis-parallel bounding box containing all currently defined cells.

As arrays can have variable bounds according to their type definition (see *Array types*), their spatial domain cannot always be determined from the schema information, but must be recorded individually by the database system. In case of a fixed-size array, this will coincide with the schema information, in case of a variable-size array it delivers the spatial domain to which the array has been set. The operators presented below and in *Update* allow to change an array's spatial domain. Notably, a collection defined over variable-size arrays can hold arrays which, at a given moment in time, may differ in the lower and/or upper bounds of their variable dimensions.

Syntax

```
sdom( mddExp )
```

Function sdom() evaluates to the current spatial domain of *mddExp*.

Examples

Consider an image a of collection mr. Elements from this collection are defined as having free bounds, but in practice our collection elements all have spatial domain [0 : 255, 0 : 210]. Then, the following equivalences hold:

```
sdom(a)        = [0 : 255, 0 : 210]
sdom(a)[0]     = [0 : 255]
sdom(a)[0].lo = 0
sdom(a)[0].hi = 255
```

4.10.2 Geometric Operations

Trimming

Reducing the spatial domain of an array while leaving the cell values unchanged is called *trimming*. Array dimension remains unchanged. Attempting to extend or intersect the array's spatial domain will lead to an error; use the *extend function* in this case.

Syntax

```
mddExp [ mintervalExp ]
```

Examples

The following query returns cutouts from the area [120: 160 , 55 : 75] of all images in collection `mr` (see Figure 4.7).

```
select mr[ 120:160, 55:75 ]
from mr
```

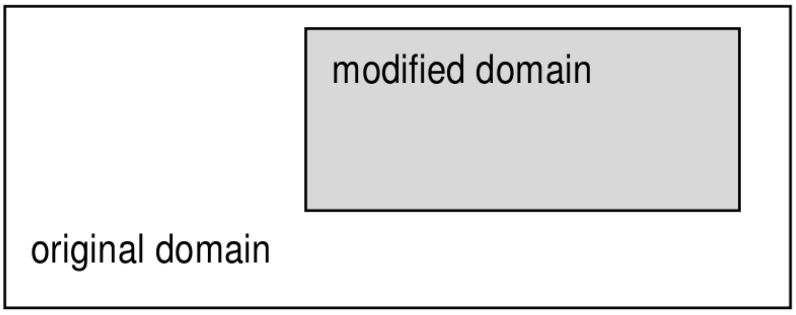


Figure 4.6: Spatial domain modification through trimming (2-D example)

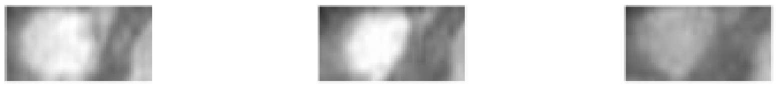


Figure 4.7: Trimming result

Section

A *section* allows to extract lower-dimensional layers (“slices”) from an array.

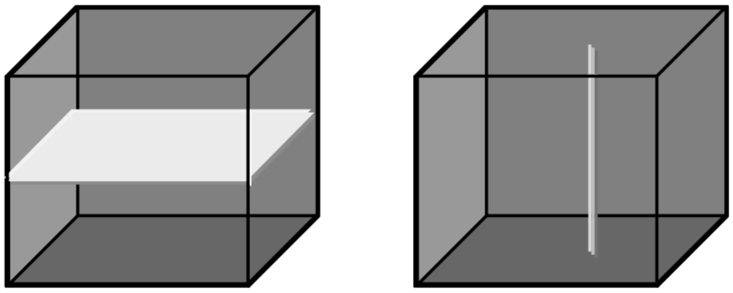


Figure 4.8: Single and double section through 3-D array, yielding 2-D and 1-D sections.

A section is accomplished through a trim expression by indicating the slicing position rather than a selection interval. A section can be made in any dimension within a trim expression. Each section reduces the dimension by one.

Like with trimming, a section must be within the spatial domain of the array, otherwise an error indicating that the subset domain extends outside of the array spatial domain will be thrown.

Syntax

```
mddExp [ integerExp_0 , ... , integerExp_n ]
```

This makes sections through *mddExp* at positions *integerExp_i* for each dimension *i*.

Example

The following query produces a 2-D section in the 2nd dimension of a 3-D cube:

```
select Images3D[ 0:256, 10, 0:256 ]
from Images3D
```

Note: If a section is done in *every* dimension of an array, the result is one single cell. This special case resembles array element access in programming languages, e.g., C/C++. However, in rasql the result still is an array, namely one with zero dimensions and exactly one element.

Example

The following query delivers a set of 0-D arrays containing single pixels, namely the ones with coordinate [100,150]:

```
select mr[ 100, 150 ]
from mr
```

Trim Wildcard Operator "*"

An asterisk "*" can be used as a shorthand for an sdom() invocation in a trim expression; the following phrases all are equivalent:

```
a [ *:*, *:* ] = a [ sdom(a)[0] , sdom(a)[1] ]
               = a [ sdom(a)[0].lo : sdom(a)[0].hi ,
                     sdom(a)[1].lo : sdom(a)[1].hi ]
```

An asterisk "*" can appear at any lower or upper bound position within a trim expression denoting the current spatial domain boundary. A trim expression can contain an arbitrary number of such wildcards. Note, however, that an asterisk cannot be used for specifying a section.

Example

The following are valid applications of the asterisk operator:

```
select mr[ 50:*, *:200 ]
from mr

select mr[ *:*, 10:150 ]
from mr
```

The next is illegal because it attempts to use an asterisk in a section:

```
select mr[ *, 100:200 ] -- illegal "*" usage in dimension 0
from mr
```

Note

It is well possible (and often recommended) to use an array's spatial domain or part of it for query formulation; this makes the query more general and, hence, allows to establish query libraries. The following query cuts away the rightmost pixel line from the images:

```
select mr[ *:*, *:sdom(mr)[1].hi - 1 ]    -- good, portable
from mr
```

In the next example, conversely, trim bounds are written explicitly; this query's trim expression, therefore, cannot be used with any other array type.

```
select mr[ 0:767, 0:1023 ]                -- bad, not portable
from mr
```

One might get the idea that the last query evaluates faster. This, however, is not the case; the server's intelligent query engine makes the first version execute at just the same speed.

Positionally-independent Subsetting

Rasdaman supports positionally-independent subsetting like in WCPS and SQL/MDA, where for each trim/slice the axis name is indicated as well, e.g.

```
select mr2[d0(0:100), d1(50)] from mr2
```

The axis names give a reference to the addressed axes, so the order doesn't matter anymore. This is equivalent:

```
select mr2[d1(50), d0(0:100)] from mr2
```

Furthermore, not all axes have to be specified. Any axes which are not specified default to ":". For example:

```
select mr2[d1(50)] from mr2
=
select mr2[d0(*:*), d1(50)] from mr2
```

The two subset formats cannot be mixed, e.g. this is an error:

```
select mr2[d0(0:100), 50] from mr2
```

Shifting a Spatial Domain

Built-in function shift() transposes an array: its spatial domain remains unchanged in shape, but all cell contents simultaneously are moved to another location in n-dimensional space. Cell values themselves remain unchanged.

Syntax

```
shift( mddExp , pointExp )
```

The function accepts an *mddExp* and a *pointExp* and returns an array whose spatial domain is shifted by vector *pointExp*.

Example

The following expression evaluates to an array with spatial domain `[3:13, 4:24]`. Containing the same values as the original array a.

```
shift( a[ 0:10, 0:20 ], [ 3, 4 ] )
```

Extending a Spatial Domain

Function extend() enlarges a given MDD with the domain specified. The domain for extending must, for every boundary element, be at least as large as the MDD's domain boundary. The new MDD contains 0 values in the extended part of its domain and the MDD's original cell values within the MDD's domain.

Syntax

```
extend( mddExp , mintervalExp )
```

The function accepts an *mddExp* and a *mintervalExp* and returns an array whose spatial domain is extended to the new domain specified by *mintervalExp*. The result MDD has the same cell type as the input MDD.

Precondition:

```
sdom( mddExp ) contained in mintervalExp
```

Example

Assuming that MDD `a` has a spatial domain of `[0:50, 0:25]`, the following expression evaluates to an array with spatial domain `[-100:100, -50:50]`, `a`'s values in the subdomain `[0:50, 0:25]`, and 0 values at the remaining cell positions.

```
extend( a, [-100:100, -50:50] )
```

Geographic projection

Overview

"A map projection is any method of representing the surface of a sphere or other three-dimensional body on a plane. Map projections are necessary for creating maps. All map projections distort the surface in some fashion. Depending on the purpose of the map, some distortions are acceptable and others are not; therefore different map projections exist in order to preserve some properties of the sphere-like body at the expense of other properties." (Wikipedia)

Each coordinate tieing a geographic object, map, or pixel to some position on earth (or some other celestial object, for that matter) is valid only in conjunction with the Coordinate Reference System (CRS) in which it is expressed. For 2-D Earth CRSs, a set of CRSs and their identifiers is normatively defined by the OGP Geomatics Committee, formed in 2005 by the absorption into OGP of the now-defunct European Petroleum Survey Group (EPSG). By way of tradition, however, this set of CRS definitions still is known as "EPSG", and the CRS identifiers as "EPSG codes". For example, EPSG:4326 references the well-known WGS84 CRS.

The `project()` function

Assume an MDD object `M` and two CRS identifiers `C1` and `C2` such as "EPSG:4326". The `project()` function establishes an output MDD, with same dimension as `M`, whose contents is given by projecting `M` from CRS `C1` into CRS `C2`.

The `project()` function comes in several variants based on the provided input arguments

```
(1) project( mddExpr, boundsIn, crsIn, crsOut )

(2) project( mddExpr, boundsIn, crsIn, crsOut, resampleAlg )

(3) project( mddExpr, boundsIn, crsIn, boundsOut, crsOut,
                      widthOut, heightOut )

(4) project( mddExpr, boundsIn, crsIn, boundsOut, crsOut,
                      widthOut, heightOut, resampleAlg,␣
↪errThreshold )

(5) project( mddExpr, boundsIn, crsIn, boundsOut, crsOut,
                      xres, yres)

(6) project( mddExpr, boundsIn, crsIn, boundsOut, crsOut,
                      xres, yres, resampleAlg, errThreshold )
```

where

- `mddExpr` - MDD object to be reprojected.
- `boundsIn` - geographic bounding box given as a string of comma-separated floating-point values of the format: `"xmin, ymin, xmax, ymax"`.
- `crsIn` - geographic CRS as a string. Internally, the `project()` function is mapped to GDAL; hence, it accepts the same CRS formats as GDAL:
  - Well Known Text (as per GDAL)
  - "EPSG:n"
  - "EPSGA:n"
  - "AUTO:proj_id,unit_id,lon0,lat0" indicating OGC WMS auto projections
  - "`urn:ogc:def:crs:EPSG::n`" indicating OGC URNs (deprecated by OGC)
  - PROJ.4 definitions
  - well known names, such as NAD27, NAD83, WGS84 or WGS72.
  - WKT in ESRI format, prefixed with "ESRI::"
  - "IGNF:xxx" and "+init=IGNF:xxx", etc.

  - Since recently (v1.10), GDAL also supports OGC CRS URLs, OGC's preferred way of identifying CRSs.

- `boundsOut` - geographic bounding box of the projected output, given in the same format as `boundsIn`. This can be "smaller" than the input bounding box, in which case the input will be cropped.
- `crsOut` - geographic CRS of the result, in same format as `crsIn`.
- `widthOut`, `heightOut` - integer grid extents of the result; the result will be accordingly scaled to fit in these extents.
- `xres`, `yres` - axis resolution in target georeferenced units.
- `resampleAlg` - resampling algorithm to use, equivalent to the ones in GDAL:

  near Nearest neighbour (default, fastest algorithm, worst interpolation quality).

  bilinear Bilinear resampling (2x2 kernel).

  cubic Cubic convolution approximation (4x4 kernel).

  cubicspline Cubic B-spline approximation (4x4 kernel).

  lanczos Lanczos windowed sinc (6x6 kernel).

  average Average of all non-NODATA contributing pixels. (GDAL >= 1.10.0)

  mode Selects the value which appears most often of all the sampled points. (GDAL >= 1.10.0)

  max Selects the maximum value from all non-NODATA contributing pixels. (GDAL >= 2.0.0)

  min Selects the minimum value from all non-NODATA contributing pixels. (GDAL >= 2.0.0)

  med Selects the median value of all non-NODATA contributing pixels. (GDAL >= 2.0.0)

  q1 Selects the first quartile value of all non-NODATA contributing pixels. (GDAL >= 2.0.0)

  q3 Selects the third quartile value of all non-NODATA contributing pixels. (GDAL >= 2.0.0)

- `errThreshold` - error threshold for transformation approximation (in pixel units - defaults to 0.125).

Example

The following expression projects the MDD `worldMap` with bounding box "-180, -90, 180, 90" in CRS EPSG 4326, into EPSG 54030:

```
project( worldMap, "-180, -90, 180, 90", "EPSG:4326", "EPSG:54030" )
```

The next example reprojects a subset of MDD `Formosat` with geographic bbox “265725, 2544015, 341595, 2617695” in EPSG 32651, to bbox “120.630936455 23.5842129067 120.77553782 23.721772322” in EPSG 4326 fit into a 256 x 256 pixels area. The resampling algorithm is set to bicubic, and the pixel error threshold is 0.1.

```
project( Formosat[ 0:2528, 0:2456 ],
  "265725, 2544015, 341595, 2617695", "EPSG:32651",
  "120.630936455 23.5842129067 120.77553782 23.721772322",
↪"EPSG:4326",
  256, 256, cubic, 0.1 )
```

Limitations

Only 2-D arrays are supported. For multiband arrays, all bands must be of the same cell type.

Notes

Reprojection implies resampling of the cell values into a new grid, hence usually they will change.

As for the resampling process typically a larger area is required than the reprojected data area itself, it is advisable to project an area smaller than the total domain of the MDD.

Per se, rasdaman is a domain-agnostic Array DBMS and, hence, does not know about CRSs; specific geo semantics is added by rasdaman’s petascope layer. However, for the sake of performance, the reprojection capability – which in geo service practice is immensely important – is pushed down into rasdaman, rather than doing reprojection in petascope’s Java code. To this end, the `project()` function provides rasdaman with enough information to perform a reprojection, however, without “knowing” anything in particular about geographic coordinates and CRSs. One consequence is that there is no check whether this lat/long project is applied to the proper axis of an array; it is up to the application (usually: petascope) to handle axis semantics.

One consequence is that there is no check whether this lat/long project is applied to the proper axis of an array; it is up to the application (usually: petascope) to handle axis semantics.

4.10.3 Clipping Operations

Clipping is a general operation covering polygon clipping, linestring selection, polytope clipping, curtain queries, and corridor queries. Presently, all operations are available in rasdaman via the `clip` function.

Further examples of clipping can be found in the systemtest for clipping.

Polygons

Syntax

```
select clip( c, polygon(( list of WKT points )) )
from coll as c
```

The input consists of an MDD expression and a list of WKT points, which determines the set of vertices of the polygon. Polygons are assumed to be closed with positive area, so the first vertex need not be repeated at the end, but there is no problem if it is. The algorithms used support polygons with self-intersection and vertex re-visitation.

Polygons may have interiors defined, such as

```
polygon( ( 0 0, 9 0, 9 9, 0 9, 0 0),
         ( 3 3, 7 3, 7 7, 3 7, 3 3 ) )
```

which would describe the annular region of the box `[0:9,0:9]` with the interior box `[3:7, 3:7]` removed. In this case, the interior polygons (there may be many, as it forms a list) must not intersect the exterior polygon.

Multipolygons

Syntax

```
select clip( c, multipolygon((( list of WKT points )),(( list of␣
↪WKT points ))...) )
from coll as c
```

The input consists of an MDD expression and a list of polygons defined by list of WKT points. The assumptions about polygons are same as the ones for Polygon.

Return type

The output of a polygon query is a new array with dimensions corresponding to the bounding box of the polygon vertices, and further restricted to the collection's spatial domain. In case of Multipolygon, the new array have dimensions corresponding to closure of bounding boxes of every individual polygon, which domain intersects the collection's spatial domain. The data in the array consists of null values where cells lie outside the polygon (or 0 values if no null values are associated with the array) and otherwise consists of the data in the collection where the corresponding cells lie inside the polygon. This could change the null values stored outside the polygon from one null value to another null value, in case a range of null values is used. By default, the first available null value will be utilized for the complement of the polygon.

An illustrative example of a polygon clipping is the right triangle with vertices located at `(0, 0,0)`, `(0,10,0)` and `(0,10,10)`, which can be selected via the following query:

```
select clip( c, polygon((0 0 0, 0 10 0, 0 10 10)) )
from coll as c
```

Oblique polygons with subspacing

In case all the points in a polygon are coplanar, in some MDD object `d` of higher dimension than 2, users can first perform a subspace operation on `d` which selects the 2-D oblique subspace of `d` containing the polygon. For example, if the polygon is the triangle `polygon((0 0 0, 1 1 1, 0 1 1, 0 0 0))`, this triangle can be selected via the following query:

```
select clip( subspace(d, (0 0 0, 1 1 1, 0 1 1) ),
             polygon(( 0 0, 1 1 , 0 1 , 0 0)) )
from coll as d
```

where the result of `subspace(d)` is used as the domain of the polygon. For more information look in *Subspace Queries*.

Linestrings

Syntax

```
select clip( c, linestring( list of WKT points ) ) [ with␣
↪coordinates ]
from coll as c
```

The input parameter `c` refers to an MDD expression of dimension equal to the dimension of the points in the list of WKT points. The list of WKT points consists of parameters such as `linestring(0 0, 19 -3, 19 -21)`, which would describe the 3 endpoints of 2 line segments sharing an endpoint at `19 -3`, in this case.

Return type

The output consists of a 1-D MDD object consisting of the points selected along the path drawn out by the linestring. The points are selected using a Bresenham Line Drawing algorithm which passes through the spatial domain in the MDD expression `c`, and selects values from the stored object. In case the linestring spends some time outside the spatial domain of `c`, the first null value will be used to fill the result of the linestring, just as in polygon clipping.

When `with coordinates` is specified, in addition to the original cell values the coordinate values are also added to the result MDD. The result cell type for clipped MDD of dimension N will be composite of the following form:

1. If the original cell type `elemtype` is non-composite:

   ```
   { long d1, ..., long dN, elemtype value }
   ```

2. Otherwise, if the original cell type is composite of `M` bands:

   ```
   { long d1, ..., long dN, elemtype1 elemname1, ..., elemetypeM␣
   ↪elemnameM }
   ```

Example

Select a Linestring from rgb data `with coordinates`. First two values of each cell in the result are the x/y coordinates, with following values (three in this case for RGB data) are the cell values of the clip operation to which `with coordinates` is applied.

```
select encode(
    clip( c, linestring(0 19, 19 24, 12 17) ) with coordinates,
↪"json")
from rgb as c
```

Result:

```
["0 19 119 208 248","1 19 119 208 248","2 20 119 208 248", ...]
```

The same query without specifying `with coordinates`:

```
select encode(
    clip( c, linestring(0 19, 19 24, 12 17) ), "json")
from rgb as c
```

results in

```
["119 208 248","119 208 248","119 208 248", ...]
```

Curtains

Syntax

```
select clip( c, curtain( projection(dimension pair),
                         polygon(( ... )) ) )
from coll as c
```

and

```
select clip( c, curtain( projection(dimension list),
                         linestring( ... ) ) )
from coll as c
```

The input in both variants consists of a dimension list corresponding to the dimensions in which the geometric object, either the polygon or the linestring, is defined. The geometry object is defined as per the above descriptions; however, the following caveat applies: the spatial domain of the mdd expression is projected along the projection dimensions in the `projection(dimension list)`. For a polygon clipping, which is 2-D, the dimension list is a pair of values such as `projection(0, 2)` which would define a polygon in the axial dimensions of 0 and 2 of the MDD expression `c`. For instance, if the spatial domain of `c` is `[0:99,0:199,0:255]`, then this would mean the domain upon which the polygon is defined would be `[0:99,0:255]`.

Return type

The output consists of a polygon clipping at every slice of the spatial domain of `c`. For instance, if the projection dimensions of `(0, 2)` are used for the same spatial domain of `c` above, then a polygon clipping is performed at every slice of `c` of the form `[0:99,x,0:255]` and appended to the result MDD object, where there is a slice for each value of x in `[0:199]`.

Corridors

Syntax

```
select clip( c, corridor( projection(dimension pair),
                          linestring( ... ),
                          polygon(( ... )) ) )
from coll as c
```

and

```
select clip( c, corridor( projection(dimension pair),
                          linestring( ... ),
                          polygon(( ... )),
                          discrete ) )
from coll as c
```

The input consists of a dimension list corresponding to the dimensions in which the geometric object, in this case a polygon, is defined. The linestring specifies the path along which this geometric object is integrated. One slice is sampled at every point, and at least the first point of the linestring should be contained within the polygon to ensure a meaningful result (an error is thrown in case it is not). There is an optional *discrete* flag which modifies the output by skipping the extrapolation of the linestring data to interior points.

Return type

The output consists of a polygon clipping at every slice of the spatial domain of `c` translated along the points in the linestring, where the first axis of the result is indexed by the linestring points and the latter axes are indexed by the mask dimensions (in this case, the convex hull of the polygon). The projection dimensions are otherwise handled as in curtains; it is the spatial offsets given by the linestring coordinates which impact the changes in the result. In the case where the *discrete* parameter was utilized, the output is indexed by the number of points in the linestring description in the query and not by the extrapolated linestring, which uses a Bresenham algorithm to find the grid points in between.

Subspace Queries

Here we cover the details of subspace queries in rasdaman. Much like slicing via a query such as

```
select c[0:9,1,0:9] from collection as c
```

the subspace query parameter allows users to extract a lower-dimensional dataset from an existing collection. It is capable of everything that a slicing query is capable of, and more. The limitation of slicing is that the selected data must lie either parallel or perpendicular to existing axes; however, with subspacing, users can arbitrarily rotate the axes of interest to select data in an oblique fashion. This control is exercised by defining an affine subspace from a list of vertices lying in the datacube. Rasdaman takes these points and finds the unique lowest-dimensional affine subspace containing them, and outputs the data closest to this slice, contained in the bounding box of the given points, into the resulting array.

Structure of the query:

```
select clip( c, subspace(list of WKT points) )
from coll as c
```

We can illustrate the usage with an example of two queries which are identical in output:

```
select clip( c, subspace(0 0 0, 1 0 0, 0 0 1) ) from coll as c

select c[0:1,0,0:1] from coll as c
```

This example will result in 1D array of sdom `[0:99]`:

```
select clip( c, subspace(19 0, 0 99) ) from test_rgb as c
```

This example will result in a a 2D array of sdom `[0:7,0:19]`:

```
select clip( c, subspace(0 0 0, 0 19 0, 7 0 7) )
from test_grey3d as c
```

and it will consist of the best integer lattice points reachable by the vectors `(1,0,1)` and `(0,1,0)` within the bounding box domain of `[0:7,0:19,0:7]` in `test_grey3d`.

Generally speaking, rasdaman uses the 1st point as a basepoint for an affine subspace containing all given points, constructs a system of equations to determine whether or not a point is in that subspace or not, and then searches the bounding box of the given points for solutions to the projection operator which maps `[0:7,0:19,0:7]` to `[0:7,0:19]`. The result dimensions are chosen such that each search yields a unique real solution, and then rasdaman rounds to the nearest integer cell before adding the value stored in that cell to the result object.

Some mathematical edge cases:

Because of arithmetic on affine subspaces, the following two queries are fundamentally identical to rasdaman:

```
select clip( c, subspace(0 0 0, 1 1 0, 0 1 0) )
from test_grey3d as c

select clip( c, subspace(0 0 0, 1 0 0, 0 1 0) )
from test_grey3d as c
```

Rasdaman's convention is to use the first point as the translation point, and constructs the vectors generating the subspace from the differences. There is no particular reason not to use another point in the WKT list; however, knowing this, users should be aware that affine subspaces differ slightly from vector subspaces in that the following two queries differ:

```
select clip( c, subspace(10 10 10, 0 0 10, 10 0 10) )
from test_grey3d as c

select clip( c, subspace(0 0 0, 10 10 0, 0 10 0) )
from test_grey3d as c
```

The two queries have the same result domains of `[0:10,0:10]`, and the projection for both lie on the first 2 coordinate axes since the 3rd coordinate remains constant; however, the data selections differ because the subspaces generated by these differ, even though the generating vectors of `(1 1 0)` and `(0 1 0)` are the same.

Even though the bounding box where one searches for solutions is the same between these two queries, there is no way to reach the origin with the vectors `(1 1 0)` and `(0 1 0)` starting at the base point of `(10 10 10)` because neither vector can impact the 3rd coordinate value of 10; similarly, starting at `(0 0 0)` must leave the third coordinate fixed at 0. There is nothing special about choosing the first coordinate as our base point – the numbers might change, but the resulting data selections in both queries would remain constant.

The following two queries generate the same subspace, but the latter has a larger output domain:

```
select clip( c, subspace(0 0 0, 1 1 0, 0 1 0) )
from test_grey3d as c

select clip( c, subspace(0 0 0, 1 1 0, 0 1 0, 0 0 0, 1 2 0) )
from test_grey3d as c
```

As much redundancy as possible is annihilated during a preprocessing stage which uses a Gram-Schmidt procedure to excise extraneous data imported during query time, and with this algorithm, rasdaman is able to determine the correct dimension of the output domain.

Some algorithmic caveats:

The complexity of searching for a solution for each result cell is related to the codimension of the affine subspace, and not the dimension of the affine subspace itself. In fact, if `k` is the difference between the dimension of the collection array and the dimension of the result array, then each cell is determined in O(k^2) time. Preprocessing happens once for the entire query, and occurs in O(k^3) time. There is one exception to the codimensionality considerations: a 1-D affine subspace (also known as a line segment) is selected using a multidimensional generalization of the Bresenham Line Algorithm, and so the results are determined in O(n) time, where n is the dimension of the collection.

Tip: If you want a slice which is parallel to axes, then you are better off using the classic slicing style of:

```
select c[0:19,0:7,0] from collection as c
```

as the memory offset computations are performed much more efficiently.

4.10.4 Induced Operations

Induced operations allow to simultaneously apply a function originally working on a single cell value to all cells of an MDD. The result MDD has the same spatial domain, but can change its base type.

Examples

```
img.green + 5 c
```

This expression selects component named "green" from an RGB image and adds 5 (of type char, i.e., 8 bit) to every pixel.

```
img1 + img2
```

This performs pixelwise addition of two images (which must be of equal spatial domain).

Induction and structs

Whenever induced operations are applied to a composite cell structure ("structs" in C/C++), then the induced operation is executed on every structure component. If some cell structure component turns out to be of an incompatible type, then the operation as a whole aborts with an error.

For example, a constant can be added simultaneously to all components of an RGB image:

```
select rgb + 5
from rgb
```

Induction and complex

Complex numbers, which actually form a composite type supported as a base type, can be accessed with the record component names re and im for the real and the imaginary part, resp.

Example

The first expression below extracts the real component, the second one the imaginary part from a complex number c:

```
c.re
c.im
```

Unary Induction

Unary induction means that only one array operand is involved in the expression. Two situations can occur: Either the operation is unary by nature (such as boolean not); then, this operation is applied to each array cell. Or the induce operation combines a single value (scalar) with the array; then, the contents of each cell is combined with the scalar value.

A special case, syntactically, is the struct/complex component selection (see next subsection).

In any case, sequence of iteration through the array for cell inspection is chosen by the database server (which heavily uses reordering for query optimisation) and not known to the user.

Syntax

```
unaryOp mddExp
mddExp binaryOp scalarExp
scalarExp binaryOp mddExp
```

Example

The red images of collection rgb with all pixel values multiplied by 2:

```
select rgb.red * 2c
from rgb
```

Note that the constant is marked as being of type char so that the result type is minimized (short). Omitting the "`c`" would lead to an addition of long integer and char, resulting in long integer with 32 bit per pixel. Although pixel values obviously are the same in both cases, the second alternative requires twice the memory space. For more details visit the *Type Coercion Rules* section.

Struct Component Selection

Component selection from a composite value is done with the dot operator well-known from programming languages. The argument can either be a number (starting with 0) or the struct element name. Both statements of the following example would select the green plane of the sample RGB image.

This is a special case of a unary induced operator.

Syntax

```
mddExp.attrName
mddExp.intExp
```

Examples

```
select rgb.green
from rgb

select rgb.1
from rgb
```

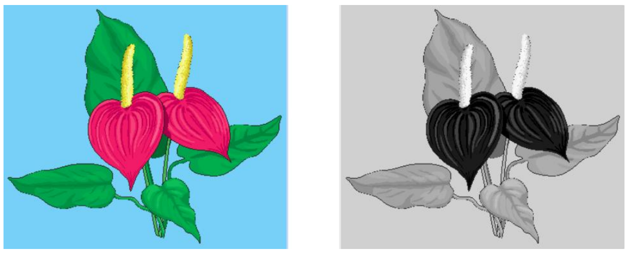


Figure 4.9: RGB image and green component

Note

Aside of operations involving base types such as integer and boolean, combination of complex base types (structs) with scalar values are supported. In this case, the operation is applied to each element of the structure in turn.

Examples

The following expression reduces contrast of a color image in its red, green, and blue channel simultaneously:

```
select rgb / 2c
from rgb
```

An advanced example is to use image properties for masking areas in this image. In the query below, this is done by searching pixels which are "sufficiently green" by imposing a lower bound on the green intensity and upper bounds on the red and blue intensity. The resulting boolean matrix is multiplied with the original image (i.e., componentwise with the red, green, and blue pixel component); the final image, then, shows the original pixel value where green prevails and is {0,0,0} (i.e., black) otherwise (Figure 4.10)

```
select rgb * ( (rgb.green > 130c) and
               (rgb.red   < 110c) and
               (rgb.blue  < 140c) )
from rgb
```

Note: This mixing of boolean and integer is possible because the usual C/C++ interpretation of true as 1 and false as 0 is supported by rasql.

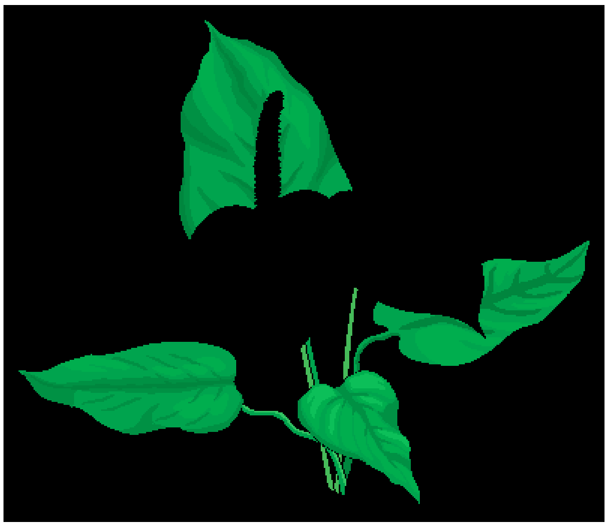


Figure 4.10: Suppressing "non-green" areas

Binary Induction

Binary induction means that two arrays are combined.

Syntax

```
mddExp binaryOp mddExp
```

Example

The difference between the images in the `mr` collection and the image in the `mr2` collection:

```
select mr - mr2
from mr, mr2
```

Note

Two cases have to be distinguished:

- Both left hand array expression and right hand array expression operate on the same array, for example:

  ```
  select rgb.red - rgb.green
  from rgb
  ```

  In this case, the expression is evaluated by combining, for each coordinate position, the respective cell values from the left hand and right hand side.

- Left hand array expression and right hand array expression operate on different arrays, for example:

  ```
  select mr - mr2
  from mr, mr2
  ```

  This situation specifies a cross product between the two collections involved. During evaluation, each array from the first collection is combined with each member of the second collection. Every such pair of arrays then is processed as described above.

Obviously the second case can become computationally very expensive, depending on the size of the collections involved - if the two collections contain n and m members, resp., then n*m combinations have to be evaluated.

Case statement

The rasdaman **case** statement serves to model n-fold case distinctions based on the SQL92 CASE statement which essentially represents a list of IF-THEN statements evaluated sequentially until either a condition fires and delivers the corresponding result or the (mandatory) ELSE alternative is returned.

In the simplest form, the **case** statement looks at a variable and compares it to different alternatives for finding out what to deliver. The more involved version allows general predicates in the condition.

This functionality is implemented in rasdaman on both scalars (where it resembles SQL) and on MDD objects (where it establishes an induced operation). Due to the construction of the rasql syntax, the distinction between scalar and induced operations is not reflected explicitly in the syntax, making query writing simpler.

Syntax

- Variable-based variant:

  ```
  case generalExp
  when scalarExp then generalExp
  ...
  else generalExp
  end
  ```

  All *generalExp*s must be of a compatible type.

- Expression-based variant:

  ```
  case
  when booleanExp then generalExp
  ...
  else generalExp
  end
  ```

  All *generalExp*'s must evaluate to a compatible type.

Example

Traffic light classification of an array object can be done as follows.

```
select
  case
  when mr > 150 then { 255c, 0c, 0c }
```

(continues on next page)

(continued from previous page)

```
    when mr > 100 then { 0c, 255c, 0c }
    else               { 0c, 0c, 255c }
    end
from mr
```

This is equivalent to the following query; note that this query is less efficient due to the increased number of operations to be evaluated, the expensive multiplications, etc:

```
select
    (mr > 150)                 { 255c, 0c, 0c }
+ (mr <= 150 and mr > 100)     { 0c, 255c, 0c }
+ (mr <= 100)                  { 0c, 0c, 255c }
from mr
```

Restrictions

In the current version, all MDD objects participating in a **case** statement must have the same tiling. Note that this limitation can often be overcome by factoring divergingly tiled arrays out of a query, or by resorting to the query equivalent in the above example using multiplication and addition.

Induction: All Operations

Below is a complete listing of all cell level operations that can be induced, both unary and binary. Supported operand types and rules for deriving the result types for each operation are specified in *Type Coercion Rules*.

+, -, *, / For each cell within some MDD value (or evaluated MDD expression), add it with the corresponding cell of the second MDD parameter. For example, this code adds two (equally sized) images:

```
img1 + img2
```

div, mod In contrast to the previous operators, div and mod are binary functions. The difference of `div` to `/` is that in the case of integer inputs, `div` results in integer result, and hence must check for division with 0, in which case an error would be thrown. The behaviour of `mod` is the same. Example usage:

```
div(a, b)
mod(a, b)
```

pow, power The power function can be written as `pow` or `power`. The signature is:

```
pow( base, exp )
```

where *base* is an MDD or scalar and *exp* is a floating point number.

=, <, >, <=, >=, != For two MDD values (or evaluated MDD expressions), compare for each coordinate the corresponding cells to obtain the Boolean result indicated by the operation.

These comparison operators work on all atomic cell types.

On composite cells, only = and != are supported; both operands must have a compatible cell structure. In this case, the comparison result is the conjunction ("and" connection) of the pairwise comparison of all cell components.

and, or, xor, is, not For each cell within some Boolean MDD (or evaluated MDD expression), combine it with the second MDD argument using the logical operation `and`, `or`, or `xor`. The `is` operation is equivalent to = (see below). The signature of the binary induced operation is

```
is, and, or, xor: mddExp, intExp -> mddExp
```

Unary function `not` negates each cell value in the MDD.

min, max For two MDD values (or evaluated MDD expressions), take the minimum / maximum for each pair of corresponding cell values in the MDDs.

Example:

```
a min b
```

For struct valued MDD values, struct components in the MDD operands must be pairwise compatible; comparison is done in lexicographic order with the first struct component being most significant and the last component being least significant.

overlay The overlay operator allows to combine two equally sized MDDs by placing the second one "on top" of the first one, informally speaking. Formally, overlaying is done in the following way:

- wherever the second operand's cell value is not zero and not null, the result value will be this value.
- wherever the second operand's cell value is zero or null, the first argument's cell value will be taken.

This way stacking of layers can be accomplished, e.g., in geographic applications. Consider the following example:

```
ortho overlay tk.water overlay tk.streets
```

When displayed the resulting image will have streets on top, followed by water, and at the bottom there is the ortho photo.

Strictly speaking, the overlay operator is not atomic. Expression

```
a overlay b
```

is equivalent to

```
(b is not null) * b + (b is null) * a
```

However, on the server the overlay operator is executed more efficiently than the above expression.

bit(mdd, pos) For each cell within MDD value (or evaluated MDD expression) mdd, take the bit with nonnegative position number pos and put it as a Boolean value into a byte. Position counting starts with 0 and runs from least to most significant bit. The bit operation signature is

```
bit: mddExp, intExp -> mddExp
```

In C/C++ style, `bit(mdd, pos)` is equivalent to `mdd >> pos & 1`.

Arithmetic, trigonometric, and exponential functions The following advanced arithmetic functions are available with the obvious meaning, each of them accepting an MDD object:

```
abs()
sqrt()
exp() log() ln()
sin() cos() tan()
sinh() cosh() tanh()
arcsin() arccos() arctan()
```

Exceptions

Generally, on domain error or other invalid cell values these functions will not throw an error, but result in NaN or similar according to IEEE floating-point arithmetic. Internally the rasdaman implementation calls the corresponding C++ functions, so the C++ documentation applies.

cast Sometimes the desired ultimate scalar type or MDD cell type is different from what the MDD expression would suggest. To this end, the result type can be enforced explicitly through the cast operator.

The syntax is:

```
(newType) generalExp
```

where newType is the desired result type of expression generalExp.

Like in programming languages, the cast operator converts the result to the desired type if this is possible at all. For example, the following scalar expression, without cast, would return a double precision float value; the cast makes it a single precision value:

```
(float) avg_cells( mr )
```

Both scalar values and MDD can be cast; in the latter case, the cast operator is applied to each cell of the MDD yielding an array over the indicated type.

The cast operator also works properly on composite cell structures. In such a case, the cast type is applied to every component of the cell. For example, the following expression converts the pixel type of an (3x8 bit) RGB image to an image where each cell is a structure with three long components:

```
(long) rgb
```

Obviously in the result structure all components will bear the same type. In addition, the target type can be a user-defined composite type, e.g. the following will cast the operand to *{1c, 2c, 3c}*:

(RGBPixel) {1c, 2l, 3.0}

Casting from larger to smaller integer type

If the new type is smaller than the value's type, i.e. not all values can be represented by it, then standard C++ casting will typically lead to strange results due to wrap around for unsigned and implementation-defined behavior for a signed types. For example, casting int 1234 to char in C++ will result in 210, while the possible range would be 0 - 255.

Rasdaman implements a more reasonable cast behavior in this case: if the value is larger than the maximum value representable by the new type, then the result is the maximum value (e.g. 255 in the previous example); analogously, if the value is smaller than the minimum possible value, then the result is the minimum value.

This is implemented only on integer types and entails a small performance penalty in comparison to raw C++ as up to two comparisons per cell (with the maximum and minimum) are necessary when casting.

Restrictions

On base type complex, only the following operations are available right now:

```
+ - * /
```

4.10.5 Scaling

Shorthand functions are available to scale multidimensional objects. They receive an array as parameter, plus a scale indicator. In the most common case, the scaling factor is an integer or float number. This factor then is applied to all dimensions homogeneously. For a scaling with individual factors for each dimension, a scaling vector can be supplied which, for each dimension, contains the resp. scale factor. Alternatively, a target domain can be specified to which the object gets scaled.

Syntax

```
scale( mddExp, intExp )
scale( mddExp, floatExp )
scale( mddExp, intVector )
scale( mddExp, mintervalExp )
```

Examples

The following example returns all images of collection `mr` where each image has been scaled down by a factor of 2.

```
select scale( mr, 0.5 )
from mr
```

Next, mr images are enlarged by 4 in the first dimension and 3 in the second dimension:

```
select scale( mr, [ 4, 3 ] )
from mr
```

In the final example, mr images are scaled to obtain 100x100 thumbnails (note that this can break aspect ratio):

```
select scale( mr, [ 0:99, 0:99 ] )
from mr
```

Note: Function `scale()` breaks tile streaming, it needs to load all tiles affected into server main memory. In other words, the source argument of the function must fit into server main memory. Consequently, it is not advisable to use this function on very large items.

Note: Currently only nearest neighbour interpolation is supported for scaling.

4.10.6 Concatenation

Concatenation of two arrays “glues” together arrays by lining them up along an axis.

This can be achieved with a shorthand function, `concat`, which for convenience is implemented as an n-ary operator accepting an unlimited number of arrays of the same base type. The operator takes the input arrays, lines them up along the concatenation dimension specified in the request, and outputs one result array. To this end, each input array is shifted to the appropriate position, with the first array’s position remaining unchanged; therefore, it is irrelevant whether array extents, along the concatenation dimension, are disjoint, overlapping, or containing each other.

The resulting array’s dimensionality is equal to the input array dimensionality.

The resulting array extent is the sum of all extents along the concatenation dimension, and the extent of the input arrays in all other dimensions.

The resulting array cell type is same as the cell types of the input arrays.

Constraints

All participating arrays must have the same number of dimensions.

All participating arrays must have identical extents in all dimensions, except that dimension along which concatenation is performed.

Input arrays must have the same cell types, i.e. concatenating a char and float arrays is not possible and requires explicit casting to a common type.

Syntax

```
concat mddExp with mddExp ... with mddExp along integer
```

Examples

The following query returns the concatenation of all images of collection mr with themselves along the first dimension (Figure 4.11).

```
select concat mr with mr along 0
from mr
```

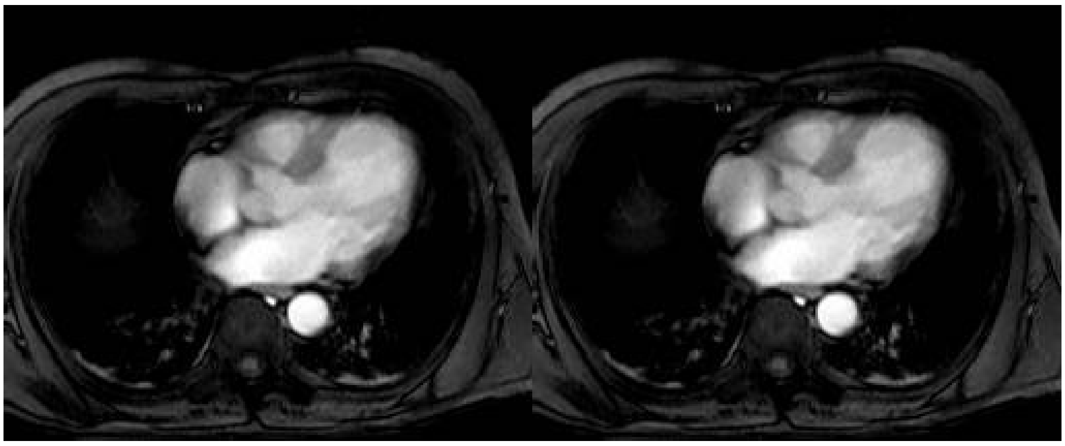


Figure 4.11: Query result of single concatenation

The next example returns a 2x2 arrangement of images (Figure 4.12):

```
select concat (concat mr with mr along 0)
with (concat mr with mr along 0)
along 1
from mr
```

4.10.7 Condensers

Frequently summary information of some kind is required about some array, such as sum or average of cell values. To accomplish this, rasql provides the concept of condensers.

A condense operation (or short: condenser) takes an array and summarizes its values using a summarization function, either to a scalar value (e.g. computing the sum of all its cells), or to another array (e.g. summarizing a 3-D cube into a 2-D image by adding all the horizontal slices that the cube is composed of).

A number of condensers is provided as rasql built-in functions.

- For *numeric* arrays, `add_cells()` delivers the sum and `avg_cells()` the average of all cell values. Operators `min_cells()` and `max_cells()` return the minimum and maximum, resp., of all cell values in the argument array. `stddev_pop`, `stddev_samp`, `var_pop`, and `var_samp` allow to calculate the population and sample standard deviation, as well as the population and sample variance of the MDD cells.

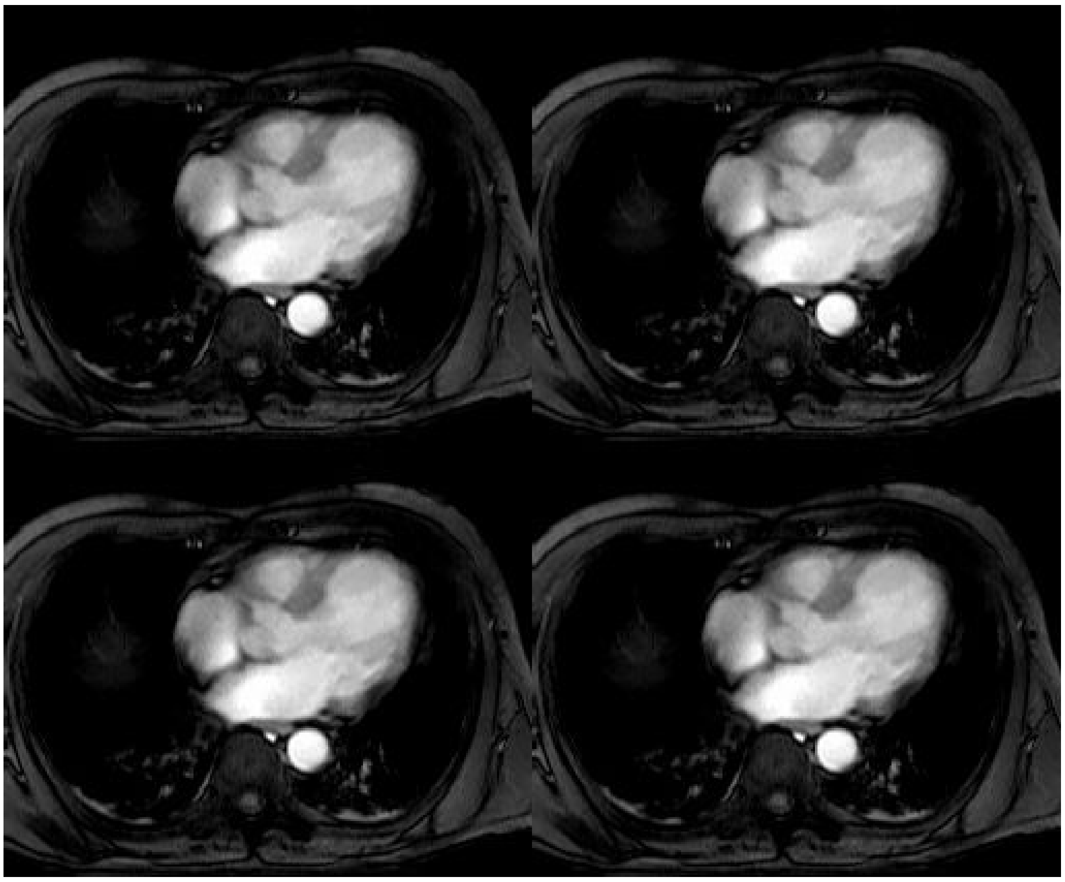


Figure 4.12: Query result of multiple concatenation

- For *boolean* arrays, the condenser `count_cells()` counts the cells containing `true`; `some_cells()` operation returns true if at least one cell of the boolean MDD is `true`, `all_cells()` returns true if all of the MDD cells contain `true` as value.

Please keep in mind that, depending on their nature, operations take a boolean, numeric, or arbitrary *mddExp* as argument.

Syntax

```
count_cells( mddExp )
add_cells( mddExp )
avg_cells( mddExp )
min_cells( mddExp )
max_cells( mddExp )
some_cells( mddExp )
all_cells( mddExp )
stddev_pop( mddExp )
stddev_samp( mddExp )
var_pop( mddExp )
var_samp( mddExp )
```

Examples

The following example returns all images of collection `mr` where all pixel values are greater than 20. Note that the induction "`>20`" generates a boolean array which, then, can be collapsed into a single boolean value by the condenser.

```
select mr
from mr
where all_cells( mr > 20 )
```

The next example selects all images of collection `mr` with at least one pixel value greater than 250 in region `[ 120:160, 55:75]` (Figure 4.13).

```
select mr
from mr
where some_cells( mr[120 : 160, 55 : 75] > 250 )
```

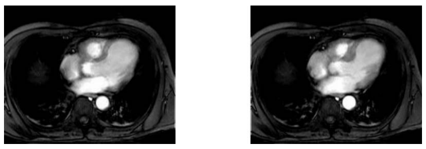


Figure 4.13: Query result of specific selection

Finally, this query calculates the sample variance of `mr2`:

select var_samp(mr2) from mr2

4.10.8 SORT operator

The SORT operator allows to sort the slices along a given axis of an array. This is done by calculating a rank value for each of the slices according to a given *ranking function*, and then reordering the slices according to their ranks in ascending (by default) or descending order.

Syntax

```
    SORT generalExp
ALONG sortAxis AS sortAxisIterator [listingOrder]
BY cellExp
```

Where

```
sortAxis: integerLit | identifier
sortAxisIterator: identifier
listingOrder: ASC | DESC
```

The generalExp denotes the array to be sorted; arrays of any dimensionality and type can be specified, or expressions that produce an array.

The sortAxis along which the array is sliced and reordered is specified in the ALONG clause. It can be specified by axis name according to the MDD type definition (e.g. x, y, Lat, . . .), or by an integer indicating its 0-based order (0 for the first axis, 1 for the second, and so on). Additionally a sortAxisIterator must be specified as an alias for addressing the axis in the ranking function, e.g. in subsetting the generalExp into slices and aggregating each into a rank. Depending on the optional listingOrder the slices are sorted in ascending ASC (default if not specified) or descending DESC order.

The cellExp in the BY clause is the slice ranking function. It must result in an atomic scalar value for each point in the sort axis extent. Slices for which the same rank is calculated retain their relative order as in the input array (stable sorting). The sortAxisIterator can be used to reference the points along the sortAxis.

The mechanics of the SORT expressions is perhaps more clearly explained via an equivalence to an MARRAY constructor expression, which creates a 1D array of ranks calculated by the cellExp for each point in the 1D domain created by the extend of the sortAxis; then SORT sorts these ranks and the slices to which they correspond in ascending or descending order. For example, if A is a 3D array and we have the a SORT expression that reorders the slices along the first axis by their average values:

```
SORT A
ALONG 0 AS sortAxis
BY avg_cells( A[ sortAxis[0], *:*, *:* ] )
```

then before the sorting is applied, first the ranks for each slice are calculated with an MARRAY:

```
MARRAY sortAxis in [ sdom(A)[0].lo : sdom(A)[0].hi ]
VALUES avg_cells( A[ sortAxis[0], *:*, *:* ] )
```

The sorting causes no change in the spatial domain, base type, or dimensionality in the result.

Examples

The following examples illustrate the syntax of the `SORT` operator; `raster2D` and `raster3D` are 2D and 3D MDDs with axes x/y and t/x/y respectively.

```
SORT raster2D ALONG 0 AS i
BY raster2D[i[0], 1]

SORT raster2D ALONG 0 AS i
BY add_cells(raster2D[i[0], *:*])

SORT raster2D ALONG x AS i
BY add_cells(raster2D[i[0], *:*])

SORT raster2D ALONG 0 AS n DESC
BY add_cells(raster2D[n[0], 0:100])

SORT raster2D ALONG 0 AS i
BY add_cells(raster2D[i[0], *:*]) - raster2D[i[0], 0]

SORT raster3D ALONG t AS time DESC
BY max_cells(raster3D[time[0], *:*, *:*])
```

The following examples show the semantics of the sort operator. Array cells which contribute to the rank result are highlighted in red.

The following example shows how a 10x3 array of double floating-point values is sorted along its second axis in an ascending order by the minimum of all values in each slice.

```
    SORT raster2D ALONG 1 AS i
BY min_cells(raster2D[*:*, i[0]])
```

The array looks as follows before and after sorting:

$$\begin{bmatrix} 14 & 95.33 & 72.74 \\ 42 & 17.38 & 41.91 \\ 33 & 76.16 & 97.27 \\ 84 & 7.72 & 19.31 \\ 75 & 43.66 & 22.02 \\ 96 & 66.47 & 39.14 \\ 757 & \color{red}{3.26} & 39.05 \\ \color{red}{8} & 50.29 & \color{red}{14.8} \\ 97 & 9.73 & 27.53 \\ 10 & 93.5 & 42.86 \end{bmatrix} \quad \begin{bmatrix} 95.33 & 14 & 72.74 \\ 17.38 & 42 & 41.91 \\ 76.16 & 33 & 97.27 \\ 7.72 & 84 & 19.31 \\ 43.66 & 75 & 22.02 \\ 66.47 & 96 & 39.14 \\ \color{red}{3.26} & 757 & 39.05 \\ 50.29 & \color{red}{8} & \color{red}{14.8} \\ 9.73 & 97 & 27.53 \\ 93.5 & 10 & 42.86 \end{bmatrix}$$

Figure 4.14: Minimum values in columns 0,1,2 are 8, 3.26, and 14.8 respectively, highlighted in red.

We observe that the sorting was a reordering of the slices along the second axis, represented by the columns.

By subsetting along an axis other than the `sortAxis`, we can further restrict areas in the slices which contribute to the ranking function. For example, the following 2 queries consider only the value at index positions 0 or 1, respectively, along axis 1.

```
      SORT raster2D ALONG 0 AS i
BY (raster2D2[i[0], 0])

      SORT raster2D ALONG 0 AS i
BY (raster2D2[i[0], 1])
```

$$\begin{bmatrix} 13 & 95.33 & 72.74 \\ 32 & 17.38 & 41.91 \\ 3 & 76.16 & 97.27 \\ 44 & 7.72 & 19.31 \\ 5.7 & 43.66 & 22.02 \\ 6 & 66.47 & 39.14 \\ 77.1 & 3.26 & 39.05 \\ 58.8 & 50.29 & 14.8 \\ 97 & 9.73 & 27.53 \\ 5.9 & 93.5 & 42.86 \end{bmatrix} \quad \begin{bmatrix} 3 & 76.16 & 97.27 \\ 5.7 & 43.66 & 22.02 \\ 5.9 & 93.5 & 42.86 \\ 6 & 66.47 & 39.14 \\ 13 & 95.33 & 72.74 \\ 32 & 17.38 & 41.91 \\ 44 & 7.72 & 19.31 \\ 58.8 & 50.29 & 14.8 \\ 77.1 & 3.26 & 39.05 \\ 97 & 9.73 & 27.53 \end{bmatrix} \quad \begin{bmatrix} 77.1 & 3.26 & 39.05 \\ 44 & 7.72 & 19.31 \\ 97 & 9.73 & 27.53 \\ 32 & 17.38 & 41.91 \\ 5.7 & 43.66 & 22.02 \\ 58.8 & 50.29 & 14.8 \\ 6 & 66.47 & 39.14 \\ 3 & 76.16 & 97.27 \\ 5.9 & 93.5 & 42.86 \\ 13 & 95.33 & 72.74 \end{bmatrix}$$

Figure 4.15: Sorting an array with further subsetting each slice along the sort axis.

You might also want to compare two values in a specific axis at once, at each slice, and sort by the minimum value between those, using an aggregate operation:

```
      SORT raster2D ALONG 0 AS i
BY min_cells(raster2D2[i[0], 1:2])
```

$$\begin{bmatrix} 77.1 & 3.26 & 39.05 \\ 44 & 7.72 & 19.31 \\ 97 & 9.73 & 27.53 \\ 58.8 & 50.29 & 14.8 \\ 32 & 17.38 & 41.91 \\ 5.7 & 43.66 & 22.02 \\ 6 & 66.47 & 39.14 \\ 5.9 & 93.5 & 42.86 \\ 13 & 95.33 & 72.74 \\ 3 & 76.16 & 97.27 \end{bmatrix}$$

Figure 4.16: Sorting an array by the minimum value with further subsetting at a specific axis.

The next example, shows how a 3D array of double floating-point values for temperature is sorted along its first axis of time in a descending order, by the maximum temperature value in each latitude/longitude combination:

```
      SORT weather ALONG 0 AS i DESC
BY max_cells(weather[i[0], *:*, *:*])
```

The datasheet looks as follows:

Date	*lat*	*lon*	*temperature*
2008.11.13 – 02 : 05 : 00	30.4367	−88.0117	18.05
2008.11.13 – 02 : 06 : 00	27.601	−82.751	20.52
2008.11.13 – 02 : 06 : 59	30.4367	−88.0117	17.96
2008.11.13 – 06 : 07 : 00	30.4367	−88.0117	18.08
2008.11.13 – 06 : 10 : 00	30.4367	−88.0117	17.84
2008.11.13 – 17 : 42 : 00	27.601	−82.751	21.35
2008.11.13 – 17 : 48 : 00	27.601	−82.751	21.68
2008.11.13 – 22 : 54 : 00	27.601	−82.751	21.41
2008.11.13 – 23 : 00 : 00	27.601	−82.751	21.4
2008.11.13 – 23 : 00 : 00	25.084	−81.096	24.51
2008.11.14 – 05 : 12 : 00	30.4367	−88.0117	18.61
2008.11.14 – 05 : 12 : 00	27.601	−82.751	21.22
2008.11.14 – 19 : 18 : 00	27.601	−82.751	21.98
2008.11.14 – 19 : 18 : 00	25.084	−81.096	24.87

Dim.0 : Date	*Dim.1 : lat*	*Dim.2 : lon*	*cellvalue : temperature*
0	2	2	18.05
1	1	1	20.52
2	2	2	17.96
3	2	2	18.08
4	2	2	17.84
5	1	1	21.35
6	1	1	21.68
7	1	1	21.41
8	1	1	21.4
8	0	0	24.51
9	2	2	18.61
9	1	1	21.22
10	1	1	21.98
10	0	0	24.87

The first datasheet represents all the data that we have. The second offers a datacube interpretation of the available data. The first dimension is time, the second the latitude, and the third represents the longitude. In this example, the lat/lon are limited to 3 entries each. And the temperature is recorded at 11 unique timestamps. Thus, the data is represented using the available combinations of longitude and latitude (25.084,-81.096),(27.601,-82.751) and (30.4367,-88.0117), by (0,0), (1,1) and (2,2) respectively, each in their timestamp. All the cell values are recorded temperatures.

The following illustration represents a 3D array holding this data:

```
[
[ [0,0,0],[0,0,0],[0,0,18.05] ] ,
[ [0,0,0],[0,20.52,0],[0,0,0] ] ,
[ [0,0,0],[0,0,0],[0,0,17.96] ] ,
[ [0,0,0],[0,0,0],[0,0,18.08] ] ,
[ [0,0,0],[0,0,0],[0,0,17.84] ] ,
[ [0,0,0],[0,21.35,0],[0,0,0] ] ,
[ [0,0,0],[0,21.68,0],[0,0,0] ] ,
[ [0,0,0],[0,21.41,0],[0,0,0] ] ,
[ [24.51,0,0],[0,21.4,0],[0,0,0] ] ,
[ [0,0,0],[0,21.22,0],[0,0,18.61] ] ,
[ [24.87,0,0],[0,21.98,0],[0,0,0] ]
]
```

On the first timestamp, for instance, we have a temperature measurement of 18.05 degrees, for lat/lon (2,2), which represents (30.4367,-88.0117) of the original table. In the last timestamp, we have two temperature records, at (0,0) and at (1,1). If we had more measurements, they would fill in the zero-values.

After sorting, the array looks as follows:

```
[
[ [24.87,0,0],[0,21.98,0],[0,0,0] ] ,
[ [24.51,0,0],[0,21.4,0],[0,0,0] ] ,
[ [0,0,0],[0,21.68,0],[0,0,0] ] ,
[ [0,0,0],[0,21.41,0],[0,0,0] ] ,
[ [0,0,0],[0,21.35,0],[0,0,0] ] ,
[ [0,0,0],[0,21.22,0],[0,0,18.61] ] ,
[ [0,0,0],[0,20.52,0],[0,0,0] ] ,
[ [0,0,0],[0,0,0],[0,0,18.08] ] ,
[ [0,0,0],[0,0,0],[0,0,18.05] ] ,
[ [0,0,0],[0,0,0],[0,0,17.96] ] ,
[ [0,0,0],[0,0,0],[0,0,17.84] ]
]
```

We observe that the time-slices have been sorted by their maximum value, in a descending manner.

One can also sort by a specific longitude/latitude combination, e.g:

```
      SORT weather ALONG 0 AS i
BY weather[i[0], 0, 0] DESC
```

4.10.9 FLIP operator

The `FLIP` operator allows to reverse the values/slices of an MDD along a particular axis. Similar to `SORT`, the array is sliced at the chosen axis. The slices are then reordered in opposite order, resulting in an array with *no* change in the spatial domain, base type, or dimensionality.

Syntax

```
FLIP generalExp ALONG flipAxis
```

Where

```
flipAxis: integerLit | identifier
```

The `generalExp` denotes the array argument; arrays of any dimensionality and type can be specified (or expressions that produce an array).

The `flipAxis` along which the array is sliced and reordered in reverse order is specified in the `ALONG` clause. It can be specified by axis name according to the MDD type definition (e.g. x, y, Lat, . . .), or by an integer indicating its 0-based order (0 for the first axis, 1 for the second, and so on).

Examples

The following examples illustrate the syntax of the `FLIP` operator; `raster2D` and `raster3D` are 2D and 3D MDDs with axes x/y and time/x/y respectively.

```
FLIP raster2D ALONG 0
FLIP raster2D ALONG x

FLIP raster3D ALONG time
```

The next example illustrates the inversion of the following array:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

Flipping the array on its first axis with

```
FLIP raster2D ALONG 0
```

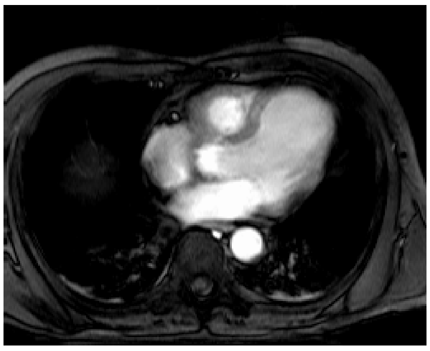
and flipping on the second axis with

```
FLIP raster2D ALONG 1
```

yields the following results, respectively:

$$\begin{bmatrix} 10 & 11 & 12 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \\ 12 & 11 & 10 \end{bmatrix}$$

In a more visual example, applying the FLIP operation on the sample MRT imagery collection `mr2` will mirror the image vertically or horizontally. The original image looks as follows:



flipping on the first axis with

```
FLIP mr2 ALONG 0
```

results in

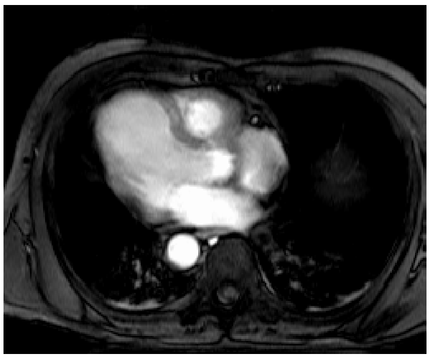


Figure 4.17: the image is mirrored on the vertical axis

and flipping on the second axis with

```
FLIP mr2 ALONG 1
```

results in

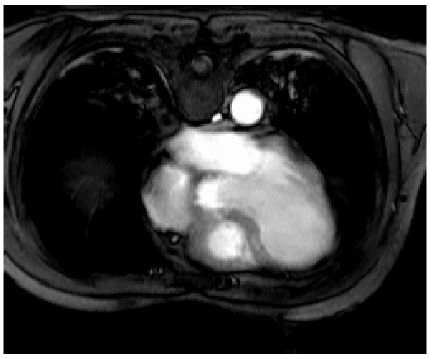


Figure 4.18: the image is mirrored on the horizontal axis

4.10.10 General Array Condenser

All the condensers introduced above are special cases of a general principle which is represented by the *general condenser* statement.

The general condense operation consolidates cell values of a multidimensional array to a scalar value based on the condensing operation indicated. It iterates over a spatial domain while combining the result values of the *cellExp*s through the *condenserOp* indicated.

The general condense operation consolidates cell values of a multidimensional array to a scalar value or an array, based on the condensing operation indicated.

Condensers are heavily used in two situations:

- To collapse boolean arrays into scalar boolean values so that they can be used in the **where** clause.
- In conjunction with the **marray** constructor (see next section) to phrase high-level signal processing and statistical operations.

Syntax

```
condense condenserOp
over var in mintervalExp
using cellExp

condense condenserOp
over var in mintervalExp
where booleanExp
using cellExp
```

The *mintervalExp* terms together span a multidimensional spatial domain over which the condenser iterates. It visits each point in this space exactly once, assigns the point's respective coordinates to the *var* variables and evaluates *cellExp* for the current point. The result values are combined using condensing function *condenserOp*. Optionally, points used for the aggregate can be filtered through a *booleanExp*; in this case, *cellExp* will be evaluated only for those points where *booleanExp* is true, all others will not be regarded. Both *booleanExp* and *cellExp* can contain occurrences of variables *pointVar*.

Examples

This expression below returns a scalar representing the sum of all array values, multiplied by 2 (effectively, this is equivalent to add_cells(2*a)):

```
condense +
over x in sdom(a)
using 2 * a[ x ]
```

The following expression returns a 2-D array where cell values of 3-D array a are added up along the third axis:

```
condense +
over x in [0:100]
using a[ *:*, *:*, x[0] ]
```

Note that the addition is induced as the result type of the value clause is an array. This type of operation is frequent, for example, in satellite image time series analysis where aggregation is performed along the time axis.

Shorthands

Definition of the specialized condensers in terms of the general condenser statement is as shown in Table 4.5.

Table 4.5: Specialized condensers; a is a numeric, b a boolean array.

Aggregation definition	Meaning
`add_cells(a) = condense + over x in sdom(a) using a[x]`	sum over all cells in a
`avg_cells(a) = sum_cells(a) / card(sdom(a))`	Average of all cells in a
`min_cells(a) = condense min over x in sdom(a) using a [x]`	Minimum of all cells in a
`max_cells(a) = condense max over x in sdom(a) using a[x]`	Maximum of all cells in a
`count_cells(b) = condense + over x in sdom(b) where b[x] != 0 using 1`	Number of cells in b which are non-zero / not *false*
`some_cells(b) = condense or over x in sdom(b) using b[x]`	is there any cell in b with value *true*?
`all_cells(b) = condense and over x in sdom(b) using b[x]`	do all cells of b have value *true*?

Restriction

Currently condensers over complex numbers are generally not supported, with exception of `add_cells` and `avg_cells`.

4.10.11 General Array Constructor

The **marray** constructor allows to create n-dimensional arrays with their content defined by a general expression. This is useful

- whenever the array is too large to be described as a constant (see *Array Constants*) or
- when the array's contents is derived from some other source, e.g., for a histogram computation (see examples below).

Syntax

The basic shape of the `marray` constructor is as follows:

```
marray var in mintervalExp [, var in mintervalExp]
values cellExp
```

The *cellExp* describes how the resulting array is produced at each point of its domain.

Iterator Variable Declaration

The result array is defined by the cross product of all *mintervalExp*. For example, the following defines a 2-D 5x10 matrix:

```
marray x in [1:5], y in [1:10]
values ...
```

The base type of the array is determined by the type of *cellExp*. Each variable *var* can be of any number of dimensions.

Iteration Expression

The resulting array is filled in at each coordinate of its spatial domain by successively evaluating *cellExp*; the result value is assigned to the cell at the coordinate currently under evaluation. To this end, *cellExp* can contain arbitrary occurrences of *var*, which are accordingly substituted with the values of the current coordinate. The syntax for using a variable is:

- for a one-dimensional variable:

  ```
  var
  ```

- for a one- or higher-dimensional variable:

  ```
  var [ index-expr ]
  ```

where *index-expr* is a constant expression evaluating to a non-negative integer; this number indicates the variable dimension to be used.

Examples

The following creates an array with spatial domain [1:100,-50:200] over cell type char, each cell being initialized to 1.

```
marray x in [ 1:100, -50:200 ]
values 1c
```

Figure 4.19: 2-D array with values derived from first coordinate

In the next expression, cell values are dependent on the first coordinate component (cf. Figure 4.19):

```
marray x in [ 0:255, 0:100 ]
values x[0]
```

The final two examples comprise a typical marray/condenser combination. The first one takes a sales table and consolidates it from days to week per product. Table structure is as given in Figure 4.20.:

```
select marray tab in [ 0:sdom(s)[0].hi/7, sdom(s)[1] ]
       values condense +
              over day in [ 0:6 ]
              using s[ day[0] + tab7 ] , tab[1] ]
from salestable as s
```

The last example computes histograms for the mr images. The query creates a 1-D array ranging from 0 to 9 where each cell contains the number of pixels in the image having the respective intensity value.

```
select marray v in [ 0 : 9 ]
       values condense +
              over x in sdom(mr)
              where mr[x] = v[0]
              using 1
from mr
```

Shorthand

As a shorthand, variable *var* can be used without indexing; this is equivalent to var[0]:

```
marray x in [1:5]
values a[ x ]        -- equivalent to a[ x[0] ]
```

Known issue: the shorthand notation currently works as expected only when one variable is defined.

Many vs. One Variable

Obviously an expression containing several 1-D variables, such as:

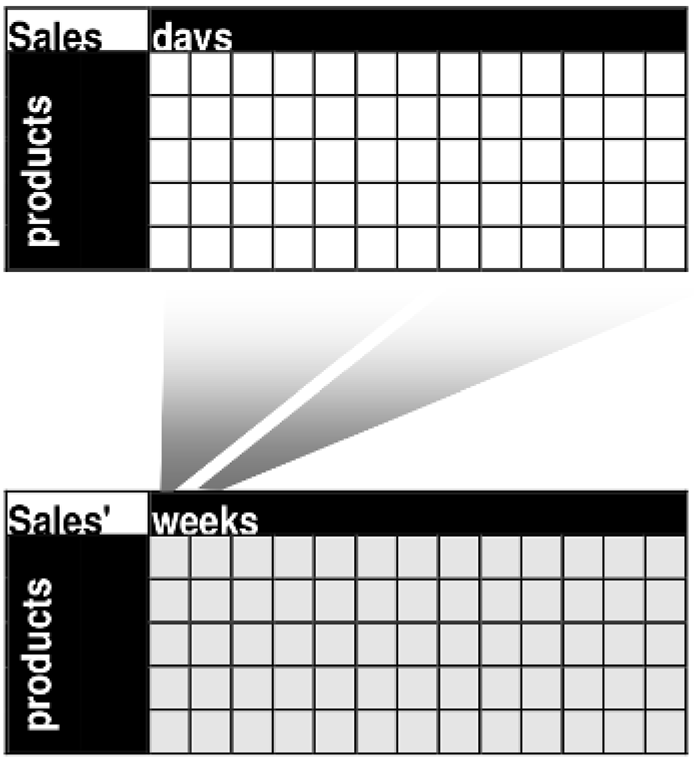


Figure 4.20: Sales table consolidation

```
marray x in [1:5], y in [1:10]
values a[ x[0], y[0] ]
```

can always be rewritten to an equivalent expression using one higher-dimensional variable, for example:

```
marray xy in [1:5, 1:10]
values a[ xy[0], xy[1] ]
```

Iteration Sequence Undefined

The sequence in which the array cells defined by an marray construct are inspected is not defined. In fact, server optimisation will heavily make use of reordering traversal sequence to achieve best performance.

Restriction

Currently there is a restriction in variable lists: for each marray variable declaration, either there is only one variable which can be multidimensional, or there is a list of one-dimensional variables; mixing the two is not allowed.

A Note on Expressiveness and Performance

The general condenser and the array constructor together allow expressing a very broad range of signal processing and statistical operations. In fact, all other rasql array operations can be expressed through them, as Table 4.6 exemplifies. Nevertheless, it is advisable to use the specialized operations whenever possible; not only are they more handy and easier to read, but also internally their processing has been optimized so that they execute considerably faster than the general phrasing.

Table 4.6: Phrasing of Induction, Trimming, and Section via marray

operation	shorthand	phrasing with marray
Trimming	`a[ *:*, 50:100 ]`	`marray x in [sdom(a)[0], 50:100] values a[ x ]`
Section	`a[ 50, *:* ]`	`marray x in sdom(a)[1] values a[ 50, x ]`
Induction	`a + b`	`marray x in sdom(a) values a[x] + b[x]`

4.10.12 Type Coercion Rules

This section specifies the type coercion rules in query expressions, i.e. how the base type of the result from an operation applied on operands of various base types is derived.

The *guiding design principle* for these rules is to minimize the risk for overflow, but also "type inflation": when a smaller result type is sufficient to represent all possible values of an operation, then it is preferred over a larger result type. This is especially important in the context of rasdaman, where the difference between float and double for example can be multiple GBs or TBs for large arrays. As such, the rules are somewhat different from C++ for example or even numpy, where in general careful explicit casting is required to avoid overflow or overtyping.

Here a summary is presented, while full details can be explored in rasdaman's systemtest. The type specifiers (c, o, s, . . .) are the literal type suffixes as documented in Table 4.2; `X` and `Y` indicate any cell type, `U` corresponds to any unsigned integer type, `S` to any signed integer type, `C` to any complex type. In every table the upper rows have precedence, i.e. the deduction rules are ordered; if a particular operand type combination is missing it means that it is not supported and would lead to a type error. The first/second operand types are commutative by default and only one direction is shown to reduce clutter. Types have a *rank* determined by their size in bytes and signedness, so that double has a higher rank than float, and long has higher rank than ulong; max/min of two types returns the type with higher/lower type. Adding 1 to a type results in the next type by rank, preserving signedness; the integer/floating-point boundary is not crossed, however, i.e. `long + 1 = long`.

Binary Induced

Complex operands are only supported by `+, -, \*, /, div, =, and !=`. If any operand of these operations is complex, then the result is complex with underlying type derived by applying the rules to the underlying types of the inputs. E.g. `char + CInt16 = char + short = CInt32`, and `CInt32 * CFloat32 = long * float = CFloat64`.

+, *, div, mod

first	second	result
X	d	d
l,ul	f	d
X	f	f
U1	U2	max(U1, U2) + 1
X	Y	signed(max(X, Y) + 1)

- (subtraction)

The result can always be negative, even if inputs are unsigned (positive), so for integers the result type is always the next greater signed type. Otherwise, the rules are the same as for +, *, div, mod.

first	second	result
X	d	d
l,ul	f	d
X	f	f
X	Y	signed(max(X, Y) + 1)

/ (division)

Division returns floating-point to avoid inadvertent precision loss as well as unnecessary check for division by zero. Integer division is supported with the `div` function.

first	second	result
c,o,s,us,f	c,o,s,us,f	f
X	Y	d

pow, power

Note: operand types are not commutative, the second operand must be a float or double scalar.

first	second	result
c,o,s,us,f	f, d	f
ul,l,d	f, d	d

<, >, <=, >=, =, !=

first	second	result
X	Y	bool

min, max, overlay

first	second	result
X	X	X

and, or, xor, is

first	second	result
bool	bool	bool

bit

`I` stands for any signed and unsigned integer type.

first	second	result
I	I	bool

complex(re, im)

first (re)	second (im)	result
s	s	CInt16
l	l	CInt32
f	f	CFloat32
d	d	CFloat64

Unary Induced

not

op	result
bool	bool

abs

op	result
C	error
X	X

sqrt, log, ln, exp, sin, cos, tan, sinh, cosh, tanh, arcsin, arccos, arctan

op	result
c,o,us,s,f	f
u,l,d	d

Condensers

count_cells

op	result
bool	ul

add_cells and **condense +, ***

op	result
C	CFloat64
f,d	d
S	l
U	ul

avg_cells

op	result
C	CFloat64
X	d

stddev_pop, stddev_samp, var_pop, var_samp

op	result
C	error
X	d

min_cells, max_cells and **condense min, max**

op	result
C	error
X	X

some_cells, all_cells and **condense and, or**

op	result
bool	bool

Geometric Operations

The base type does not change in the result of subset, shift, extend, scale, clip, concat, and geographic reprojection.

op	result
X	X

4.11 Data Format Conversion

Without further indication, arrays are accepted and delivered in the client's main memory format, regardless of the client and server architecture. Sometimes, however, it is desirable to use some data exchange format - be it because some device provides a data stream to be inserted in to the database in a particular format, or be it a Web application where particular output formats have to be used to conform with the respective standards.

To this end, rasql provides two functions for

- decoding format-encoded data into an MDD
- encoding an MDD to a particular format

Implementation of these functions is based on GDAL and, hence, supports all GDAL formats. Some formats are implemented natively in addition: NetCDF, GRIB, JSON, and CSV.

4.11.1 Decode for data import

The `decode()` function allows for decoding data represented in one of the supported formats, into an MDD which can be persisted or processed in rasdaman.

Syntax

```
decode( mddExp )

encode( mddExp , format , formatParameters )
```

As a first paramater the data to be decoded must be specified. Technically this data must be in the form of a 1D char array. Usually it is specified as a query input parameter with `$1`, while the binary data is attached with the `--file` option of the rasql command-line client tool, or with the corresponding methods in the client API. If the data is on the same machine as rasdaman, it can be loaded directly by specifying the path to it in the format parameters; more details on this in *Format parameters*.

Data format

The source data format is automatically detected in case it is handled by GDAL (e.g. PNG, TIFF, JPEG, etc; see output of `gdalinfo --formats` or the GDAL documentation for a full list), so there is no format parameter in this case.

A format is necessary, however, when a custom internal implementation should be selected instead of GDAL for decoding the data, e.g. NetCDF (`"netcdf"` / `"application/netcdf"`), GRIB (`"grib"`), JSON (`"json"` / `"application/json"`), or CSV (`"csv"` / `"text/csv"`).

Format parameters

Optionally, a format parameters string can be specified as a third parameter, which allows to control the format decoding. For GDAL formats it is necessary to specify format `"GDAL"` in this case.

The format parameters must be formatted as a valid JSON object. As the format parameters are in quotes, i.e. `"formatParameters"`, all quotes inside of the `formatParameters` need to be escaped (`\"`). For example, `"{ \"transpose\": [0,1] }"` is the right way to specify transposition, while `"{ "transpose": [0,1] }"` will lead to failure. Note that in examples further on quotes are not escaped for readability.

Common parameters

The following parameters are common to GDAL, NetCDF, and GRIB data formats:

- `variables` - An array of variable names or band ids (0-based, as strings) to be extracted from the data. This allows to decode only some of the variables in a NetCDF file for example with `["var1", "var2"]`, or the bands of a TIFF file with `["0", "2"]`.
- `filePaths` - An array of absolute paths to input files to be decoded, e.g. `["/path/to/rgb.tif"]`. This improves ingestion performance if the data is on the same machine as the rasdaman server, as the network transport is bypassed and the data is read directly from disk. Supported only for GDAL, NetCDF, and GRIB data formats.
- `subsetDomain` - Specify a subset to be extracted from the input file, instead of the full data. The subset should be specified in rasdaman minterval format as a string, e.g. `"[0:100,0:100]"`. Note that the subset domain must match in dimensionality with the file dimensionality, and must be accordingly offset to the grid origin in the file, which is typically [0,0,0,. . .].
- `transpose` - Specify if x/y should be transposed with an array of 0-based axis ids indicating the axes that need to be transposed; the axes must be contiguous `[N,N+1]`, e.g. `[0,1]`. This is often relevant in NetCDF and GRIB data which have a swapped x/y order than what is usually expected in e.g. GDAL. Note that transposing axes has a performance penalty, so avoid if possible.
- `formatParameters` - A JSON object containing extra options which are format-specific, specified as string key-value pairs. This is where one would specify the base

type and domain for decoding a CSV file for example, or GDAL format-specific options. Example for a CSV file:

```
"formatParameters": {
  "basetype": "struct { float f, long l }",
  "domain": "[0:100,0:100]"
}
```

GDAL

- `formatParameters` - any entries in the formatParameters object are forwarded to the specific GDAL driver; consult the GDAL documentation for the options recognized by each particular driver. E.g. for PNG you could specify, among other details, a description metadata field with:

  ```
  "formatParameters": {
    "DESCRIPTION": "Data description..."
  }
  ```

- `configOptions` - A JSON object containing *configuration options* as string key-value pairs; more details in the GDAL documentation. Example:

  ```
  "configOptions": {
    "GDAL_CACHEMAX": "64",
    ...
  }
  ```

- `openOptions` - A JSON object containing *open options* as string key-value pairs; an option for selecting overview level from the file with, e.g. `"OVERVIEW_LEVEL": "2"`, is available for all formats (more details); further options may be supported by each driver, e.g. for TIFF;

  ```
  "openOptions": {
    "OVERVIEW_LEVEL": "2",
    "NUM_THREADS": "ALL_CPUS"
  }
  ```

Note: This feature is only available since GDAL 2.0, so if you have an older GDAL these options will be ignored.

GRIB

- `internalStructure` - Describe the internal structure of a GRIB file, namely the domains of all messages to be extracted from the file:

```
"internalStructure": {
  "messageDomains": [
    { "msgId": 1, "domain": "[0:0,0:0,0:719,0:360]" },
    { "msgId": 2, "domain": "[0:0,1:1,0:719,0:360]" },
    { "msgId": 3, "domain": "[0:0,2:2,0:719,0:360]" },
    ...
  ]
}
```

CSV / JSON

The following are mandatory options that have to be specified in the `formatParameters` object:

- `domain` - The domain of the MDD encoded in the CSV data. It has to match the number of cells read from input file, e.g. for `"domain": "[1:5, 0:10, 2:3]"`, there should be 110 numbers in the input file.
- `basetype` - Atomic or struct base type of the cell values in the CSV data; named structs like `RGBPixel` are not supported. Examples:

```
long
char
struct { char red, char blue, char green }
```

Numbers from the input file are read in order of appearance and stored without any reordering in rasdaman; whitespace plus the following characters are ignored:

```
'{', '}', ',', '"', '\'', '(', ')', '[', ']'
```

Examples

GDAL

The following query loads a TIFF image into collection `rgb`:

```
rasql -q 'insert into rgb values decode( $1 )' --file rgb.tif
```

If you use double quotes for the query string, note that the `$` must be escaped to avoid interpretation by the shell:

```
rasql -q "insert into rgb values decode( \$1 )" --file rgb.tif
```

The example below shows directly specifying a file path in the format parameters; `<[0:0] 1c>` is a dummy array value which is not relevant in this case, but is nevertheless mandatory:

```
UPDATE test_mr SET test_mr[0:255,0:210]
ASSIGN decode(<[0:0] 1c>, "GDAL",
    "{ \"filePaths\": [\"/home/rasdaman/mr_1.png\"] }")
WHERE oid(test_mr) = 6145
```

CSV / JSON

Let array `A` be a 2x3 array of longs given as a string as follows:

```
1,2,3,2,1,3
```

Inserting `A` into rasdaman can be done with

```
insert into A
values decode($1, "csv", "{ \"formatParameters\": {
      \"domain\": \"[0:1,0:2]\",
      \"basetype\": \"long\" } }")
```

Further, let `B` be an 1x2 array of RGB values given as follows:

```
{1,2,3},{2,1,3}
```

Inserting `B` into rasdaman can be done by passing it to this query:

```
insert into B
values decode($1, "csv", "{ \"formatParameters\": {
      \"domain\": \"[0:0,0:1]",
      \"basetype\": \"struct{char red, char blue, char green}\" } }
↪")
```

`B` could just as well be formatted like this with the same effect (note the line break):

```
1 2 3
2 1 3
```

4.11.2 Encode for data export

The `encode()` function allows encoding an MDD in a particular data format representation; formally, the result will be a 1D char array.

Syntax

```
encode( mddExp , format )

encode( mddExp , format , formatParameters )
```

The first parameter is the MDD to be encoded. It must be 2D if encoded to GDAL formats (PNG, TIFF, JPEG, etc.), while the native rasdaman encoders (NetCDF, JSON, and CSV) support MDDs of any dimension; note that presently encode to GRIB is not supported. As not all base types supported by rasdaman (char, octet, float, etc.) are necessarily supported by each format, care must be taken to cast the MDD beforehand.

Data format

A mandatory `format` must be specified as the second parameter, indicating the data format to which the MDD will be encoded; allowed values are

- GDAL format identifiers (see output of `gdalinfo --formats` or the GDAL documentation);
- a mime-type string, e.g. `"image/png"`;
- `"netcdf"` / `"application/netcdf"`, `"csv"` / `"text/csv"`, or `"json"` / `"application/json"`, for formats natively supported by rasdaman.

Format parameters

Optionally, a format parameters string can be specified as a third parameter, which allows to control the format encoding. As in the case of decode(), it must be a valid JSON object. As the format parameters are in quotes, i.e. `"formatParameters"`, all quotes inside of the `formatParameters` need to be escaped (`\"`). For example, `"{ \"transpose\": [0,1] }"` is the right way to specify transposition, while `"{ "transpose": [0,1] }"` will lead to failure.

Common parameters to most or all formats include:

- `metadata` - A single string, or an object of string key-value pairs which are added as global metadata when encoding.
- `transpose` - Specify if x/y should be transposed with an array of 0-based axis ids indicating the axes that need to be transposed; the axes must be contiguous `[N,N+1]`, e.g. `[0,1]`. This is often relevant when encoding data with GDAL formats, which was originally imported from NetCDF and GRIB files. Note that transposing axes has a performance penalty, so avoid if possible.
- `nodata` - Specify nodata value(s). If a single number is specified it will be applicable to all bands (e.g. `0`), otherwise an array of numbers for each band can be provided (e.g. `[0,255,255]`). Special floating-point constants are supported (case-sensitive): NaN, NaNf, Infinity, -Infinity.

- `formatParameters` - A JSON object containing extra options which are format-specific, specified as string key-value pairs. This is where one would specify the options for controling what separators and values are used in CSV encoding for example, or GDAL format-specific options.

GDAL

- `formatParameters` - any entries in the formatParameters object are forwarded to the specific GDAL driver; consult the GDAL documentation for the options recognized by each particular driver. E.g. for PNG you could specify, among other details, a description metadata field with:

  ```
  "formatParameters": {
    "DESCRIPTION": "Data description..."
  }
  ```

  Rasdaman itself does not change the default values for these parameters, with the following *exceptions*:

  – `PNG` - the compression level when encoding to PNG (option `ZLEVEL`) will be set to `2` if the user does not specify it explicitly and the result array is not of type `boolean`. The default compression level of `6` does not offer considerable space savings on typical image results (e.g. around 10% lower file size for satellite image), while significantly increasing the time to encode, taking up to 3-5x longer.

- `configOptions` - A JSON object containing configuration options as string key-value pairs; only relevant for GDAL currently, more details in the GDAL documentation. Example:

  ```
  "configOptions": {
    "GDAL_CACHEMAX": "64", ...
  }
  ```

Geo-referencing

- `geoReference` - An object specifying geo-referencing information; either “bbox” or “GCPs” must be provided, along with the “crs”:

  – `crs` - Coordinate Reference System (CRS) in which the coordinates are expressed. Any of the CRS representations acceptable by GDAL can be used:

    ∗ Well known names, such as `"NAD27"`, `"NAD83"`, `"WGS84"` or `"WGS72"`

    ∗ `"EPSG:n"`, `"EPSGA:n"`

    ∗ PROJ.4 definitions

    ∗ OpenGIS Well Known Text

    ∗ ESRI Well Known Text, prefixed with `"ESRI::"`

    * Spatial References from URLs
    * "AUTO:proj_id,unit_id,lon0,lat0" indicating OGC WMS auto projections
    * "urn:ogc:def:crs:EPSG::n" indicating OGC URNs (deprecated by OGC)

  - bbox - A geographic X/Y bounding box as an object listing the coordinate values (as floating-point numbers) for xmin, ymin, xmax, and ymax properties, e.g.:

```
"bbox": {
  "xmin": 0.0,
  "ymin": -1.0,
  "xmax": 1.0,
  "ymax": 2.0
}
```

  - GCPs - Alternative to a bbox, an array of GCPs (Ground Control Points) can be specified; see GCPs section in the GDAL documentation for details. Each element of the array is an object describing one control point with the following properties:
    * id - optional unique identifier (gets the GCP array index by default);
    * info - optional text associated with the GCP;
    * pixel, line - location on the array grid;
    * x, y, z - georeferenced location with coordinates in the specified CRS; "z" is optional (zero by default);

Coloring Arrays

- colorMap - Map single-band cell values into 1, 3, or 4-band values. It can be done in different ways depending on the specified type:
  - values - Each pixel is replaced by the entry in the colorTable where the key is the pixel value. In the example below, it means that all pixels with value -1 are replaced by [255, 255, 255, 0]. Pixels with values not present in the colorTable are not rendered: they are replaced with a color having all components set to 0.

```
"colorMap": {
  "type": "values",
  "colorTable": {
    "-1": [255, 255, 255, 0],
    "-0.5": [125, 125, 125, 255],
    "1": [0, 0, 0, 255]
  }
}
```

  - intervals - All pixels with values between two consecutive entries are rendered using the color of the first (lower-value) entry. Pixels with values equal to or less than the minimum value are rendered with the bottom color (and opacity). Pixels with values equal to or greater than the maximum value are rendered with the top color and opacity.

```
"colorMap": {
  "type": "intervals",
  "colorTable": {
    "-1": [255, 255, 255, 0],
    "-0.5": [125, 125, 125, 255],
    "1": [0, 0, 0, 255]
  }
}
```

In this case, all pixels with values in the interval (-inf, -0.5) are replaced with [255, 255, 255, 0], pixels in the interval [-0.5, 1) are replaced with [125, 125, 125, 255], and pixels with value >= 1 are replaced with [0, 0, 0, 255].

– `ramp` - Same as "intervals", but instead of using the color of the lowest value entry, linear interpolation between the lowest value entry and highest value entry, based on the pixel value, is performed.

```
"colorMap": {
  "type": "ramp",
  "colorTable": {
    "-1": [255, 255, 255, 0],
    "-0.5": [125, 125, 125, 255],
    "1": [0, 0, 0, 255]
  }
}
```

Pixels with value -0.75 are replaced with color [189, 189, 189, 127], because they sit in the middle of the distance between -1 and -0.5, so they get, on each channel, the color value in the middle. The interpolation formula for a pixel of value x, where 2 consecutive entries in the colorTable a, b with $a \leq x \leq b$, is:

$$resultColor = \frac{b-x}{b-a} * colorTable[b] + \frac{x-a}{b-a} * colorTable[a]$$

For the example above, a = -1, x = -0.75, b = -0.5, colorTable[a] = [255, 255, 255, 0], colorTable[b] = [125, 125, 125, 255], so:

$$\begin{aligned} resultColor = & \frac{-0.5+0.75}{-0.5+1} * [255,255,255,0] + \\ & \frac{-0.75+1}{-0.5+1} * [125,125,125,255] \\ & = 0.5 * [255,255,255,0] + 0.5 * [125,125,125,255] \\ & = [127,127,127,0] + [62,62,62,127] \\ & = [189,189,189,127] \end{aligned}$$

Note the integer division, because the colors are of type unsigned char.

- `colorPalette` - Similar to `colorMap`, however, it allows specifying color information on a metadata level, rather than by actually transforming array pixel values; for details see the GDAL documentation. It is an object that contains several optional properties:
  - `paletteInterp` - Indicate how the entries in the colorTable should be interpreted; allowed values are "Gray", "RGB", "CMYK", "HSL" (default "RGB");
  - `colorInterp` - Array of color interpretations for each band; allowed values are Undefined, Gray, Palette, Red, Green, Blue, Alpha, Hue, Saturation, Lightness, Cyan, Magenta, Yellow, Black, YCbCr_Y, YCbCr_Cb, YCbCr_Cr, YCbCr_Cr;
  - `colorTable` - Array of arrays, each containing 1, 3, or 4 short values (depending on the `colorInterp`) for each color entry; to associate a color with an array cell value, the cell value is used as a subscript into the color table (starting from 0).

NetCDF

The following are mandatory options when encoding to NetCDF:

- `variables` - Specify variable names for each band of the MDD, dimension names if they need to be saved as coordinate variables, as well as non-data grid mapping variables. There are two ways to specify the variables:
  1. An array of strings for each variable name, e.g. `["var1", "var2"]`; no coordinate variables should be specified in this case, as there is no way to specify the data for them;
  2. An object of variable name - object pairs, where each object lists the following variable details:
     - `metadata` - An object of string key-value pairs which are added as attributes to the variable;
     - `type` - Type of the data values this variable contains relevant (and required) for coordinate or non-data variables; allowed values are "byte", "char", "short", "ushort", "int", "uint", "float", and "double";
     - `data` - An array of data values for the variable relevant (and required) only for coordinate variables (as regular variables get their data values from the array to be encoded); the number of values must match the dimension extent;

     If the variable name is not listed in the `dimensions` array and still has a `data` attribute, then it will be considered to be a non-data variable and will not be used for storing MDD band data; the `data` attribute is ignored in this case, so the value for it can be an empty JSON array `[]`.
- `dimensions` - An array of names for each dimension, e.g. `["Lat","Long"]`.

CSV / JSON

Data encoded with CSV or JSON is a comma-separated list of values, such that each row of values (for every dimension, not just the last one) is between `{` and `}` braces (`[` and `]` for JSON). The table below documents all *"formatParameters"* options that allow controlling the output, and the default settings for both formats.

Table 4.7: optional options for controlling CSV / JSON encoding.

option	description	CSV default	JSON default
`order`	array linearization order, can be “outer_inner” (default, last dimension iterates fastest, i.e. column-major for 2-D), or vice-versa, “inner_outer”.	“outer_inner”	“outer_inner”
`trueValue`	string denoting true values	“t”	“true”
`falseValue`	string denoting false values	“f”	“false”
`dimensionStart`	string to indicate starting a new dimension slice	“{“	“[“
`dimensionEnd`	string to indicate ending a dimension slice	“}”	“]”
`dimensionSeparator`	separator between dimension slices	“,”	“,”
`valueSeparator`	separator between cell values	“,”	“,”
`componentSeparator`	separator between components of struct cell values	” “	” “
`structValueStart`	string to indicate starting a new struct value	“\"”	“\"”
`structValueEnd`	string to indicate ending a new struct value	“\"”	“\"”
`outerDelimiters`	wrap output in dimensionStart and dimensionEnd	false	true

Examples

GDAL

This query extracts PNG images (one for each tuple) from collection `mr`:

```
select encode( mr, "png" )
from mr
```

Transpose the last two axes of the output before encoding to PNG:

```
select encode(c, "png", "{ \"transpose\": [0,1] }") from mr2 as c
```

NetCDF

Add some global attributes as metadata in netcdf:

```
select encode(c, "netcdf", "{ \"transpose\": [1,0], \"nodata\":␣
↪[100],
    \"metadata\": { \"new_metadata\": \"This is a new added␣
↪metadata\" } }")
from test_mean_summer_airtemp as c
```

The format parameters below specify the variables to be encoded in the result NetCDF file (Lat, Long, forecast, and drought_code); of these Lat, Long, and forecast are dimension variables for which the values are specified in the `"data"` array, which leaves drought_code is the proper variable for encoding the array data.

```
"dimensions": [ "Lat", "Long", "forecast" ],
"variables": {
  "Lat": {
    "type": "double",
    "data": [ 34.650524, 34.509953 ],
    "name": "Lat",
    "metadata": {
      "standard_name": "latitude",
      "units": "degrees_north",
      "axis": "Y"
    }
  },
  "Long": {
    "type": "double",
    "data": [ 29.671875, 29.8125 ],
    "name": "Long",
    "metadata": {
      "standard_name": "longitude",
      "units": "degrees_east",
      "axis": "X"
    }
  },
  "forecast": {
    "type": "double",
    "data": [ 0, 3 ],
    "name": "forecast"
  },
  "drought_code": {
    "type": "float",
    "name": "drought_code",
    "metadata": {
      "description": "Global Fire Forecast- Drought Code",
      "units": "10^0"
    }
  }
```

(continues on next page)

(continued from previous page)

```
}
```

Below format parameters for a rotated grid are specified, which define a "rotated_pole" grid mapping variable in addition to the dimension variables (rlong and rlat) and the band variable CAPE_ML. More information on grid mappings can be found here.

```
"dimensions": [
  "rlon",
  "rlat"
],
"variables": {
  "rotated_pole": {
    "type": "int",
    "name": "rotated_pole",
    "metadata": {
      "grid_mapping_name": "rotated_latitude_longitude",
      "grid_north_pole_longitude": "-170",
      "units": "degrees",
      "axis": "X"
    },
    "data": []
  },
  "rlon": {
    "type": "double",
    "data": [-5,-4.975,-4.95,-4.925],
    "name": "rlon",
    "metadata": {
      "standard_name": "grid_longitude",
      "long_name": "longitude in rotated pole grid",
      "grid_north_pole_latitude": "40",
      "semi_major_axis": "6371229."
    }
  },
  "rlat": {
    "type": "double",
    "data": [-4.925,-4.95,-4.975,-5],
    "name": "rlat",
    "metadata": {
      "standard_name": "grid_latitude",
      "long_name": "latitude in rotated pole grid",
      "units": "degrees",
      "axis": "Y"
    }
  },
  "CAPE_ML": {
    "type": "float",
    "name": "CAPE_ML",
    "metadata": {
      "description": "Count of the number of observations from the␣
↪SeaWiFS sensor",
```

(continues on next page)

(continued from previous page)

```
      "units": "J kg-1",
      "long_name": "Convective Available Potential Energy, mean␣
↪layer",
      "param": "6.7.0",
      "grid_mapping": "rotated_pole",
      "realization": "1",
      "ensemble_members": "20",
      "forecast_init_type": "192"
    }
  }
}
```

CSV / JSON

Suppose we have array `A = <[0:1,0:1] 0, 1; 2, 3>`. Encoding to CSV by default

```
select encode(A, "csv") from A
```

will result in the following output:

```
{{0, 1}, {2, 3}}
```

while encoding to JSON with:

```
select encode(A, "json") from A
```

will result in the following output:

```
[[0, 1], [2, 3]]
```

Specifying `inner_outer` order with

```
select encode(A, "csv", "{ \"formatParameters\":
                         { \"order\": \"inner_outer\" } }") from A
```

will result in the following output (left-most dimensions iterate fastest):

```
{{0, 2}, {1, 3}}
```

Let `B` be an RGB array `<[0:0,0:1] {0c, 1c, 2c}, {3c, 4c, 5c}>`. Encoding it to CSV with default order will result in the following output:

{“0 1 2”,”3 4 5”}

4.12 Object identifiers

Function `oid()` gives access to an array's object identifier (OID). It returns the local OID of the database array. The input parameter must be a variable associated with a collection, it cannot be an array expression. The reason is that `oid()` can be applied to only to persistent arrays which are stored in the database; it cannot be applied to query result arrays - these are not stored in the database, hence do not have an OID.

Syntax

```
oid( variable )
```

Example

The following example retrieves the MDD object with local OID 10 of set `mr`:

```
select mr
from mr
where oid( mr ) = 10
```

The following example is incorrect as it tries to get an OID from a non-persistent result array:

```
select oid( mr * 2 ) -- illegal example: no expressions
from mr
```

Fully specified external OIDs are inserted as strings surrounded by brackets:

```
select mr
from mr
where oid( mr ) = < localhost | RASBASE | 10 >
```

In that case, the specified system (system name where the database server runs) and database must match the one used at query execution time, otherwise query execution will result in an error.

4.12.1 Expressions

Parentheses

All operators, constructors, and functions can be nested arbitrarily, provided that each sub-expression's result type matches the required type at the position where the sub-expression occurs. This holds without limitation for all arithmetic, Boolean, and array-valued expressions. Parentheses can (and should) be used freely if a particular desired evaluation precedence is needed which does not follow the normal left-to-right precedence.

Example

```
select (rgb.red + rgb.green + rgb.blue) / 3c
from rgb
```

Operator Precedence Rules

Sometimes the evaluation sequence of expressions is ambiguous, and the different evaluation alternatives have differing results. To resolve this, a set of precedence rules is defined. You will find out that whenever operators have their counterpart in programming languages, the rasdaman precedence rules follow the same rules as are usual there.

Here the list of operators in descending strength of binding:

- dot ".", trimming, section
- unary `-`
- `sqrt`, `sin`, `cos`, and other unary arithmetic functions
- `*`, `/`
- `+`, `-`
- `<`, `<=`, `>`, `>=`, `!=`, `=`
- `and`
- `or`, `xor`
- ":" (interval constructor), `condense`, `marray`
- `overlay`, `concat`
- In all remaining cases evaluation is done left to right.

4.13 Null Values

"Null is a special marker used in Structured Query Language (SQL) to indicate that a data value does not exist in the database. NULL is also an SQL reserved keyword used to identify the Null special marker." (Wikipedia) In fact, null introduces a three-valued logic where the result of a Boolean operation can be null itself; likewise, all other operations have to respect null appropriately. Said Wikipedia article also discusses issues the SQL language has with this three-valued logic.

For sensor data, a Boolean null indicator is not enough as null values can mean many different things, such as "no value given", "value cannot be trusted", or "value not known". Therefore, rasdaman refines the SQL notion of null:

- Any value of the data type range can be chosen to act as a null value;
- a set of cell values can be declared to act as null (in contrast to SQL where only one null per attribute type is foreseen).

Caveat

Note that defining values as nulls reduces the value range available for known values. Additionally, computations can yield values inadvertently (null values themselves are not changed during operations, so there is no danger from this side). For example, if 5 is defined to mean null then addition of two non-null values, such as 2+3, yields a null.

Every bit pattern in the range of a numeric type can appear in the database, so no bit pattern is left to represent "null". If such a thing is desired, then the database designer must provide, e.g., a separate bit map indicating the status for each cell.

To have a clear semantics, the following rule holds:

Uninitialized value handling

A cell value not yet addressed, but within the current domain of an MDD has a value of zero by definition; this extends in the obvious manner to composite cells.

Remark

Note the limitation to the *current* domain of an MDD. While in the case of an MDD with fixed boundaries this does not matter because always *definition domain* = *current domain*, an MDD with variable boundaries can grow and hence will have a varying current domain. Only cells inside the current domain can be addressed, be they uninitialized/null or not; addressing a cell outside the current domain will result in the corresponding exception.

Masks as alternatives to null

For example, during piecewise import of satellite images into a large map, there will be areas which are not written yet. Actually, also after completely creating the map of, say, a country there will be untouched areas, as normally no country has a rectangular shape with axis-parallel boundaries. The outside cells will be initialized to 0 which may or may not be defined as null. Another option is to define a Boolean mask array of same size as the original array where each mask value contains *true* for "cell valid" and *false* for "cell invalid. It depends on the concrete application which approach benefits best.

4.13.1 Nulls in MDD-Valued Expressions

Dynamically Set/Replace the Null Set

The null set of an MDD value resulting from a sub-expression can be dynamically changed on-the-fly with a postfix `null values` operator as follows:

```
mddExp null values nullSet
```

As a result *mddExp* will have the null values specified by *nullSet*; if *mddExp* already had a null set, it will be replaced.

Null Set Propagation

The null value set of an MDD is part of its type definition and, as such, is carried along over the MDD's lifetime. Likewise, MDDs which are generated as intermediate results during query processing have a null value set attached. Rules for constructing the output MDD null set are as follows:

- The null value set of an MDD generated through an `marray` operation is empty[13].
- The null value set of an operation with one input MDD object is identical to the null set of this input MDD.

[13] This is going to be changed in the near future.

- The null value set of an operation with two input MDD objects is the union of the null sets of the input MDDs.
- The null value set of an MDD expression with a postfix `null values` operator is equal to the null set specified by it.

Null Values in Operations

Subsetting (trim and slice operations, as well as `struct` selection, etc.) perform just as without nulls and deliver the original cell values, be they null (relative to the MDD object on hand) or not. The null value set of the output MDD is the same as the null value set of the input MDD.

In MDD-generating operations with only one input MDD (such as marray and unary induced operations), if the operand of a cell operation is null then the result of this cell operation is null.

Generally, if somewhere in the input to an individual cell value computation a null value is encountered then the overall result will be null - in other words: *if at least one of the operands of a cell operation is null then the overall result of this cell operation is null.*

Exceptions:

- Comparison operators (that is: `==`, `!=`, `>`, `>=`, `<`, `<=`) encountering a null value will *always* return a Boolean value; for example, both `n == n` and `n != n` (for any null value `n`) will evaluate to `false`.
- In a cast operation, nulls are treated like regular values.
- In a `scale()` operation, null values are treated like regular values[14].
- Format conversion of an MDD object ignores null values. Conversion from some data format into an MDD likewise imports the actual cell values; however, during any eventual further processing of the target MDD as part of an **update** or **insert** statement, cell values listed in the null value set of the pertaining MDD definition will be interpreted as null and will not overwrite persistent non-null values.

Choice of Null Value

If an operation computes a null value for some cell, then the null value effectively assigned is determined from the MDD's type definition.

If the overall MDD whose cell is to be set has exactly one null value, then this value is taken. If there is more than one null value available in the object's definition, then one of those null values is picked non-deterministically. If the null set of the MDD is empty then no value in the MDD is considered a null value.

Example

Assume an MDD `a` holding values `<0, 1, 2, 3, 4, 5>` and a null value set of `{2, 3}`. Then, `a*2` might return `<0, 2, 2, 2, 8, 10>`. However, `<0, 2, 3, 3, 8, 10>` and `<0, 2, 3, 2, 8, 10>` also are valid results, as the null value gets picked non-deterministically.

[14] This will be changed in future.

4.13.2 Nulls in Aggregation Queries

In a condense operation, cells containing nulls do not contribute to the overall result (in plain words, nulls are ignored).

If all values are null, then the result is the identity element in this case, e.g. `0` for `+`, `true` for `and`, `false` for `or`, maximum value possible for the result base type for `min`, minimum value possible for the result base type for `max`, `0` for `count_cells`.

The scalar value resulting from an aggregation query does not carry a null value set like MDDs do; hence, during further processing it is treated as an ordinary value, irrespective of whether it has represented a null value in the MDD acting as input to the aggregation operation.

4.13.3 Limitations

All cell components of an MDD share the same same set of nulls, it is currently not possible to assign individual nulls to cell type components.

4.13.4 NaN Values

NaN ("not a number") is the representation of a numeric value representing an undefined or unrepresentable value, especially in floating-point calculations. Systematic use of NaNs was introduced by the IEEE 754 floating-point standard (Wikipedia).

In rasql, `nan` (double) and `nanf` (float) are symbolic floating point constants that can be used in any place where a floating point value is allowed. Arithmetic operations involving `nans` always result in `nan`. Equality and inequality involving nans work as expected, all other comparison operators return false.

If the encoding format used supports NaN then rasdaman will encode/decode NaN values properly.

Example

```
select count_cells( c != nan ) from c
```

4.14 Miscellaneous

4.14.1 rasdaman version

Builtin function version() returns a string containing information about the rasdaman version of the server, and the gcc version used for compiling it. The following query

```
select version()
```

will generate a 1-D array of cell type char containing contents similar to the following:

```
rasdaman 9.6.0 on x86_64-linux-gnu, compiled by g++
(Ubuntu 5.4.1-2ubuntu1~16.04) 5.4.1 20160904
```

Warning: The message syntax is not standardized in any way and may change in any rasdaman version without notice.

4.14.2 Retrieving Object Metadata

Sometimes it is desirable to retrieve metadata about a particular array. To this end, the `dbinfo()` function is provided. It returns a 1-D char array containing a JSON encoding of key array metadata:

- Object identifier;
- Base type, mdd type name, set type name;
- Total size of the array;
- Number of tiles and further tiling information: tiling scheme, tile size (if specified), and tile configuration;
- Index information: index type, and further details depending on the index type.

The output format is described below by way of an example.

Syntax

```
dbinfo( mddExp )

dbinfo( mddExp , formatParams )
```

Example

```
$ rasql -q 'select dbinfo(c) from mr2 as c' --out string
{
  "oid": "150529",
  "baseType": "marray <char>",
  "mddTypeName": "GreyImage",
  "setTypeName": "GreySet",
  "tileNo": "1",
  "totalSize": "54016B",
  "tiling": {
    "tilingScheme": "no_tiling",
    "tileSize": "2097152",
    "tileConfiguration": "[0:511,0:511]"
  },
  "index": {
    "type": "rpt_index",
    "indexSize": "0",
```

(continues on next page)

(continued from previous page)

```
    "PCTmax": "4096B",
    "PCTmin": "2048B"
  }
}
```

The function supports a string of format parameters as the second argument. By now the only supported parameter is printTiles. It can take multiple values: “embedded”, “json”, “svg”. Example of syntax:

```
select dbinfo(c, "printtiles=svg") from test_overlap as c
```

Parameter “printiles=embedded” will print additionally domains of every tile.

```
$ rasql -q 'select dbinfo(c, "printtiles=embedded") from test_grey␣
↪as c' --out string
{
 "oid": "136193",
 "baseType": "marray <char, [*:*,*:*]>",
 "setTypeName": "GreySet",
 "mddTypeName": "GreyImage",
 "tileNo": "48",
 "totalSize": "54016",
 "tiling": {
        "tilingScheme": "aligned",
        "tileSize": "1500",
        "tileConfiguration": "[0:49,0:29]"",
        "tileDomains":
        [
                "[100:149,210:210]",
                "[150:199,0:29]",
                "[150:199,30:59]",
                "[150:199,60:89]",
                "[150:199,90:119]",
                "[150:199,120:149]",
                "[150:199,150:179]",
                "[150:199,180:209]",
                "[150:199,210:210]",
                "[200:249,0:29]",
                "[200:249,30:59]",
                "[200:249,60:89]",
                "[200:249,90:119]",
                "[200:249,120:149]",
                "[200:249,150:179]",
                "[200:249,180:209]",
                "[200:249,210:210]",
                "[250:255,0:29]",
                "[250:255,30:59]",
                "[250:255,60:89]",
                ...
```

(continues on next page)

(continued from previous page)

```
        ]
 },
 "index": {
        "type": "rpt_index",
        "PCTmax": "4096",
        "PCTmin": "2048"
 }
}
```

Option "json" will output only the tile domains as a json object.

```
["[100:149,210:210]","[150:199,0:29]",..."[0:49,30:59]"]
```

Last option "svg" will output tiles as svg that can be visualised. Example:

```
<svg width="array width" height="array height">
        <rect x="100" y="210" width="50" height="1" id="1232"></
↪rect>
        <rect x="150" y="0" width="50" height="30" id="3223"></rect>
        ...
</svg>
```

Note: This function can only be invoked on persistent MDD objects, not on derived (transient) MDDs.

Warning: This function is in beta version. While output syntax is likely to remain largely unchanged, invocation syntax is expected to change to something like

```
describe array oidExp
```

4.15 Arithmetic Errors and Other Exception Situations

During query execution, a number of situations can arise which prohibit to deliver the desired query result or database update effect. If the server detects such a situation, query execution is aborted, and an error exception is thrown. In this Section, we classify the errors that occur and describe each class.

However, we do not go into the details of handling such an exception - this is the task of the application program, so we refer to the resp. API Guides.

4.15.1 Overflow

Candidates

Overflow conditions can occur with `add_cells` and induced operations such as +.

System Reaction

The overflow will be silently ignored, producing a result represented by the bit pattern pruned to the available size. This is in coherence with overflow handling in performance-oriented programming languages.

Remedy

Query coding should avoid potential overflow situations by applying numerical knowledge - simply said, the same care should be applied as always when dealing with numerics.

It is worth being aware of the *type coercion rules <type-coercion>* in rasdaman and overflow handling in C++. The type coercion rules have been crafted to avoid overflow as much as possible, but of course it remains a possibility. Adding or multiplying two chars for example is guaranteed to not overflow. However, adding or multyplying two ulongs would result in a ulong by default, which may not be large enough to hold the result. Therefore, it may be worth casting to double in this case based on knowledge about the data.

Checking for overflow with a case statement like the below will not work as one might expect and is hence not recommended:

```
case
when a.longatt1 * a.longatt2 > 2147483647 then 2147483647
else a.longatt1 * a.longatt2
end
```

If `a.longatt1 * a.longatt2` overflows, the result is undefined behavior according to C++ so it is not clear what the result value would be in this case. It will never be larger than the maximum value of 32-bit signed integer, however, because that is the result type according to the type coercion rules. Hence the comparison to 2147483647 (maximum value of 32-bit signed integer) will never return true.

4.15.2 Illegal operands

Candidates

Division by zero, non-positive argument to logarithm, negative arguments to the square root operator, etc. are the well-known candidates for arithmetic exceptions.

The IEEE 754 standard lists, for each operation, all invalid input and the corresponding operation result (Sections *Select Clause: Result Preparation*, *From Clause: Collection Specification*, *Multidimensional Intervals*). Examples include:

- division(0,0), division(INF,INF)
- sqrt(x) where x < 0
- log(x) where x < 0

System Reaction

In operations returning floating point numbers, results are produced in conformance with IEEE 754. For example, 1/0 results in nan.

In operations returning integer numbers, results for illegal operations are as follows:

- `div(x, 0)` leads to a "division by zero" exception
- `mod(x, 0)` leads to a "division by zero" exception

Remedy

To avoid an exception the following code is recommended for `a div b` (replace accordingly for `mod`), replacing all illegal situations with a result of choice, `c`:

```
case when b = 0 then c else div(a, b) end
```

If the particular situation allows, it may be more efficient to cast to floating-point, and cast back to integer after the division (if an integer result is wanted):

```
(long)((double)a / b)
```

Division by 0 will result in Inf in this case, which turns into 0 when cast to integer.

4.15.3 Access Rights Clash

If a database has been opened in read-only mode, a write operation will be refused by the server; "write operation" meaning an insert, update, or delete statement.

4.16 Database Retrieval and Manipulation

4.16.1 Collection Handling

Create a Collection

The create collection statement is used to create a new, empty MDD collection by specifying its name and type. The type must exist in the database schema. There must not be another collection in this database bearing the name indicated.

Syntax

```
create collection collName typeName
```

Example

```
create collection mr GreySet
```

Drop a Collection

A database collection can be deleted using the `drop collection` statement.

Syntax

```
drop collection collName
```

Example

```
drop collection mr1
```

Alter Collection

The type of a collection can be changed using the `alter collection` statement. The new collection type is accordingly checked for compatibility (same cell type, dimensionality) as the existing type of the collection before setting it.

Syntax

```
alter collection collName
set type newCollType
```

Example

```
alter collection mr2
set type GreySetWithNullValues
```

Retrieve All Collection Names

With the following rasql statement, a list of the names of all collections currently existing in the database is retrieved; both versions below are equivalent:

```
select RAS_COLLECTIONNAMES
from RAS_COLLECTIONNAMES

select r
from RAS_COLLECTIONNAMES as r
```

Note that the meta collection name, `RAS_COLLECTIONNAMES`, must be written in upper case only. No operation in the select clause is permitted. The result is a set of one-dimensional char arrays, each one holding the name of a database collection. Each such char array, i.e., string is terminated by a zero value ('0').

4.16.2 Select

The select statement allows for the retrieval from array collections. The result is a set (collection) of items whose structure is defined in the select clause. Result items can be arrays, atomic values, or structs. In the where clause, a condition can be expressed which acts as a filter for the result set. A single query can address several collections.

Syntax

```
select resultList
from collName [ as collIterator ]
     [, collName [ as collIterator ] ] ...

select resultList
from collName [ as collIterator ]
     [, collName [ as collIterator ] ] ...
where booleanExp
```

Examples

This query delivers a set of grayscale images:

```
select mr[100:150,40:80] / 2
from mr
where some_cells( mr[120:160, 55:75] > 250 )
```

This query, on the other hand, delivers a set of integers:

```
select count_cells( mr[120:160, 55:75] > 250 )
from mr
```

Finally, this query delivers a set of **struct**s, each one with an integer and a 2-D array component:

```
select struct { max_cells( a ), a }
from mr as a
```

4.16.3 Insert

MDD objects can be inserted into database collections using the **insert** statement. The array to be inserted must conform with the collection's type definition concerning both cell type and spatial domain. One or more variable bounds in the collection's array type definition allow degrees of freedom for the array to be inserted. Hence, the resulting collection in this case can contain arrays with different spatial domain.

Syntax

```
insert into collName
values mddExp
```

collName specifies the name of the target set, *mddExp* describes the array to be inserted.

Example

Add a black image to collection mr1.

```
insert into mr1
values marray x in [ 0:255, 0:210 ]
values 0c
```

See the programming interfaces described in the *rasdaman Developer's Guides* on how to ship external array data to the server using `insert` and `update` statements.

4.16.4 Update

The **update** statement allows to manipulate arrays of a collection. Which elements of the collection are affected can be determined with the **where** clause; by indicating a particular OID, single arrays can be updated.

An update can be *complete* in that the whole array is replaced or *partial*, i.e., only part of the database array is changed. Only those array cells are affected the spatial domain of the replacement expression on the right-hand side of the `set` clause. Pixel locations are matched pairwise according to the arrays' spatial domains. Therefore, to appropriately position the replacement array, application of the `shift()` function (see *Shifting a Spatial Domain*) can be necessary; for more details and practical examples continue to *Partial Updates*.

As a rule, the spatial domain of the righthand side expression must be equal to or a subset of the database array's spatial domain.

Cell values contained in the update null set will *not* overwrite existing cell values which are not null. The update null set is taken from the source MDD if it is not empty, otherwise it will be taken from the target MDD.

Syntax

```
update collName as collIterator
set    updateSpec assign mddExp

update collName as collIterator
set    updateSpec assign mddExp
where  booleanExp
```

where *updateSpec* can optionally contain a restricting minterval (see examples further below):

```
var
var [ mintervalExp ]
```

Each element of the set named *collName* which fulfils the selection predicate *booleanEpxr* gets assigned the result of *mddExp*. The right-hand side mddExp overwrites the corresponding area in the collection element; note that no automatic shifting takes place: the spatial domain of mddExp determines the very place where to put it.

If you want to include existing data from the database in *mddExp*, then this needs to be specified in an additional `from` clause, just like in normal `select` queries. The syntax in this case is

```
update collName as collIterator
set    updateSpec assign mddExp
from   existingCollName [ as collIterator ]
       [, existingCollName [ as collIterator ] ] ...
where  booleanExp
```

Example

An arrow marker is put into the image in collection `mr2`. The appropriate part of `a` is selected and added to the arrow image which, for simplicity, is assumed to have the appropriate spatial domain.

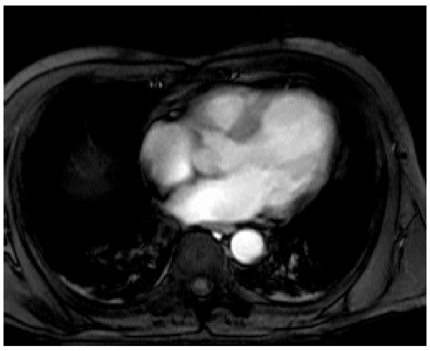


Figure 4.21: Original image of collection `mr2`

```
update mr2 as a
set a assign a[0 : 179 , 0:54] + $1/2c
```

The argument $1 is the arrow image (Figure 4.22) which has to be shipped to the server along with the query. It is an image showing a white arrow on a black background. For more information on the use of $ variables you may want to consult the language binding guides of the rasdaman Documentation Set.

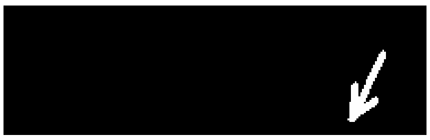


Figure 4.22: Arrow used for updating

Looking up the **mr2** collection after executing the update yields the result as shown in Figure 4.23:

Note: The replacement expression and the MDD to be updated (i.e., left and right-hand side of the **assign** clause) in the above example must have the same dimensionality. Updating a (lower-dimensional) section of an MDDs can be achieved through a section operator indicating the “slice” to be modified. The following query appends one line to a fax (which is assumed to be extensible in the second dimension):

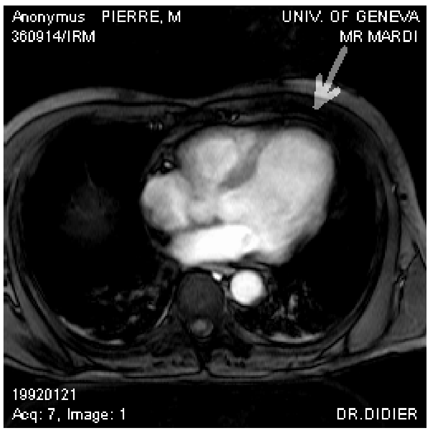


Figure 4.23: Updated collection mr2

```
update fax as f
set f[ *:* , sdom(f)[1].hi+1 ] assign $1
```

The example below updates target collection `mr2` with data from `rgb` (collection that exists already in the database):

```
update mr2 as a
set a assign b[ 0:150, 50:200 ].red
from rgb as b
```

Partial Updates

Often very large data files need to be inserted in rasdaman, which don't fit in main memory. One way to insert such a large file is to split it into smaller parts, and then import each part one by one via partial updates, until the initial image is reconstructed in rasdaman.

This is done in two steps: initializing an MDD in a collection, and inserting each part in this MDD.

Initialization

Updates replace an area in a target MDD object with the data from a source MDD object, so first the target MDD object needs to be initialized in a collection. To initialize an MDD object it's sufficient to insert an MDD object of size 1 (a single point) to the collection:

```
insert into Coll
values marray it in [0:0,0:0,...] values 0
```

Note that the MDD constructed with the marray constructor should match the type of Coll (dimension and base type). If the dimension of the data matches the Coll dimensions (e.g. both are 3D), then inserting some part of the data would work as well. Otherwise, if data is 2D and Coll is 3D for example, it is necessary to initialize an array in the above way.

Updates

After we have an MDD initialized in the collection, we can continue with updating it with the individual parts using the update statement in rasql.

Refering to the update statement syntax, `mddExp` can be any expression that results in an MDD object M, like an marray construct, a format conversion function, etc. The position where M will be placed in the target MDD (`collIterator`) is determined by the spatial domain of M. When importing data in some format via the decode function, by default the resulting MDD has an sdom of `[0:width,0:height,..]`, which will place M at `[0,0,..]` in the target MDD. In order to place it in a different position, the spatial domain of M has to be explicitly set with the shift function in the query. For example:

```
update Coll as c set c
assign shift(decode($1),[100,100])
```

The update statement allows one to dynamically expand MDDs (up to the limits of the MDD type if any have been specified), so it's not necessary to fully materialize an MDD.

When the MDD is first initialized with:

```
insert into Coll
values marray it in [0:0,0:0,...] values 0
```

it has a spatial domain of `[0:0,0:0,...]` and only one point is materialized in the database. Updating this MDD later on, further expands the spatial domain if the source array M extends outside the sdom of target array T.

Example: 3D timeseries

Create a 3D collection first for arrays of type float:

```
create collection Coll FloatSet3
```

Initialize an array with a single cell in the collection:

```
insert into Coll
values marray it in [0:0,0:0,0:0] values 0f
```

Update array with data at the first time slice:

```
update Coll as c set c[0,*:*,*:*]
assign decode($1)
```

Update array with data at the second time slice, but shift spatially to `[10,1]`:

```
update Coll as c set c[1,*:*,*:*]
assign shift( decode($1), [10,1] )
```

And so on.

Example: 3D cube of multiple 3D arrays

In this case we build a 3D cube by concatenating multiple smaller 3D cubes along a certain dimension, i.e. build a 3D mosaic.

Create the 3D collection first (suppose it's for arrays of type float):

```
create collection Coll FloatSet3
```

Initialize an array with a single cell in the collection:

```
insert into Coll
values marray it in [0:0,0:0,0:0] values 0f
```

Update array with the first cube, which has itself sdom `[0:3,0:100,0:100]`:

```
update Coll as c set c[0:3,0:100,0:100]
assign decode($1, "netcdf")
```

After this Coll has sdom `[0:3,0:100,0:100]`.

Update array with the second cube, which has itself sdom `[0:5,0:100,0:100]`; note that now we want to place this one on top of the first one with respect to the first dimension, so its origin must be shifted by 5 so that its sdom will be in effect `[5:10,0:100,0:100]`:

```
update Coll as c set c[5:10,0:100,0:100]
assign shift(decode($1, "netcdf"), [5,0,0])
```

The sdom of Coll is now `[0:10,0:100,0:100]`.

Update array with the third cube, which has itself sdom `[0:2,0:100,0:100]`; note that now we want to place this one next to the first two with respect to the second dimension and a bit higher by 5 pixels, so that its sdom will be in effect `[5:7,100:200,0:100]`:

```
update Coll as c set c[5:7,100:200,0:100]
assign shift(decode($1, "netcdf"), [5,100,0])
```

The sdom of Coll is now `[0:10,100:200,0:100]`.

4.16.5 Delete

Arrays are deleted from a database collection using the **delete** statement. The arrays to be removed from a collection can be further characterized in an optional **where** clause. If the condition is omitted, all elements will be deleted so that the collection will be empty afterwards.

Syntax

```
delete from collName [ as collIterator ]
[ where booleanExp ]
```

Example

```
delete from mr1 as a
where all_cells( a < 30 )
```

This will delete all "very dark" images of collection `mr1` with all pixel values lower than 30.

4.17 Transaction Scheduling

Since rasdaman 9.0, database transactions lock arrays on fine-grain level. This prevents clients from changing array areas currently being modified by another client.

4.17.1 Locking

Lock compatibility is as expected: read access involves shared ("S") locks which are mutually compatible while write access imposes an exclusive lock ("X") which prohibits any other access:

	S	X
S	+	-
X	-	-

Shared locks are set by `SELECT` queries, exclusive ones in `INSERT`, `UPDATE`, and `DELETE` queries.

Locks are acquired by queries dynamically as needed during a transaction. All locks are held until the end of the transaction, and then released collectively[15].

4.17.2 Lock Granularity

The unit of locking is a tile, as tiles also form the unit of access to persistent storage.

4.17.3 Conflict Behavior

If a transaction attempts to acquire a lock on a tile which has an incompatible lock it will abort with a message similar to the following:

```
Error: One or more of the target tiles are locked by another
transaction.
```

Only the query will return with an exception, the rasdaman transaction as such is not affected. It is up to the application program to catch the exception and react properly, depending on the particular intended behaviour.

[15] This is referred to as *Strict 2-Phase Locking* in databases.

4.17.4 Lock Federation

Locks are maintained in the PostgreSQL database in which rasdaman stores data. Therefore, all rasserver processes accessing the same RASBASE get synchronized.

4.17.5 Examples

The following two SELECT queries can be run concurrently against the same database:

```
rasql -q "select mr[0:10,0:10] from mr"

rasql -q "select mr[5:10,5:10] from mr"
```

The following two UPDATE queries can run concurrently as well, as they address different collections:

```
rasql -q "update mr set mr[0:10,0:10] \
          assign marray x in [0:10,0:10] values 127c" \
      --user rasadmin --passwd rasadmin

rasql -q "update mr2 set mr2[0:5,0:5] \
          assign marray x in [0:5,0:5] values 65c" \
      --user rasadmin --passwd rasadmin
```

From the following two queries, one will fail (the one which happens to arrive later) because the address the same tile:

```
rasql -q "update mr set mr[0:10,0:10] assign \
          marray x in [0:10,0:10] values 127c" \
      --user rasadmin --passwd rasadmin

rasql -q "update mr set mr[0:5,0:5] assign \
          marray x in [0:5,0:5] values 65c" \
      --user rasadmin --passwd rasadmin
```

4.17.6 Limitations

Currently, only tiles are locked, not other entities like indexes.

4.18 Linking MDD with Other Data

4.18.1 Purpose of OIDs

Each array instance and each collection in a rasdaman database has a identifier which is unique within a database. In the case of a collection this is the collection name and an object identifier (OID), whereas for an array this is only the OID. OIDs are generated by the system upon creation of an array instance, they do not change over an array's lifetime, and OIDs of deleted arrays will never be reassigned to other arrays. This way, OIDs form the means to unambiguously identifiy a particular array. OIDs can be used several ways:

- In rasql, OIDs of arrays can be retrieved and displayed, and they can be used as selection conditions in the condition part.
- OIDs form the means to establish references from objects or tuples residing in other databases systems to rasdaman arrays. Please refer for further information to the language-specific *rasdaman Developer's Guides* and the *rasdaman External Products Integration Guide* available for each database system to which rasdaman interfaces.

Due to the very different referencing mechanisms used in current database technology, there cannot be one single mechanism. Instead, rasdaman employs its own identification scheme which, then, is combined with the target DBMS way of referencing. See *Object identifier (OID) Constants* of this document as well as the *rasdaman External Products Integration Guide* for further information.

4.18.2 Collection Names

MDD collections are named. The name is indicated by the user or the application program upon creation of the collection; it must be unique within the given database. The most typical usage forms of collection names are

- as a reference in the from clause of a rasql query
- their storage in an attribute of a base DBMS object or tuple, thereby establishing a reference (also called foreign key or pointer).

4.18.3 Array Object identifiers

Each MDD array is world-wide uniquely identified by its object identifier (OID). An OID consists of three components:

- A string containing the system where the database resides (system name),
- A string containing the database (base name), and
- A number containing the local object id within the database.

The main purposes of OIDs are

- to establish references from the outside world to arrays and
- to identify a particular array by indicating one OID or an OID list in the search condition of a query.

4.19 Storage Layout Language

4.19.1 Overview

Tiling

To handle arbitrarily large arrays, rasdaman introduces the concept of *tiling* them, that is: partitioning a large array into smaller, non-overlapping sub-arrays which act as the unit of storage access during query evaluation. To the query client, tiling remains invisible, hence it constitutes a tuning parameter which allows database designers and administrators to adapt database storage layout to specific query patterns and workloads.

To this end, rasdaman offers a storage layout language for arrays which embeds into the query language and gives users comfortable, yet concise control over important physical tuning parameters. Further, this sub-language wraps several strategies which turn out useful in face of massive spatio-temporal data sets.

Tiling can be categorized into aligned and non-aligned (Figure 4.24).A tiling is *aligned* if tiles are defined through axis-parallel hyperplanes cutting all through the domain. Aligned tiling is further classified into *regular* and *aligned irregular* depending on whether the parallel hyperplanes are equidistant (except possibly for border tiles) or not. The special case of equally sized tile edges in all directions is called *cubed*.

Non-aligned tiling contains tiles whose faces are not aligned with those of their neighbors. This can be *partially aligned* with still some hyperplanes shared or *totally non-aligned* with no such sharing at all.

Syntax

We use a BNF variant where optional elements are indicated as

```
( ... )?
```

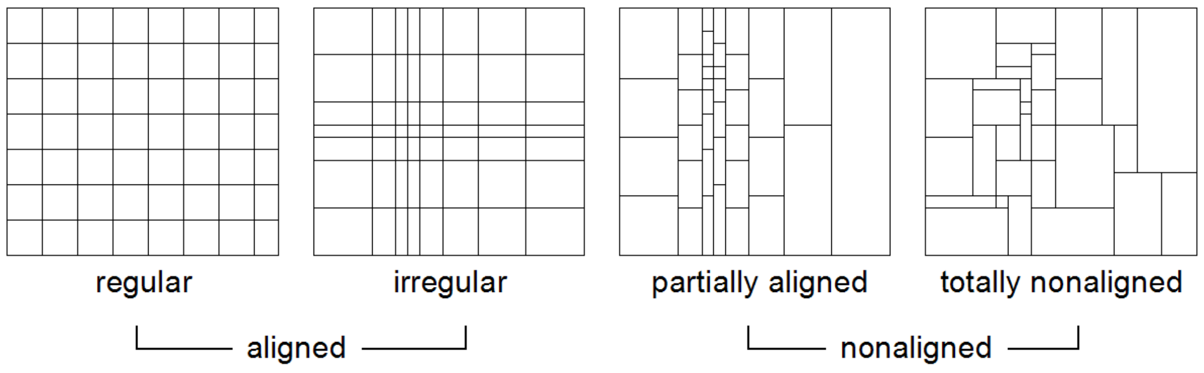


Figure 4.24: Types of tilings

to clearly distinguish them from the "[" and "]" terminals.

Tiling Through API

In the rasdaman C++ API (cf. C++ Guide), this functionality is available through a specific hierarchy of classes.

Introductory Example

The following example illustrates the overall syntax extension which the storage layout sublanguage adds to the **insert** statement:

```
insert into MyCollection
values ...
tiling
    area of interest [0:20,0:40],[45:80,80:85]
    tile size 1000000
```

4.19.2 General Tiling Parameters

Maximum Tile Size

The optional **tile size** parameter allows specifying a maximum tile size; irrespective of the algorithm employed to obtain a particular tile shape, its size will never exceed the maximum indicated in this parameter.

Syntax:

```
tile size t
```

where *t* indicates the tile size in bytes.

If nothing is known about the access patterns, tile size allows streamlining array tiling to architectural parameters of the server, such as DMA bandwidth and disk speed.

Tile Configuration

A tile configuration is a list of bounding boxes specified by their extent. No position is indicated, as it is the shape of the box which will be used to define the tiling, according to various strategies.

Syntax:

```
[ integerLit , ... , integerLit ]
```

For a `d`-dimensional MDD, the tile configuration consists of a vector of `d` elements where the i^{th} vector specifies the tile extent in dimension *i*, for 0le*i*<`d`. Each number indicates the tile extent in cells along the corresponding dimension.

For example, a tile configuration `[100, 100, 1000]` for a 3-D MDD states that tiles should have an extent of 100 cells in dimension 0 and 1, and an extent of 1,000 cells in dimension 2. In image timeseries analysis, such a stretching tiles along the time axis speeds up temporal analysis.

4.19.3 Regular Tiling

Concept

Regular tiling applies when there is some varying degree of knowledge about the subsetting patterns arriving with queries. We may or may not know the lower corner of the request box, the size of the box, or the shape (i.e., edge size ratio) of the box. For example, map viewing clients typically send several requests of fixed extent per mouse click to maintain a cache of tiles in the browser for faster panning. So the extent of the tile is known – or at least that tiles are quadratic. The absolute location often is not known, unless the client is kind enough to always request areas only in one fixed tile size and with starting points in multiples of the tile edge length.If additionally the configuration follows a uniform probability distribution then a cubed tiling is optimal.

In the storage directive, regular tiling is specified by providing a bounding box list, *TileConf*, and an optional maximum tile size:

Syntax

```
tiling regular TileConf ( tile size integerLit )?
```

Example

This line below dictates, for a 2-D MDD, tiles to be of size 1024 x 1024, except for border tiles (which can be smaller):

```
tiling regular [ 1024 , 1024 ]
```

4.19.4 Aligned Tiling

Concept

Generalizing from regular tiling, we may not know a good tile shape for all dimensions, but only some of them. An axis *p*in { 1, . . . , d } which never participates in any subsetting box is called a *preferred* (or *preferential*) *direction of access* and denoted as tc_p = *. An optimal tile structure in this situation extends to the array bounds in the preferential directions.

Practical use cases include satellite image time series stacks over some region. Grossly simplified, during analysis there are two distinguished access patterns (notwithstanding that others occur sometimes as well): either a time slice is read, corresponding to *tc* = (*, *, t) for some given time instance *t*, or a time series is extracted for one particular position (*x*, *y*) on the earth surface; this corresponds to *tc* = (*x*, *y*, *). The aligned tiling algorithm creates tiles as large as possible based on the constraints that (i) tile proportions adhere to tc and (ii) all tiles have the same size. The upper array limits constitute an exception: for filling the remaining gap (which usually occurs) tiles can be smaller and deviate from the configuration sizings. Figure 4.25 illustrates aligned tiling with two examples, for configuration *tc* = (1, 2) (left) and for *tc* =(1, 3, 4) (right).

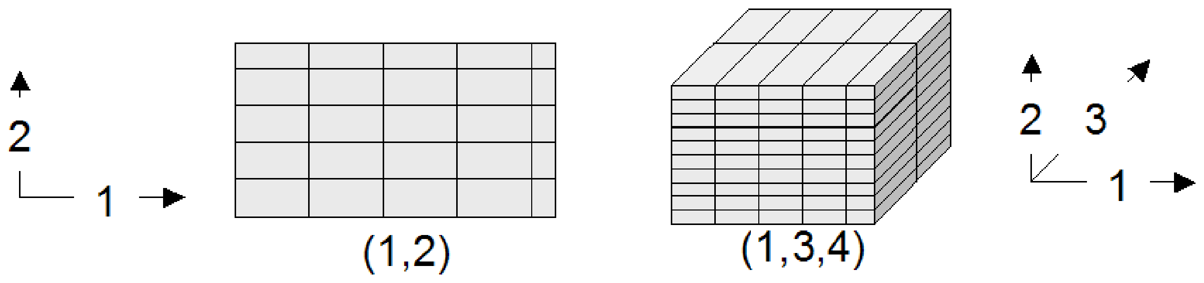


Figure 4.25: Aligned tiling examples

Preferential access is illustrated in Figure 4.26. Left, access is performed along preferential directions 1 and 2, corresponding to configuration *tc* = (*, *, 1). The tiling tothe right supports configuration *tc* = (4, 1, *) with preferred axis 3.

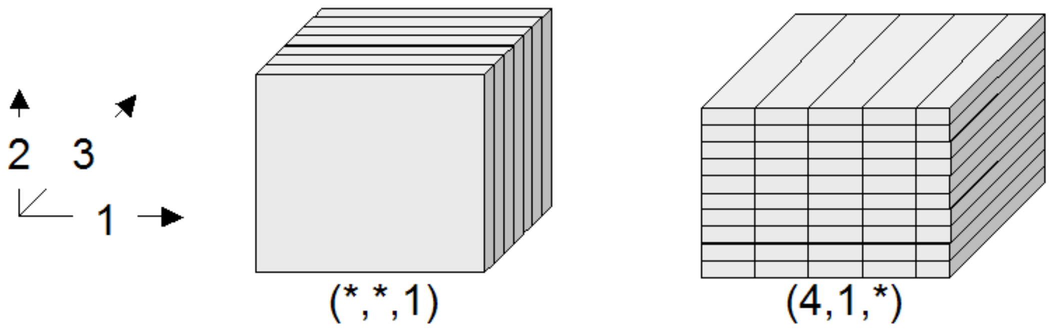


Figure 4.26: Aligned tiling examples with preferential access directions

The aligned tiling construction consists of two steps. First, a concrete tile shape is determined. After that, the extent of all tiles is calculated by iterating over the array's complete domain. In presence of more than one preferred directions - i.e., with a configuration containing more than one "*" values - axes are prioritized in descending order. This exploits the fact that array linearization is performed in a way that the "outermost loop" is the first dimension and the "innermost loop" the last. Hence, by clustering along higher coordinate axes a better spatial clustering is achieved.

Syntax

```
tiling aligned TileConf ( tile size IntLit )?
```

Example

The following clause accommodates map clients fetching quadratic images known to be no more than 512 x 512 x 3 = 786,432 bytes:

```
tiling aligned [1,1] tile size 786432
```

Important: Aligned tiling is the default strategy in rasdaman.

4.19.5 Directional Tiling

Concept

Sometimes the application semantics prescribes access in well-known coordinate intervals. In OLAP, such intervals are given by the semantic categories of the measures as defined by the dimension hierarchies, such as product categories which are defined for the exact purpose of accessing them group-wise in queries. Similar effects can occur with spatio-temporal data where, for example, a time axis may suggest access in units of days, weeks, or years. In rasdaman, if bounding boxes are well known then spatial access may be approximated by those; if they are overlapping then this is a case for area-of-interest tiling (see below), if not then directional tiling can be applied.

The tiling corresponding to such a partition is given by its Cartesian product. Figure 4.27 shows such a structure for the 2-D and 3-D case.

To construct it, the partition vectors are used to span the Cartesian product first. Should one of the resulting tiles exceed the size limit, as it happens in the tiles marked with a "*" in Figure 4.27, then a so-called sub-tiling takes place. Sub-tiling applies regular tiling by introducing additional local cutting hyperplanes. As these hyperplanes do not stretch through all tiles the resulting tiling in general is not regular. The resulting tile set guarantees that for answering queries using one of the subsetting patterns in part, or any union of these patterns, only those cells are read which will be delivered in the response. Further, if the area requested is smaller than the tile size limit then only one tile needs to be accessed.

Sometimes axes do not have categories associated. One possible reason is that subsetting is never performed along this axis, for example in an image time series where slicing is done along the time axis while the x/y image planes always are read in total. Similarly, for importing

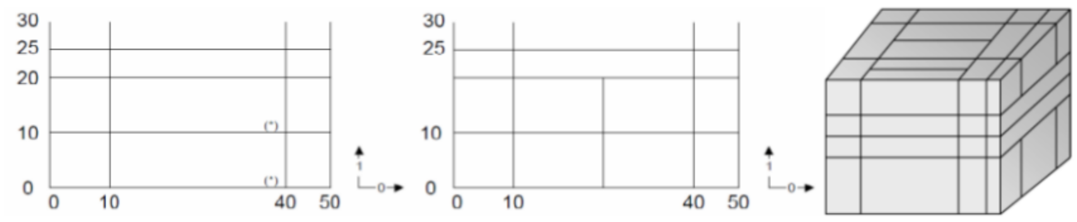


Figure 4.27: Directional tiling

4-D climate data into a GIS a query might always slice at the lowest atmospheric layer and at the most current time available without additional trimming in the horizontal axes.

We call such axes *preferred access directions* in the context of a directional tiling; they are identified by empty partitions. To accommodate this intention expressed by the user the sub-tiling strategy changes: no longer is regular tiling applied, which would introduce undesirable cuts along the preferred axis, but rather are subdividing hyperplanes constructed parallel to the preference axis. This allows accommodating the tile size maximum while, at the same time, keeping the number of tiles accessed in preference direction at a minimum.

In Figure 4.28, a 3-D cube is first split by way of directional tiling (left). One tile is larger than the maximum allowed, hence sub-tiling starts (center). It recognizes that axes 0 and 2 are preferred and, hence, splits only along dimension 1. The result (right) is such that subsetting along the preferred axes - i.e., with a trim or slice specification only in dimension 1 - can always be accommodated with a single tile read.

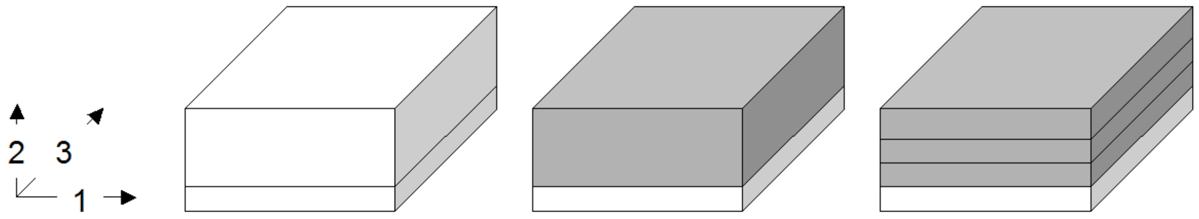


Figure 4.28: Directional tiling of a 3-D cube with one degree of freedom

Syntax

```
tiling directional splitList
( with subtiling ( tile size integerLit)? )?
```

where *splitList* is a list of split vectors ($t_{1,1}$; . . . ; $t_{1,n1}$),. . . ,($t_{d,1}$; . . . ; $t_{d,nd}$). Each split vector consists of an ascendingly ordered list of split points for the tiling algorithm, or an asterisk “*” for a preferred axis. The split vectors are positional, applying to the dimension axes of the array in order of appearance.

Example

The following defines a directional tiling with split vectors (0; 512; 1024) and (0; 15; 200) for axes 0 and 2, respectively, with dimension 1 as a preferred axis:

```
tiling directional [0,512,1024], [], [0,15,200]
```

4.19.6 Area of Interest Tiling

Concept

An *area of interest* is a frequently accessed sub-array of an array object. An area-of-interest pattern, consequently, consists of a set of domains accessed with an access probability significantly higher than that of all other possible patterns. Goal is to achieve a tiling which optimizes access to these preferred patterns; performance of all other patterns is ignored.

These areas of interest do not have to fully cover the array, and the may overlap. The system will establish an optimal disjoint partitioning for the given boxes in a way that the amount of data and the number of tiles accessed for retrieval of any area of interest are minimized. More exactly, it is guaranteed that accessing an area of interest only reads data belonging to this area.

Figure 4.29 gives an intuition of how the algorithm works. Given some area-of-interest set (a), the algorithm first partitions using directional tiling based on the partition boundaries (b). By construction, each of the resulting tiles (c) contains only cells which all share the same areas of interest, or none at all. As this introduces fragmentation, a merge step follows where adjacent partitions overlapping with the same areas of interest are combined. Often there is more than one choice to perform merging; the algorithm is inherently nondeterministic. Rasdaman exploits this degree of freedom and cluster tiles in sequence of dimensions, as this represents the sequentialization pattern on disk and, hence, is the best choice for maintaining spatial clustering on disk (d,e). In a final step, sub-tiling is performed on the partitions as necessary, depending on the tile size limit. In contrast to the directional tiling algorithm, an aligned tiling strategy is pursued here making use of the tile configuration argument, tc. As this does not change anything in our example, the final result (f) is unchanged over (e).

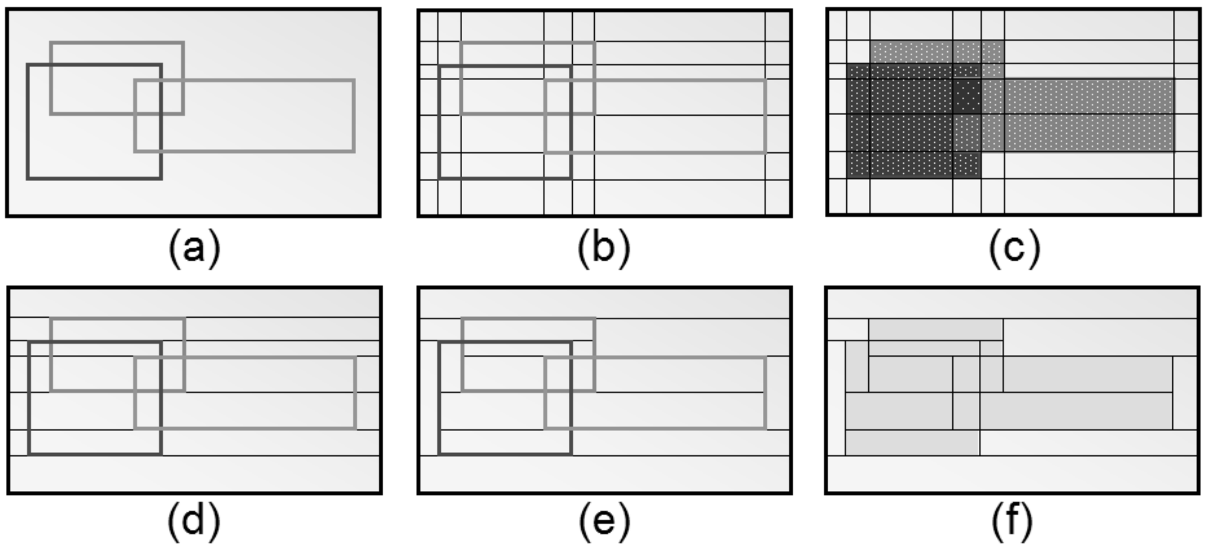


Figure 4.29: Steps in performing area of interest tiling**

Syntax

```
tiling area of interest tileConf ( tile size integerLit )?
```

Example

```
tiling area of interest
[0:20,0:40],[945:980,980:985],[10:1000,10:1000]
```

4.19.7 Tiling statistic

Concept

Area of interest tiling requires enumeration of a set of clearly delineated areas. Sometimes, however, retrieval does not follow such a focused pattern set, but rather shows some random behavior oscillating around hot spots. This can occur, for example, when using a pointing device in a Web GIS: while many users possibly want to see some "hot" area, coordinates submitted will differ to some extent. We call such a pattern multiple accesses to areas of interest. Area of interest tiling can lead to significant disadvantages in such a situation. If the actual request box is contained in some area of interest then the corresponding tiles will have to be pruned from pixels outside the request box; this requires a selective copying which is significantly slower than a simple memcpy(). More important, however, is a request box going slightly over the boundaries of the area of interest - in this case, an additional tile has to be read from which only a small portion will be actually used. Disastrous, finally, is the output of the area-of-interest tiling, as an immense number of tiny tiles will be generated for all the slight area variations, leading to costly merging during requests.

This motivates a tiling strategy which accounts for statistically blurred access patterns. The statistic tiling algorithm receives a list of access patterns plus border and frequency thresholds. The algorithm condenses this list into a smallish set of patterns by grouping them according to similarity. This process is guarded by the two thresholds. The border threshold determines from what maximum difference on two areas are considered separately. It is measured in number of cells to make it independent from area geometry. The result is a reduced set of areas, each associated with a frequency of occurrence. In a second run, those areas are filtered out which fall below the frequency threshold. Having calculated such representative areas, the algorithm performs an area of interest tiling on these.

This method has the potential of reducing overall access costs provided thresholds are placed wisely. Log analysis tools can provide estimates for guidance. In the storage directive, statistical tiling receives a list of areas plus, optionally, the two thresholds and a tile size limit.

Syntax

```
tiling statistic tileConf
( tile size integerLit )?
( border threshold integerLit)?
( interest threshold floatLit)?
```

Example

The following example specifies two areas, a border threshold of 50 and an interest probability threshold of 30%:

```
tiling statistic [0:20,0:40],[30:50,70:90]
border threshold 50
interest threshold 0.3
```

4.19.8 Summary: Tiling Guidelines

This section summarizes rules of thumb for a good tiling. However, a thorough evaluation of the query access pattern, either empirically through server log inspection or theoretically by considering application logics, is strongly recommended, as it typically offers a potential for substantial improvements over the standard heuristics.

- **Nothing is known about access patterns:** choose regular tiling with a maximum tile size; on PC-type architectures, tile sizes of about 4-5 MB have yielded good results.
- **Trim intervals in direction x are n times more frequent than in direction y and z together:** choose directional tiling where the ratios are approximately x*n=y*z. Specify a maximum tile size.
- **Hot spots (i.e., their bounding boxes) are known:** choose Area of Interest tiling on these bounding boxes.

4.20 Web Access to rasql

4.20.1 Overview

As part of petascope, the geo service frontend to rasdaman, Web access to rasql is provided. The request format is described in *Request Format*, the response format in *Response Format* below.

4.20.2 Service Endpoint

The service endpoint for rasql queries is

```
http://{service}/{path}/rasdaman/rasql
```

4.20.3 Request Format

A request is sent as an http GET URL with the query as key-value pair parameter. By default, the rasdaman login is taken from the petascope settings in `petascope.properties`; optionally, another valid rasdaman user name and password can be provided as additional parameters.

Syntax

```
http://{service}/{path}/rasdaman/rasql?params
```

This servlet endpoint accepts KVP requests with the following parameters:

query=q where *q* is a valid rasql query, appropriately escaped as per http specification.

username=u where *u* is the user name for logging into rasdaman (optional, default: value of variable `rasdaman_user` in `petascope.properties`)

password=p where *p* is the password for logging into rasdaman (optional, default: value of variable `rasdaman_pass` in `petascope.properties`)

Example

The following URL sends a query request to a fictitious server www.acme.com:

```
http://www.acme.com/rasdaman?
    query=select%20rgb.red+rgb.green%20from%20rgb
    &username=rasguest
    &password=rasguest
```

Since v10, this servlet endpoint can accept the credentials for `username:password` in basic authentication headers and `POST` protocol, for example using `curl` tool:

```
curl -u rasguest:rasguest
     -d 'query=select 1 + 15 from test_mr as c'
     'http://localhost:8080/rasdaman/rasql'
```

If results from rasql server are multiple objects (e.g: `SELECT .. FROM RAS_*` or a collection contains multiple arrays), then they are written in `multipart/related` MIME format with `End` string as **multipart boundary**. Below is an example from `SELECT c from RAS_COLLECTIONNAMES as c`:

```
--End
Content-type: text/plain

rgb
--End
Content-type: text/plain

mr
--End
Content-type: text/plain

mr2
--End--
```

Clients need to parse the **multipart** results for these cases. There are some useful libraries to do that, e.g. NodeJS with Mailparser.

4.20.4 Response Format

The response to a rasdaman query gets wrapped into a http message. The response format is as follows, depending on the nature of the result:

If the query returns arrays, then the MIME type of the response is `application/octet-stream`.

- If the result is empty, the document will be empty.
- If the result consists of one array object, then this object will be delivered as is.
- If the result consists of several array objects, then the response will consist of a Multipart/MIME document.
- If the query returns scalars, all scalars will be delivered in one document of MIME type `text/plain`, separated by whitespace.

4.20.5 Security

User and password are expected in cleartext, so do not use this tool in security sensitive contexts.

The service endpoint `rasdaman/rasql`, being part of the petascope servlet, can be disabled in the servlet container's setup (such as Tomcat).

4.20.6 Limitations

Currently, no uploading of data to the server is supported. Hence, functionality is restricted to queries without positional parameters `$1`, `$2`, etc.

Currently, array responses returned invariably have the same MIME type, `application/octet-stream`. In future it is foreseen to adjust the MIME type to the identifier of the specific file format as chosen in the `encode()` function.

4.21 Appendix A: rasql Grammar

This appendix presents a simplified list of the main rasql grammar rules used in the rasdaman system. The grammar is described as a set of production rules. Each rule consists of a non-terminal on the left-hand side of the colon operator and a list of symbol names on the right-hand side. The vertical bar `|` introduces a rule with the same left-hand side as the previous one. It is usually read as *or*. Symbol names can either be non-terminals or terminals (the former ones printed in bold face as a link which can be followed to the non-terminal production). Terminals represent keywords of the language, or identifiers, or number literals; "`(`", "`)`", "`[`", and "`]`" are also terminals, but they are in double quotes to distinguish them from the grammar parentheses (used to group alternatives) or brackets (used to indicate optional parts).

```
query                ::=  createExp
                        | dropExp
                        | selectExp
                        | updateExp
                        | insertExp
                        | deleteExp
createExp            ::=  createCollExp
                        | createStructTypeExp
                        | createMarrayTypeExp
                        | createSetTypeExp
createCollExp        ::=  create collection
                          namedCollection typeName
createCellTypeExp    ::=  create type typeName
                          a" cellTypeExp
cellTypeExp          ::=  "(" attributeName typeName
                          [ , attributeName typeName ]... ")"
createMarrayTypeExp  ::=  create type typeName
                          as "(" cellTypeExp | typeName ")"
                          mdarray domainSpec
domainSpec           ::=  "[" extentExpList "]"
extentExpList        ::=  extentExp [ , extentExpList ]
extentExp            ::=  axisName
                          [ "(" integerLit | intervalExp ")" ]
boundSpec            ::=  integerExp
createSetTypeExp     ::=  create type typeName
                          as set "(" typeName ")"
                          "[" nullExp "]"
nullExp              ::=  null values mintervalExp
dropExp              ::=  drop collection namedCollection
                        | drop type typeName
```

```
selectExp          ::=  select resultList
                        from collectionList
                        [ where generalExp ]
updateExp          ::=  update iteratedCollection
                        set updateSpec
                        assign generalExp
                        [ where generalExp ]
insertExp          ::=  insert into namedCollection
                        values generalExp
                        [ tiling [ StorageDirectives ] ]
StorageDirectives  ::=  RegularT | AlignedT | DirT
                      | AoiT | StatT
RegularT           ::=  regular TileConf
                        [ tile size integerLit ]
```

```
AlignedT          ::=  aligned TileConf [ TileSize ]
DirT              ::=  directional SplitList
                       [ with subtiling [ TileSize ] ]
AoiT              ::=  area of interest BboxList
                       [ TileSize ]
StatT             ::=  statistic TileConf [ TileSize ]
                       [ border threshold integerLit ]
                       [ interest threshold floatLit ]
TileSize          ::=  tile size integerLit
TileConf          ::=  BboxList [ , BboxList ]...
BboxList          ::=  "[" integerLit :  integerLit
                       [ , integerLit :  integerLit ]... "]"
Index             ::=  index IndexName
deleteExp         ::=  delete from iteratedCollection
                       [ where generalExp ]
updateSpec        ::=  variable [ mintervalExp ]
resultList        ::=  [ resultList , ] generalExp
```

```
generalExp         ::=  mddExp
                       | trimExp
                       | reduceExp
                       | inductionExp
                       | caseExp
                       | functionExp
                       | integerExp
                       | condenseExp
                       | variable
                       | mintervalExp
                       | intervalExp
                       | generalLit
mintervalExp       ::=  "[" spatialOpList "]"
                       | sdom "(" collIterator ")"
intervalExp        ::=  ( integerExp | * ) :
                        ( integerExp | * )
integerExp         ::=  integerTerm + integerExp
                       | integerTerm - integerExp
                       | integerTerm
integerTerm        ::=  integerFactor * integerTerm
                       | integerFactor / integerTerm
                       | integerFactor
integerFactor      ::=  integerLit
                       | identifier [ structSelection ]
                       | mintervalExp . lo
                       | mintervalExp . hi
                       | "(" integerExp ")"
spatialOpList      ::=  spatialOpList2
```

```
spatialOpList2      ::=  spatialOpList2 , spatialOp
                        | spatialOp
spatialOp           ::=  generalExp
condenseExp         ::=  condense condenseOpLit
                         over condenseVariable in generalExp
                         [ where generalExp ]
                         using generalExp
condenseOpLit       ::=  + | * | and | or | max | min
functionExp         ::=  version "(" ")"
                        | unaryFun "(" collIterator ")"
                        | binaryFun
                         "(" generalExp , generalExp ")"
                        | transcodeExp
unaryFun            ::=  oid | dbinfo
binaryFun           ::=  shift | scale | bit | pow | power | div | mod
transcodeExp        ::=  encode "(" generalExp , StringLit
                         [ , StringLit ] ")"
                        | decode "(" $ integerLit
                         [ , StringLit
                         , StringLit ] ")"
                        | decode "(" generalExp ")"
structSelection     ::=  . ( attributeName | integerLitExp )
inductionExp        ::=  unaryInductionOp "(" generalExp ")"
                        | generalExp . ( re | im )
                        | generalExp structSelection
                        | not generalExp
                        | generalExp binaryInductionOp generalExp
                        | ( + | - ) generalExp
                        | "(" castType ")" generalExp
                        | "(" generalExp ")"
unaryInductionOp    ::=  sqrt | abs | exp | log | ln
                        | sin | cos | tan | sinh | cosh
                        | tanh | arcsin | arccos | arctan
binaryInductionOp   ::=  overlay | is | = | and | or
                        | xor | plus | minus | mult
                        | div| equal | < | > | <=
                        | >= | !=
castType            ::=  bool | char | octet | short
                        | long | ulong | float | double
                        | ushort | unsigned ( short | long )
caseExp             ::=  case [ generalExp ] whenList
                         else generalExp end
whenList            ::=  [ whenList ]
                         when generalExp
                         then generalExp
collectionList      ::=  [ collectionList , ]
                         iteratedCollection
iteratedCollection  ::=  namedCollection
```

```
                             [ [ as ] collIterator ]
reduceExp              ::=   reduceIdent "(" generalExp ")"
reduceIdent            ::=   all_cells | some_cells | count_cells
                            | avg_cells | min_cells | max_cells
                            | add_cells
                            | stddev_samp | stddev_pop
                            | var_samp | var_pop
trimExp                ::=   generalExp mintervalExp
mddExp                 ::=   marray ivList values generalExp
ivList                 ::=   [ ivList , ]
                             marrayVariable in generalExp
generalLit             ::=   scalarLit | mddLit | StringLit | oidLit
oidLit                 ::=   < StringLit >
mddLit                 ::=   < mintervalExp dimensionLitList >
                            | $ integerLit
dimensionLitList       ::=   [ dimensionLitList ; ] scalarLitList
scalarLitList          ::=   [ scalarLitList , ] scalarLit
scalarLit              ::=   complexLit | atomicLit
complexLit             ::=   [ struct ] { scalarLitList }
atomicLit              ::=   booleanLit | integerLit | floatLit
                            | complex "(" floatLit , floatLit ")"
                            | complex "(" integerLit , integerLit ")"
typeName               ::=   identifier
variable               ::=   identifier
namedCollection        ::=   identifier
collIterator           ::=   identifier
attributeName          ::=   identifier
marrayVariable         ::=   identifier
condenseVariable       ::=   identifier
identifier             ::=   [a-zA-Z_] [a-zA-Z0-9_]*
```

4.22 Appendix B: Reserved keywords

This appendix presents the list of all tokens that CANNOT be used as variable names in rasql.

CHAPTER

FIVE

GEO SERVICES GUIDE

This guide covers the details of geo data management and retrieval in rasdaman. The rasdaman Array DBMS is domain agnostic; the specific semantics of space and time is provided through a layer on top of rasdaman, historically known as *petascope*. It offers spatio-temporal access and analytics through APIs based on the OGC data standard *Coverage Implementation Schema* (CIS) and the OGC service standards *Web Map Service* (WMS), *Web Coverage Service* (WCS), and *Web Coverage Processing Service* (WCPS).

Note: While the name petascope addresses a specific component we frequently use the name *rasdaman* to refer to the complete system, including petascope.

5.1 OGC Coverage Standards Overview

For operating rasdaman geo services as well as for accessing such geo services through these APIs it is important to understand the mechanics of the relevant standards. In particular, the concept of OGC / ISO *coverages* is important.

In standardization, coverages are used to represent space/time varying phenomena, concretely: regular and irregular grids, point clouds, and general meshes. The coverage standards offer data and service models for dealing with those. In rasdaman, focus is on multi-dimensional gridded ("raster") coverages.

In rasdaman, the OGC standards WMS, WCS, and WCPS are supported, being reference implementation for WCS. These APIs serve different purposes:

- WMS delivers a 2D map as a visual image, suitable for consunmption by humans
- WCS delivers n-D data, suitable for further processing and analysis
- WCPS performs flexible server-side processing, filtering, analytics, and fusion on coverages.

These coverage data and service concepts are summarized briefly below. Ample material is also available on the Web for familiarization with coverages (best consult in this sequence):

- hands-on demos for multi-dimensional coverage services provided by Jacobs University;
- a series of webinars and tutorial slides provided by EarthServer;

- a range of background information on these standards provided by OGC;
- the official standards documents maintained by OGC:
- WCS 2.0.1
- WCS-T 2.0
- WCPS 1.0
- WMS 1.3.0

5.1.1 Coverage Data

OGC CIS specifies an interoperable, conformance-testable coverage structure independent from any particular format encoding. Encodings are defined in OGC in GML, JSON, RDF, as well as a series of binary formats including GeoTIFF, netCDF, JPEG2000, and GRIB2).

By separating the data definition (CIS) from the service definition (WCS) it is possible for coverages to be served throuigh a variety of APIs, such as WMS, WPS, and SOS. However, WCS and WCPS have coverage-specific functionality making them particularly suitable for flexible coverage acess, analytics, and fusion.

5.1.2 Coverage Services

OGC WMS delivers 2D maps generated from styled layers stacked up. As such, WMS is a visualization service sitting at the end of processing pipelines, geared towards human consumption.

OGC WCS, on the other hand, provides data suitable for further processing (including visualization); as such, it is suitable also for machine-to-machine communication as it appears in the middle of longer processing pipelines. WCS is a modular suite of service functionality on coverages. WCS Core defines download of coverages and parts thereof, through *subsetting* directives, as well as delivery in some output format requested by the client. A set of WCS Extensions adds further functionality facets.

One of those is WCS Processing; it defines the `ProcessCoverages` request which allows sending a coverage analytics request through the WCPS spatio-temporal analytics language. WCPS supports extraction, analytics, and fusion of multi-dimensional coverage expressed in a high-level, declarative, and safe language.

5.2 OGC Web Services Endpoint

Once the petascope geo service is deployed (see *rasdaman installation guide*) coverages can be accessed through the HTTP service endpoint `/rasdaman/ows`.

For example, assuming that the service IP address is `123.456.789.1` and the service port is `8080`, the following request URLs would deliver the Capabilities documents for OGC WMS and WCS, respectively:

```
http://123.456.789.1:8080/rasdaman/ows?SERVICE=WMS&
↪REQUEST=GetCapabilities&VERSION=1.3.0
http://123.456.789.1:8080/rasdaman/ows?SERVICE=WCS&
↪REQUEST=GetCapabilities&VERSION=2.0.1
```

5.3 OGC Coverage Implementation Schema (CIS)

A coverage consists mainly of:

- *domain set*: provides information about where data sit in space and time. All coordinates expressed there are relative to the coverage's Coordinate Reference System or *Native CRS*. Both CRS and its axes, units of measure, etc. are indiciated in the domain set. Petascope currently supports grid topologies whose axes are aligned with the axes of the CRS. Along these axes, grid lines can be spaced regularly or irregularly.
- *range set*: the "pixel payload", ie: the values (which can be atomic, like in a DEM, or records of values, like in hyperspectral imagery).
- *range type*: the semantics of the range values, given by type information, null values, accuracy, etc.
- *metadata*: a black bag which can contain any data: the coverage will not understand these, but duly transport them along so that the connection between data and metadata is not lost.

Further components include `Envelope` which gives a rough, simplified overview on the coverage's location in space and time and `CoverageFunction` which is unused by any implementation known to us.

5.3.1 Coverage CRS

Every coverage, as per OGC CIS, must have exactly one *native* Coordinate Reference System (CRS), which is given by a URL. Resolving this URL should deliver the CRS definition. The OGC CRS resolver is an example of a public service for resolving CRS URLs; the same service is also bundled in every rasdaman installation, so that it is avialable locally. More details on this topic can be found in the *CRS Management* chapter.

Sometimes definitions for CRSs are readily available, such as the 2-D WGS84 with code EPSG:4326 in the EPSG registry. In particular spatio-temporal CRSs, however, are not always readily available, at least not in all combinations of spatial and temporal axes. To this end, composition of CRS is supported so that the single Native CRS can be built from "ingredient" CRSs by concatenating them into a composite one. For instance, a time-series of WGS84 images would have the following Native CRS:

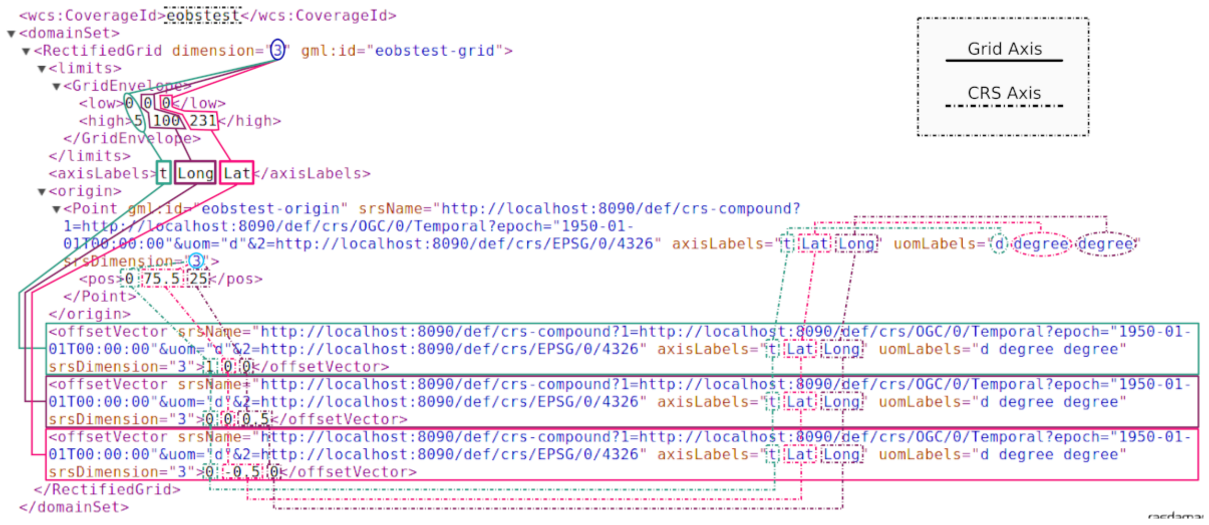
```
http://localhost:8080/def/crs-compound?
      1=http://localhost:8080/def/crs/OGC/0/AnsiDate
     &2=http://localhost:8080/def/crs/EPSG/0/4326
```

Coordinate tuples in this CRS represent an ordered composition of a temporal coordinate expressed in ISO 8601 syntax, such as `2012-01-01T00:01:20Z`, followed by latitude and longitude coordinates, as per EPSG:4326.

The native CRS of a coverage domain set can be determined in severay ways:

- in a WCS `GetCapabilities` response, the `wcs:CoverageSummary/ows:BoundingBox@crs` attribute;
- in a WCS `DescribeCoverage` response, the `srsName` attribute in the `gml:domainSet` element; Furthermore, the `axisLabels` attribute contains the CRS axis names according to the CRS sequency, and the `uomLabels` attribute contains the units of measure for each corresponding axis.
- in WCPS, the function `crsSet(e)` returns the CRS of a coverage expression *e*;

The following graphics illustrates, on the example of an image timeseries, how dimension, CRS, and axis labels affect the domain set in a CIS 1.0 `RectifiedGridCoverage`.



```
  <wcs:CoverageId>eobstest</wcs:CoverageId>
▾<domainSet>
  ▾<RectifiedGrid dimension="3" gml:id="eobstest-grid">
    ▾<limits>
      ▾<GridEnvelope>
          <low>0 0 0</low>
          <high>5 100 231</high>
       </GridEnvelope>
     </limits>
     <axisLabels>t Long Lat</axisLabels>
    ▾<origin>
      ▾<Point gml:id="eobstest-origin" srsName="http://localhost:8090/def/crs-compound?
        1=http://localhost:8090/def/crs/OGC/0/Temporal?epoch="1950-01-
        01T00:00:00"&uom="d"&2=http://localhost:8090/def/crs/EPSG/0/4326" axisLabels="t Lat Long" uomLabels="d degree degree"
        srsDimension="3">
          <pos>0 75.5 25</pos>
       </Point>
     </origin>
     <offsetVector srsName="http://localhost:8090/def/crs-compound?1=http://localhost:8090/def/crs/OGC/0/Temporal?epoch="1950-01-
     01T00:00:00"&uom="d"&2=http://localhost:8090/def/crs/EPSG/0/4326" axisLabels="t Lat Long" uomLabels="d degree degree"
     srsDimension="3">1 0 0</offsetVector>
     <offsetVector srsName="http://localhost:8090/def/crs-compound?1=http://localhost:8090/def/crs/OGC/0/Temporal?epoch="1950-01-
     01T00:00:00"&uom="d"&2=http://localhost:8090/def/crs/EPSG/0/4326" axisLabels="t Lat Long" uomLabels="d degree degree"
     srsDimension="3">0 0 0.5</offsetVector>
     <offsetVector srsName="http://localhost:8090/def/crs-compound?1=http://localhost:8090/def/crs/OGC/0/Temporal?epoch="1950-01-
     01T00:00:00"&uom="d"&2=http://localhost:8090/def/crs/EPSG/0/4326" axisLabels="t Lat Long" uomLabels="d degree degree"
     srsDimension="3">0 -0.5 0</offsetVector>
   </RectifiedGrid>
</domainSet>
```

Note: This handling of coordinates in CIS 1.0 bears some legacy burden from GML; in the `GeneralGridCoverage` introduced with CIS 1.1 coordinate handling is much simplified.

5.3.2 Range Type

Range values can be atomic or (possibly nested) records over atomic values, described by the range type. In rasdaman the following atomic data types are supported; all of these can be combined freely in records of values, such as in hyperspectral images or climate variables.

Table 5.1: Mapping of rasdaman base types to SWE Quantity types

rasdaman type	size	Quantity types
`boolean`	8 bit	unsignedByte
`octet`	8 bit	signedByte
`char`	8 bit	unsignedByte
`short`	16 bit	signedShort
`unsigned short` = `ushort`	16 bit	unsignedShort
`long`	32 bit	signedInt
`unsigned long` = `ulong`	32 bit	unsignedInt
`float`	32 bit	float32
`double`	64 bit	float64
`complex`	64 bit	cfloat32
`complexd`	128 bit	cfloat64

5.3.3 Nil Values

Nil (null) values, as per SWE, are supported by rasdaman in an extended way:

- null values can be defined over any data type
- nulls can be single values
- nulls can be intervals
- a null definnition in a coverage can be a list of all of the above alternatives.

Full details can be found in the null values *section*.

Note: It is highly recommended to NOT define **single** null values over floating-point data as this causes numerical problems well known in mathematics. This is not related to rasdaman, but intrinsic to the nature and handling of floating-point numbers in computers. **A floating-point interval** around the desired float null value should be preferred (this corresponds to interval arithmetics in numerical mathematics).

5.3.4 Errors

Errors from OGC requests to rasdaman are returned to the client formatted as `ows:ExceptionReport` (OGC Common Specification). An `ExceptionReport` can contain multiple `Exception` elements. For example, when running a WCS GetCoverage or a WCPS query which execute rasql queries in rasdaman, in case of an error the `ExceptionReport` will contain two `Exception` elements:

1. One with the error message returned from rasdaman.
2. Another with the rasql query that failed.

For example:

```
<ows:ExceptionReport>
    <ows:Exception exceptionCode="RasdamanRequestFailed">
        <ows:ExceptionText>The Encode function is applicable to␣
↪array arguments only.</ows:ExceptionText>
    </ows:Exception>
    <ows:Exception exceptionCode="RasdamanRequestFailed">
        <ows:ExceptionText>Failed internal rasql query: SELECT␣
↪encode(1, "png" ) FROM mean_summer_airtemp AS c</
↪ows:ExceptionText>
    </ows:Exception>
</ows:ExceptionReport>
```

5.4 OGC Web Coverage Service

WCS Core offers the following request types:

- `GetCapabilities` for obtaining a list of coverages offered together with an overall service description;
- `DescribeCoverage` for obtaining information about a coverage without downloading it;
- `GetCoverage` for downloading, extracting, and reformatting of coverages; this is the central workhorse of WCS.

WCS Extensions in part enhance `GetCoverage` with additional functionality controlled by further parameters, and in part establish new request types, such as:

- WCS-T defining `InsertCoverage`, `DeleteCoverage`, and `UpdateCoverage` requests;
- WCS Processing defining `ProcessCoverages` for submitting WCPS analytics code.

You can use `http://localhost:8080/rasdaman/ows` as service endpoints to which to send WCS requests, for example:

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1&
↪request=GetCapabilities
```

See example queries in the WCS systemtest which send KVP (key value pairs) GET request and XML POST request to Petascope.

5.4.1 Subsetting behavior

In general, subsetting in petascope behaves similarly to subsetting in gdal, with a couple of deviations necessary for n-D. Specifically, subsetting follows the next rules:

- Slicing (`geoPoint`): the grid slice with index corresponding to the requested slicing geo point is returned. This is computed as follows:

  ```
  gridIndex = floor((geoPoint - minGeoLowerBound) /␣
  ↪axisResolution)
  ```

- Trimming (`geoLowerBound:geoUpperBound`): the lower bound of the grid interval is determined as in the case of slicing. The number of returned grid points follows gdal:

  - If axis resolution is positive (e.g. `Long` axis):

    ```
    gridLowerBound = floor((geoLowerBound - minGeoLowerBound) /␣
    ↪axisResolution)
    numberOfGridPixels = floor(((geoUpperBound - geoLowerBound)␣
    ↪/ axisResolution) + 0.5)
    gridUpperBound = gridLowerBound + numberOfGridPixels - 1
    ```

  - If axis resolution is negative (e.g. `Lat` axis):

    ```
    gridLowerBound = floor((geoUpperBound - maxGeoLowerBound) /␣
    ↪axisResolution)
    numberOfGridPixels = floor((geoLowerBound - geoUpperBound) /
    ↪ axisResolution) + 0.5)
    gridUpperBound = gridLowerBound + numberOfGridPixels - 1
    ```

Note: If a trimming subset is applied on an axis with `(geoUpperBound - geoLowerBound) / axisResolution < 0.5`, then lower grid bound is translated by the slicing formula and upper grid bound is set to lower grid bound.

For example, a 2D coverage has `Long` (X) and `Lat` (Y) axes with CRS `EPSG:4326`. The resolution for axis `Long` is `10` and the resolution for axis `Lat` is `-10`. The geo bounds of axis `Long` are `[0:180]` and the geo bounds of axis `Lat` are `[0:90]`.

- Calculate *slicing* on `Long` axis by geo coordinates to grid coordinates:

```
- Long(0):          returns [0]
- Long(9):          returns [0]
- Long(10):         returns [1]
- Long(15):         returns [1]
- Long(20):         returns [2]
```

(continues on next page)

(continued from previous page)

```
- Long(40):         returns [4]
- Long(49.99999):   returns [4]
- Long(50.0):       returns [5]
```

- Calculate *trimming* on `Long` axis by geo coordinates to grid coordinates:

```
- Long(0:5):         returns [0:0]
- Long(0:10):        returns [0:0]
- Long(0:14.999):    returns [0:0]
- Long(0:15):        returns [0:1]
- Long(0:24.999):    returns [0:1]
- Long(0:25.0):      returns [0:2]
- Long(9,11): returns [0:0]
```

5.4.2 CIS 1.0 to 1.1 Transformation

Under WCS 2.1 - ie: with `SERVICE=2.1.0` - both `DescribeCoverage` and `GetCoverage` requests understand the proprietary parameter `OUTPUTTYPE=GeneralGridCoverage` which formats the result as CIS 1.1 `GeneralGridCoverage` even if it has been imported into the server as a CIS 1.0 coverage, for example:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&VERSION=2.1.0
    &REQUEST=DescribeCoverage
    &COVERAGEID=test_mean_summer_airtemp
    &OUTPUTTYPE=GeneralGridCoverage

http://localhost:8080/rasdaman/ows?SERVICE=WCS&VERSION=2.1.0
    &REQUEST=GetCoverage
    &COVERAGEID=test_mean_summer_airtemp
    &FORMAT=application/gml+xml
    &OUTPUTTYPE=GeneralGridCoverage
```

5.4.3 Polygon/Raster Clipping

WCS and WCPS support clipping of polygons expressed in the WKT format format. Polygons can be `MultiPolygon (2D)`, `Polygon (2D)` and `LineString (1D+)`. The result is always a 2D coverage in case of MultiPolygon and Polygon, and is a 1D coverage in case of `LineString`.

Further clipping patterns include `curtain` and `corridor` on 3D+ coverages from `Polygon (2D)` and `Linestring (1D)`. The result of `curtain` clipping has the same dimensionality as the input coverage whereas the result of `corridor` clipping is always a 3D coverage, with the first axis being the *trackline* of the corridor by convention.

In WCS, clipping is expressed by adding a `&CLIP=` parameter to the request. If the `SUBSETTINGCRS` parameter is specified then this CRS also applies to the clipping WKT, otherwise it is assumed that the WKT is in the Native coverage CRS. In WCPS, clipping is done with a `clip` function, much like in *rasql*.

Further information can be found in the *rasql clipping section*. Below we list examples illustrating the functionality in WCS and WCPS.

Clipping Examples

- Polygon clipping on coverage with Native CRS `EPSG:4326`, for example:
  - WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1&
  &REQUEST=GetCoverage
  &COVERAGEID=test_wms_4326
  &CLIP=POLYGON((-40.3130 144.8657, -40.8969 146.
↪3818, -40.7140 148.5352,
                  -43.2612 148.4253, -43.6122 146.
↪9531, -43.4689 145.6348,
                  -42.4721 145.0854, -41.4757 144.
↪778))
  &FORMAT=image/png
```

  - WCPS:

```
for $c in (test_wms_4326)
return
        encode(
                clip(
                      $c, POLYGON((-40.3130 144.8657,
↪ -40.8969 146.3818, -40.7140 148.5352,
                                   -43.2612 148.4253,
↪ -43.6122 146.9531, -43.4689 145.6348,
                                   -42.4721 145.0854,
↪ -41.4757 144.778))
                    )
                , "image/png"
              )
```

- Polygon clipping with coordinates in `EPSG:3857` (from `subsettingCRS` parameter) on coverage with Native CRS `EPSG:4326`, for example:
  - WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_wms_4326
  &CLIP=POLYGON((13589894.568 -2015496.69612,␣
↪15086830.0246 -1780682.3822))
```

(continues on next page)

(continued from previous page)

```
  &SUBSETTINGCRS=http://localhost:8080/rasdaman/def/
↪crs/EPSG/0/3857
  &FORMAT=image/png
```

– WCPS:

```
for $c in (test_wms_4326)
return
        encode(
                clip(
                      $c,
                      POLYGON((13589894.568 -2015496.
↪69612, 15086830.0246 -1780682.3822)),
                      "http://localhost:8080/def/crs/
↪EPSG/0/3857"
                    )
                , "image/png"
             )
```

- Linestring clipping on a 3D coverage with axes `X`, `Y`, `ansidate`, for example:

  – WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_irr_cube_2
  &CLIP=LineString("2008-01-01T02:01:20.000Z" 75042.
↪7273594 5094865.55794,
                   "2008-01-08T00:02:58.000Z" 705042.
↪727359 5454865.55794)
  &FORMAT=text/csv
```

  – WCPS:

```
for $c in (test_irr_cube_2)
return
       encode(
                clip(
                       $c,
                       LineString("2008-01-
↪01T02:01:20.000Z" 75042.7273594 5094865.55794,
                                  "2008-01-
↪08T00:02:58.000Z" 705042.727359 5454865.55794)
                    )
                    , "text/csv"
             )
```

- Multipolygon clipping on 2D coverage, for example:

  – WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_mean_summer_airtemp
  &CLIP=Multipolygon( ((-23.189600 118.432617, -27.
↪458321 117.421875,
                        -30.020354 126.562500, -24.
↪295789 125.244141)),
                      ((-27.380304 137.768555, -30.
↪967012 147.700195,
                        -25.491629 151.259766, -18.
↪050561 142.075195)) )
  &FORMAT=image/png
```

– WCPS:

```
for $c in (test_mean_summer_airtemp)
return
       encode(
                clip($c,
                         Multipolygon(
                           ((-23.189600 118.432617, -
↪27.458321 117.421875,
                            -30.020354 126.562500, -
↪24.295789 125.244141)),
                           ((-27.380304 137.768555, -
↪30.967012 147.700195,
                            -25.491629 151.259766, -
↪18.050561 142.075195))
                         )
                    )
                   , "image/png"
             )
```

- Curtain clipping by a Linestring on 3D coverage, for example:

  – WCS:

```
http://localhost:8080/rasdaman/ows?
↪SERVICE=WCSVERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_eobstest
  &CLIP=CURTAIN( projection(Lat, Long),
                 linestring(25 41, 30 41, 30 45, 30␣
↪42) )
  &FORMAT=text/csv
```

  – WCPS:

```
for $c in (test_eobstest)
```

(continues on next page)

(continued from previous page)

```
return
       encode(
                clip($c,
                         CURTAIN(
                                  projection(Lat,␣
↪Long),
                                  linestring(25 41,␣
↪30 41, 30 45, 30 42)
                                )
                    )
                  , "text/csv"
             )
```

- Curtain clipping by a Polygon on 3D coverage, for example:
  - WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_eobstest
  &CLIP=CURTAIN(projection(Lat, Long),
                Polygon((25 40, 30 40, 30 45, 30␣
↪42)))
  &FORMAT=text/csv
```

  - WCPS:

```
for $c in (test_eobstest)
return
       encode(
                clip($c,
                         CURTAIN(
                                  projection(Lat,␣
↪Long),
                                  Polygon((25 40, 30␣
↪40, 30 45, 30 42))
                                )
                    )
                  , "text/csv"
             )
```

- Corridor clipping by a Linestring on 3D coverage, for example:
  - WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_irr_cube_2
```

(continues on next page)

(continued from previous page)

```
  &CLIP=CORRIDOR( projection(E, N),
       LineString("2008-01-01T02:01:20.000Z" 75042.
↪7273594  5094865.55794,
                  "2008-01-01T02:01:20.000Z" 75042.
↪7273594 5194865.55794),
       LineString(75042.7273594 5094865.55794, 75042.
↪7273594 5094865.55794,
                  85042.7273594 5194865.55794, 95042.
↪7273594 5194865.55794)
      )
  &FORMAT=application/gml+xml
```

– WCPS:

```
for $c in (test_irr_cube_2)
return
       encode(
                clip($c,
                         CORRIDOR(
                                   projection(E, N),
                                   LineString("2008-
↪01-01T02:01:20.000Z" 75042.7273594  5094865.55794,
                                              "2008-
↪01-01T02:01:20.000Z" 75042.7273594 5194865.55794),
                                   LineString(75042.
↪7273594 5094865.55794, 75042.7273594 5094865.55794,
                                              85042.
↪7273594 5194865.55794, 95042.7273594 5194865.55794)
                                 )
                    )
                   , "application/gml+xml"
              )
```

- Corridor clipping by a Polygon on 3D coverage, for example:

  – WCS:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&
↪VERSION=2.0.1
  &REQUEST=GetCoverage
  &COVERAGEID=test_eobstest
  &CLIP=corridor( projection(Lat, Long),
                  LineString(26 41 "1950-01-01", 28␣
↪41 "1950-01-02"),
                  Polygon((25 40, 30 40, 30 45, 25␣
↪45)), discrete )
  &FORMAT=application/gml+xml
```

  – WCPS:

```
for $c in (test_eobstest)
return
       encode(
              clip($c,
                     CORRIDOR(
                              projection(Lat,␣
↪Long),
                              LineString(26 41
↪"1950-01-01", 28 41 "1950-01-02"),
                              Polygon((25 40,␣
↪30 40, 30 45, 25 45))
                              , discrete
                           )
                    )
                 , "application/gml+xml"
           )
```

Note: *Subspace* clipping is not supported in WCS or WCPS.

5.4.4 WCS-T

Currently, WCS-T supports importing coverages in GML format. The metadata of the coverage is thus explicitly specified, while the raw cell values can be stored either explicitly in the GML body, or in an external file linked in the GML body, as shown in the examples below. The format of the file storing the cell values must be

- 2-D data supported by the GDAL library, such as TIFF / GeoTIFF, JPEG / JPEG2000, PNG, etc;
- n-D data in NetCDF or GRIB format

In addition to the WCS-T standard parameters petascope supports additional proprietary parameters, covered in the following sections.

Note: For coverage management normally WCS-T is not used directly. Rather, the more convenient `wcst_import` Python tool is recommended for *Data Import*.

Inserting coverages

Inserting a new coverage into the server's WCS offerings is done using the `InsertCoverage` request.

Table 5.2: WCS-T Standard Parameters

Request Parameter	Value	Description	Required
SERVICE	WCS	service standard	Yes
VERSION	2.0.1 or later	WCS version used	Yes
REQUEST	Insert-Coverage	Request type to be performed	Yes
INPUT-COVER-AGEREF	{url}	URl pointing to the coverage to be inserted	One of inputCoverageRef or inputCoverage is required
INPUT-COVER-AGE	{coverage}	A coverage to be inserted	One of inputCoverageRef or inputCoverage is required
USEID	new | existing	Indicates wheter to use the coverage's id ("existing") or to generate a new unique one ("new")	No (default: existing)

Table 5.3: WCS-T Proprietary Enhancements

Request Parameter	Value	Description	Required
PIXEL-DATATYPE	GDAL supported base data type (eg: "Float32") or comma-separated concatenated data types, (eg: "Float32,Int32,Float32")	In cases where range values are given in the GML body the datatype can be indicated through this parameter. Default: Byte.	No
TILING	rasdaman tiling clause, see wiki:Tiling	Indicates the array tiling to be applied during insertion	No

The response of a successful coverage request is the coverage id of the newly inserted coverage. For example: The coverage available at http://schemas.opengis.net/gmlcov/1.0/examples/exampleRectifiedGridCoverage-1.xml can be imported with the following request:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&VERSION=2.0.1
    &REQUEST=InsertCoverage
    &COVERAGEREF=http://schemas.opengis.net/gmlcov/1.0/examples/
↪exampleRectifiedGridCoverage-1.xml
```

(continues on next page)

(continued from previous page)

The following example shows how to insert a coverage stored on the server on which rasdaman runs. The cell values are stored in a TIFF file (attachment:myCov.gml), the coverage id is generated by the server and aligned tiling is used for the array storing the cell values:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&VERSION=2.0.1
    &REQUEST=InsertCoverage
    &COVERAGEREF=file:///etc/data/myCov.gml
    &USEID=new
    &TILING=aligned[0:500,0:500]
```

Updating Coverages

Updating an existing coverage into the server's WCS offerings is done using the `UpdateCoverage` request.

Table 5.4: WCS-T Standard Parameters

Request Parameter	Value	Description	Required
SERVICE	WCS	service standard	Yes
VERSION	2.0.1 or later	WCS version used	Yes
REQUEST	UpdateCoverage	Request type to be performed	Yes
COVERAGEID	{string}	Identifier of the coverage to be updated	Yes
INPUTCOVERAGEREF	{url}	URl pointing to the coverage to be inserted	One of inputCoverageRef or inputCoverage is required
INPUTCOVERAGE	{coverage}	A coverage to be updated	One of inputCoverageRef or inputCoverage is required
SUBSET	AxisLabel(geoLowerBound, geoUpperBound)	Trim or slice expression, one per updated coverage dimension	No

The following example shows how to update an existing coverage `test_mr_metadata` from a generated GML file by `wcst_import` tool:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&version=2.0.1
    &REQUEST=UpdateCoverage
    &COVRAGEID=test_mr_metadata
    &SUBSET=i(0,60)
    &subset=j(0,40)
    &INPUTCOVERAGEREF=file:///tmp/4514863c_55bb_462f_a4d9_
↪5a3143c0e467.gml
```

(continues on next page)

(continued from previous page)

```
```

Deleting Coverages

The `DeleteCoverage` request type serves to delete a coverage (consisting of the underlying rasdaman collection, the associated WMS layer (if exists) and the petascope metadata). For example: The coverage `test_mr` can be deleted as follows:

```
http://localhost:8080/rasdaman/ows?SERVICE=WCS&VERSION=2.0.1
    &REQUEST=DeleteCoverage
    &COVERAGEID=test_mr
```

5.4.5 Renaming a coverage

The `/rasdaman/admin/coverage/update` non-standard API allows to update a coverage id and the associated WMS layer if one exists (v10.0+). For example, the coverage `test_mr` can be renamed to `test_mr_new` as follows:

```
http://localhost:8080/rasdaman/admin/coverage/update
    ?COVERAGEID=test_mr
    &NEWCOVERAGEID=test_mr_new
```

5.4.6 Update coverage metadata

Coverage metadata can be updated through the interactive rasdaman WSClient on the *OGC WCS > Describe Coverage* tab, by selecting a text file (MIME type must be one of `text/xml`, `application/json`, or `text/plain`) containing the new metadata; Note that to be able to do this it is necessary to login first in the *Admin* tab.

The non-standard API for this feature is at `/rasdaman/admin/coverage/update` which operates through multipart/form-data POST requests. The request should contain 2 parts:

1. the `coverageId` to update, and
2. the path to a local text file to be uploaded to the server.

For example, the below request will update the metadata of coverage `test_mr_metadata` with the one in a local XML file at `/home/rasdaman/Downloads/test_metadata.xml` by using the `curl` tool:

```
curl --form-string "COVERAGEID=test_mr_metadata"
     -F "file=@/home/rasdaman/Downloads/test_metadata.xml"
     "http://localhost:8080/rasdaman/admin/coverage/update"
```

5.4.7 INSPIRE Coverages

The INSPIRE Download Service is an implementation of the Technical Guidance for the implementation of INSPIRE Download Services using Web Coverage Services (WCS) version 2.0+.

In order to achieve INSPIRE Download Service compliance, the following enhancements have been implemented in rasdaman for `WCS GetCapabilities` response.

- Under `<ows:OperationsMetadata>` there is a new section for INSPIRE metadata for the service. For example, the result below contains two INSPIRE coverages `cov_1` and `cov_2`.

```
<ows:OperationsMetadata>
    <ows:ExtendedCapabilities>
        <inspire_dls:ExtendedCapabilities>
            <inspire_common:MetadataUrl>
                <inspire_common:URL>https://inspire.
↪rasdaman.org/rasdaman/ows</inspire_common:URL>
                <inspire_common:MediaType>application/
↪vnd.iso.19139+xml</inspire_common:MediaType>
            </inspire_common:MetadataUrl>
            <inspire_common:SupportedLanguages>
                <inspire_common:DefaultLanguage>
                    <inspire_common:Language>eng</
↪inspire_common:Language>
                </inspire_common:DefaultLanguage>
            </inspire_common:SupportedLanguages>
            <inspire_common:ResponseLanguage>
                <inspire_common:Language>eng</inspire_
↪common:Language>
            </inspire_common:ResponseLanguage>
            <inspire_dls:SpatialDataSetIdentifier␣
↪metadataURL="https://inspire-geoportal.ec.europa.eu/
↪resources/INSPIRE-f670705f-f4e9-11e6-81e4-
↪52540023a883_20211012-160902/services/1/PullResults/
↪521-540/16.iso19139.xml">
                <inspire_common:Code>cov_1</inspire_
↪common:Code>
                <inspire_common:Namespace>https://
↪inspire.rasdaman.org/rasdaman/ows</inspire_
↪common:Namespace>
            </inspire_dls:SpatialDataSetIdentifier>
            <inspire_dls:SpatialDataSetIdentifier␣
↪metadataURL="https://sh.de/csw?record_id=SH_DEM">
                <inspire_common:Code>cov_2</inspire_
↪common:Code>
                <inspire_common:Namespace>https://
↪inspire.rasdaman.org/rasdaman/ows</inspire_
↪common:Namespace>
            </inspire_dls:SpatialDataSetIdentifier>
```

(continues on next page)

(continued from previous page)

```
        </inspire_dls:ExtendedCapabilities>
    </ows:ExtendedCapabilities>
</ows:OperationsMetadata>
```

- Service Metadata URL field (`<inspire_common:URL>`), a URL containing the location of the metadata associated with the WCS service which is configured by setting `inspire_common_url` in `petascope.properties`.
- Under `<inspire_common:SupportedLanguages>` section, the supported language is fixed to `eng` (English) only.
- A coverage is considered INSPIRE coverage, if it has a specific URL set by `metadataURL attribute`. All INSPIRE coverages is listed in the list of XML elements `<inspire_dls:SpatialDataSetIdentifier>`. The example above contains two INSPIRE coverages, each `<inspire_dls:SpatialDataSetIdentifier>` element containing an attribute metadataURL to provide more information about the coverages. The value for `<inspire_common:Namespace>` elements of each INSPIRE coverage is derived from the service endpoint.

Create an INSPIRE coverage

Controlling whether a local coverage is treated as an INSPIRE coverage can be done by:

- Manually sending a request to `/rasdaman/admin/inspire/metadata/update` with two mandatory parameters:
  - `COVERAGEID` - the coverage to be converted to an INSPIRE coverage
  - `METADATAURL` - a URL to an INSPIRE-compliant catalog entry for this coverage; if set to empty, i.e. `METADATAURL=` then the coverage is marked as non-INSPIRE coverage.

  For example, the coverage `test_inspire_metadata` can be marked as INSPIRE coverage as follows:

  ```
  curl --user rasadmin:rasadmin -X POST \
       --form-string 'COVERAGEID=test_inspire_metadata' \
       -F 'METADATAURL=https://inspire-geoportal.ec.europa.eu/16.
  ↪iso19139.xml' \
       'http://localhost:8080//rasdaman/admin/inspire/metadata/
  ↪update'
  ```

- Via `wcst_import.sh`, in an ingredients files with inspire section contains the settings for importing INSPIRE coverage:
  - `metadata_url` - If set to non-empty string, then the importing coverage will be marked as INSPIRE coverage. If set to empty string or omitted, then the coverage will be updated as non-INSPIRE coverage.

For example, the coverage `cov_3` will be imported as INSPIRE coverage with this configuration in the ingredients file:

```
{
     "config": {
        ...
     },
     "input": {
        "coverage_id": "cov_3",
         "paths": [
              "mean_summer_airtemp.tif"
         ],
         "inspire": {
             "metadata_url": "https://inspire-geoportal.
↪ec.europa.eu/resources/INSPIRE-f670705f-f4e9-11e6-
↪81e4-52540023a883_20211012-160902/services/1/
↪PullResults/521-540/16.iso19139.xml"
         }
    },
    ...
}
```

5.4.8 Check if a coverage exists

In v10+, rasdaman offers non-standard API to check if a coverage exists in a simpler and faster way than doing a GetCapabilities or a DescribeCoverage request. The result is a `true/false` string literal.

Example:

```
http://localhost:8080/rasdaman/admin/coverage/exist?coverageId=cov1
```

5.4.9 GetCapabilities response extensions

The WCS `GetCapabilities` response contains some rasdaman-specific extensions, as documented below.

- The `<ows:AdditionalParameters>` element of each coverage contains some information which can be useful to clients:
  - `sizeInBytes` - an estimated size (in bytes) of the coverage
  - `sizeInBytesWithPyramidLevels` - an estimated size (in bytes) of the base coverage plus sizes of its pyramid coverages; only available if this coverage has pyramid
  - `axisList` - the coverage axis labels in geo CRS order

  Example:

```
<ows:AdditionalParameters>
    <ows:AdditionalParameter>
        <ows:Name>sizeInBytes</ows:Name>
        <ows:Value>155</ows:Value>
    </ows:AdditionalParameter>
    <ows:AdditionalParameter>
        <ows:Name>sizeInBytesWithPyramidLevels</ows:Name>
        <ows:Value>1869</ows:Value>
    </ows:AdditionalParameter>
    <ows:AdditionalParameter>
        <ows:Name>axisList</ows:Name>
        <ows:Value>Lat,Long</ows:Value>
    </ows:AdditionalParameter>
</ows:AdditionalParameters>
```

5.4.10 GetCoverage request

Interpolation

There are two supported formats for interpolation parameter in WCS `GetCoverage` requests:

- Full URI, e.g. `http://www.opengis.net/def/interpolation/OGC/1.0/bilinear`
- Short hand format, e.g. `bilinear`

5.5 OGC Web Coverage Processing Service (WCPS)

The OGC Web Coverage Processing Service (WCPS) standard defines a protocol-independent language for the extraction, processing, analysis, and fusion of multi-dimensional gridded coverages, often called datacubes.

5.5.1 General

WCPS requests can be submitted in both abstract syntax (example) and in XML (example).

For example, using the WCS GET/KVP protocol binding, a WCPS request can be sent through the following `ProcessCoverages` request:

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1
    &request=ProcessCoverage&query=<wcps-query>
```

The following subsections list enhancements rasdaman offers *over* the OGC WCPS standard. A brief introduction to the WCPS language is given in the *WCPS cheatsheet*; further educational material is available on EarthServer.

5.5.2 Polygon/Raster Clipping

The non-standard `clip()` function enables clipping in WCPS. The signature is as follows:

```
clip( coverageExpression, wkt [, subsettingCrs ] )
```

where

- `coverageExpression` is an expression of result type coverage, e.g. `dem + 10`;
- `wkt` is a valid WKT (Well-Known Text) expression, e.g. `POLYGON((...))`, `LineString(...)`;
- `subsettingCrs` is an optional CRS URL in which the `wkt` coordinates are expressed, e.g. `"http://localhost:8080/rasdaman/def/crs/EPSG/0/4326"`.

Clipping Examples

- Polygon clipping with coordinates in `EPSG:4326` on coverage with native CRS `EPSG:3857`:

```
for c in (test_wms_3857)
return
  encode(
    clip( c,
          POLYGON((
            -17.8115 122.0801, -15.7923 135.5273,
            -24.8466 151.5234, -19.9733 137.4609,
            -33.1376 151.8750, -22.0245 135.6152,
            -37.5097 145.3711, -24.4471 133.0664,
            -34.7416 135.8789, -25.7207 130.6934,
            -31.8029 130.6934, -26.5855 128.7598,
            -32.6949 125.5078, -26.3525 126.5625,
            -35.0300 118.2129, -25.8790 124.2773,
            -30.6757 115.4004, -24.2870 122.3438,
            -27.1374 114.0820, -23.2413 120.5859,
            -22.3501 114.7852, -21.4531 118.5645
          )),
          "http://localhost:8080/rasdaman/def/crs/EPSG/0/4326"
    ),
    "image/png"
  )
```

- Linestring clipping on 3D coverage with axes `X`, `Y`, `datetime`.

```
for c in (test_irr_cube_2)
return
  encode(
    clip( c,
          LineStringZ(75042.7273594 5094865.55794 "2008-01-
↪01T02:01:20.000Z", 705042.727359 5454865.55794 "2008-01-
↪08T00:02:58.000Z")
```

(continues on next page)

(continued from previous page)

```
    ),
    "text/csv"
  )
```

- Linestring clipping on 2D coverage with axes `X`, `Y`.

```
for c in (test_mean_summer_airtemp)
return
  encode(
    clip( c, LineString(-29.3822 120.2783, -19.5184 144.4043) )␣
↪WITH COORDINATES,
    "text/csv"
  )
```

In this case with `WITH COORDINATES` extra parameter, the geo coordinates of the values on the linestring will be included as well in the result. The first two bands of the result holds the coordinates (by geo CRS order), and the remaining bands the original cell values. Example output for the above query:

```
.. code-block:: text
```

“-28.975 119.975 90”,”-28.975 120.475 84”,”-28.475 120.975 80”, …

- Multipolygon clipping on 2D coverage.

```
for c in (test_mean_summer_airtemp)
return
  encode(
    clip( c, Multipolygon(
                 (( -20.4270 131.6931, -28.4204 124.1895,
                    -27.9944 139.4604, -26.3919 129.0015 )),
                 (( -20.4270 131.6931, -19.9527 142.4268,
                    -27.9944 139.4604, -21.8819 140.5151 ))
               )
    ),
    "image/png"
  )
```

- Curtain clipping by a Linestring on 3D coverage

```
for c in (test_eobstest)
return
  encode(
    clip( c, CURTAIN(projection(Lat, Long), linestring(25 40,␣
↪30 40, 30 45, 30 42) ) ),
    "text/csv"
  )
```

- Curtain clipping by a Polygon on 3D coverage

```
for c in (test_eobstest)
return
  encode(
    clip( c, CURTAIN(projection(Lat, Long), Polygon((25 40, 30␣
↪40, 30 45, 30 42)) ) ),
    "text/csv"
  )
```

- Corridor clipping by a Linestring on 3D coverage

```
for c in (test_irr_cube_2)
return
  encode(
    clip( c,
          corridor(
            projection(E, N),
            LineString( 75042.7273594  5094865.55794 "2008-01-
↪01T02:01:20.000Z",
                        75042.7273594 5194865.55794 "2008-01-
↪01T02:01:20.000Z" ),
            Linestring( 75042.7273594 5094865.55794, 75042.
↪7273594 5094865.55794,
                        85042.7273594 5194865.55794, 95042.
↪7273594 5194865.55794 )
          )
    ),
    "application/gml+xml"
  )
```

- Corridor clipping by a Polygon on 3D coverage (geo CRS: `EPSG:4326`) with input geo coordinates in `EPSG:3857`.

```
for c in (test_eobstest)
return
  encode(
    clip( c,
          corridor(
            projection(Lat, Long),
            LineString(4566099.12252 2999080.94347 "1950-01-01",
                       4566099.12252 3248973.78965 "1950-01-02
↪"),
            Polygon((4452779.63173 2875744.62435, 4452779.63173␣
↪3503549.8435,
                     5009377.0857 3503549.8435, 5009377.0857␣
↪2875744.62435) )
          ),
          "http://localhost:8080/def/crs/EPSG/0/3857"
    ),
    "application/gml+xml"
  )
```

5.5.3 Auto-ratio for spatial scaling

The `scale()` function allows to specify the target extent of only one of the spatial horizontal axes, instead of requiring both. In such a case, the extent of the unspecified axis will be determined automatically while preserving the original ratio between the two spatial axes.

For example in the request below, the extent of `Lat` will be automatically set to a value that preserves the ratio in the output result:

```
for $c in (test_mean_summer_airtemp)
return
  encode( scale( $c, { Long:"CRS:1"(0:160) } ), "image/png" )
```

5.5.4 Non-scaled axes are optional

The `scale()` function will implicitly add the full domains of unspecified non-spatial axes of a given coverage, with the effect that they will *not* be scaled in the result. This deviates from the OGC WCPS standard, which requires all axes to be specified with target domains, even if the resolution of an axis should not be changed in the result.

In the example query below, a 3D coverage is scaled only spatially because only the spatial axes E and N are specified in the target scale intervals, while the `ansi` non-spatial axis is omitted.

```
for $c in (test_irr_cube_2)
return
 encode(
   scale(
     $c[ansi("2008-01-01T02:01:20":"2008-01-08T00:02:58")] ,
     { E:"CRS:1"(0:20), N:"CRS:1"(0:10) }
   )
 , "json" )
```

5.5.5 Extensions on domain functions

The domain interval can be extracted from a `domain` and `imageCrsDomain`. Both the interval - ie: `[lowerBound:upperBound]` - and lower as well as upper bound can be retrieved for each axis.

Syntax:

```
operator(.lo|.hi)?
```

with `.lo` or `.hi` returning the lower bound or upper bound of this interval.

Further, the third argument of the `domain()` operator, the CRS URL, is optional. If not specified, `domain()` will use the CRS of the selected axis (ie, the second argument) instead.

For example, the coverage `AvgLandTemp` has 3 dimensions with grid bounding box of `(0:184, 0:1799, 0:3599)`, and a geo bounding box of

`("2000-02-01:2015-06-01", -90:90, -180:180)`. The table below lists various expressions and their results:

Table 5.5: Non-standard domain operations

Expression	Result
`imageCrsdomain($c, Long)`	`(0:3599)`
`imageCrsdomain($c, Long).lo`	`0`
`imageCrsdomain($c, Long).hi`	`3599`
`domain($c, Long)`	`(-180:180)`
`domain($c, Long).lo`	`-180`
`domain($c, Long).hi`	`180`

5.5.6 LET clause

An optional `LET` clause allows binding alias variables to valid WCPS query sub-expressions; subsequently the alias variables can be used in the `return` clause instead of repeating the aliased sub-expressions.

The syntax in context of a full query is as follows:

```
FOR-CLAUSE
LET $variable := assignment [ , $variable := assignment ]
    ...
[ WHERE-CLAUSE ]
RETURN-CLAUSE
```

where

```
assignment ::= coverageExpression | [ dimensionalIntervalList ]
```

An example with the first case:

```
for $c in (test_mr)
let $a := $c[i(0:50), j(0:40)],
    $b := avg($c) * 2
return
  encode( scale( $c, { imageCrsDomain( $c ) } ) + $b, "image/png" )
```

The second case allows to conveniently specify domains which can then be readily used in subset expression, e.g:

```
for $c in (test_mr)
let $dom := [i(20), j(40)]
return
  encode( $c[ $dom ] + 10, "itext/json" )
```

5.5.7 min and max functions

Given two coverage expressions `A` and `B` (resulting in compatible coverages, i.e. same domains and types), `min(A, B)` and `max(A, B)` calculate a result coverage with the minimum / maximum for each pair of corresponding cell values of `A` and `B`.

For multiband coverages, bands in the operands must be pairwise compatible; comparison is done in lexicographic order with the first band being most significant and the last being least significant.

The result coverage value has the same domain and type as the input operands.

5.5.8 Positional parameters

Positional parameters allow to reference binary or string values in a WCPS query, which are specified in a POST request in addition to the WCPS query. Each positional parameter must be a positive integer prefixed by a `$`, e.g. `$1`, `$2`, etc.

The endpoint to send WCPS query by POST with extra values is:

```
/rasdaman/ows?SERVICE=WCS&VERSION=2.0.1&REQUEST=ProcessCoverages
```

with the mandatory parameter `query` and optional positional parameters `1`, `2`, etc. The value of a positional parameter can be either a **binary file data** or a **string value**.

Example

One can use the `curl` tool to send a WCPS request with **positional parameters** from the command line; it will read the contents of specified files automatically if they are prefixed with a `@`.

For example, to combine an existing coverage `$c` with two temporary coverages `$d` and `$e` provided by positional parameters `$1` and `$2` into a result encoded in `png` format (specified by positional parameter `$3`):

```
curl -s "http://localhost:8080/rasdaman/ows?SERVICE=WCS&VERSION=2.0.
↪1&REQUEST=ProcessCoverages" \
     --form-string 'query=for $c in (existing_coverage), $d in␣
↪(decode($1)), $e in (decode($2))
         return encode(($c + $d + $e)[Lat(0:90), Long(-180:180)], "
↪$3"))' \
     -F "1=@/home/rasdaman/file1.tiff" \
     -F "2=@/home/rasdaman/file2.tiff" \
     -F "3=png" > test.png
```

5.5.9 Decode Operator in WCPS

The non-standard `decode()` operator allows to combine existing coverages with temporary coverages created in memory from input files attached in the request body via POST.

Only 2D geo-referenced files readable by GDAL are supported. One way to check if a file `$f` is readable by GDAL is with `gdalinfo $f`. `netCDF/GRIB` files are not supported.

Syntax

The syntax is

```
decode(${positional_parameter})
```

where `${positional_parameter)` refers to files in the POST request. See the *previous section* for more details on positional parameters.

Example

See *example on positional parameters*.

5.5.10 Case Distinction

Conditional evaluation based on the cell values of a coverage is possible with the `switch` expression. Although the syntax is a little different, the semantics is very much compatible to the rasql `case` statement, so it is recommended to additionally have a look at its corresponding *documentation*.

Syntax

```
SWITCH
  CASE condExp RETURN resultExp
  [ CASE condExp RETURN resultExp ]*
  DEFAULT RETURN resultExpDefault
```

where `condExp` and `resultExp` are either scalar-valued or coverage-valued expressions.

Constraints

- All `condExp` must return either boolean values or boolean coverages
- All `resultExp` must return either scalar values, or coverages
- The domain of all condition expressions must be the same
- The domain of all result expressions must be the same (that means same extent, resolution/direct positions, crs)

Evaluation Rules

If the result expressions return scalar values, the returned scalar value on a branch is used in places where the condition expression on that branch evaluates to `True`. If the result expressions return coverages, the values of the returned coverage on a branch are copied in the result coverage in all places where the condition coverage on that branch contains pixels with value `True`.

The conditions of the statement are evaluated in a manner similar to the IF-THEN-ELSE statement in programming languages such as Java or C++. This implies that the conditions must be specified by order of generality, starting with the least general and ending with the default result, which is the most general one. A less general condition specified after a more general condition will be ignored, as the expression meeting the less general expression will have had already met the more general condition.

Furthermore, the following hold:

- `domainSet(result) = domainSet(condExp1)`
- `metadata(result) = metadata(condExp1)`
- `rangeType(result) = rangeType(resultExp1)`. In case resultExp1 is a scalar, the result range type is the range type describing the coverage containing the single pixel resultExp1.

Examples

```
switch
  case $c < 10 return {red: 0;   green: 0;   blue: 255}
  case $c < 20 return {red: 0;   green: 255; blue:   0}
  case $c < 30 return {red: 255; green: 0;   blue:   0}
  default      return {red: 0;   green: 0;   blue:   0}
```

The above example assigns blue to all pixels in the $c coverage having a value less than 10, green to the ones having values at least equal to 10, but less than 20, red to the ones having values at least equal to 20 but less than 30 and black to all other pixels.

```
switch
  case $c > 0 return log($c)
  default     return 0
```

The above example computes log of all positive values in $c, and assigns 0 to the remaining ones.

```
switch
  case $c < 10 return $c * {red: 0;   green: 0;   blue: 255}
  case $c < 20 return $c * {red: 0;   green: 255; blue: 0}
  case $c < 30 return $c * {red: 255; green: 0;   blue: 0}
  default      return      {red: 0;   green: 0;   blue: 0}
```

The above example assigns *blue: 255* multiplied by the original pixel value to all pixels in the $c coverage having a value less than 10, *green: 255* multiplied by the original pixel value to the ones having values at least equal to 10, but less than 20, *red: 255* multiplied by the original pixel value to the ones having values at least equal to 20 but less than 30 and black to all other pixels.

5.5.11 CIS 1.0 to CIS 1.1 encoding

For output format `application/gml+xml` WCPS supports delivery as CIS 1.1 `GeneralGridCoverage` by specifying an additional proprietary parameter `outputType` in the `encode()` function, e.g:

```
for c in (test_irr_cube_2)
return encode( c, "application/gml+xml",
                  "{\"outputType\":\"GeneralGridCoverage\"}" )
```

5.5.12 Query Parameter

For specifying the WCPS query in a request, in addition to the `query` parameter the non-standard `q` parameter is also supported. A request must contain only one `q` or `query` parameter.

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1
  &REQUEST=ProcessCoverage&q=<wcps-query>
```

5.5.13 Describe Operator in WCPS

The non-standard `describe()` function delivers a "coverage description" of a given coverage without the range set, in either GML or JSON.

Syntax

```
describe( coverageExpression, outputFormat [ , extraParameters ] )
```

where

- `outputFormat` is a string specifying the format encoding in which the result will be formatted. Formats are indicated through their MIME type identifier, just as in `encode()`. Formats supported:
  - `application/gml+xml` or `gml` for GML
  - `application/json` or `json` for JSON
- `extraParameters` is an optional string containing parameters for fine-tuning the output, just as in `encode()`. Options supported:
  - `"outputType=GeneralGridCoverage"` to return a CIS 1.1 General Grid Coverage structure

Semantics

A `describe()` operation returns a description of the coverage resulting from the coverage expression passed, consisting of domain set, range type, and metadata, but not the range set. As such, this operator is the WCPS equivalent to a WCS `DescribeCoverage` request, and the output adheres to the same WCS schema.

The coverage description generated will follow the coverage's type, so one of Rectified Grid Coverage (CIS 1.0), ReferenceableGridCoverage (CIS 1.0), or General Grid Coverage (CIS 1.0).

By default, the coverage will be provided as Rectified or Referenceable Grid Coverage (in accordance with its type); optionally, a General Grid Coverage can be generated instead through `"outputType=GeneralGridCoverage"`. As JSON is supported only from OGC CIS 1.1 onwards this format is only available (i) if the coverage is stored as a CIS 1.1 General Grid Coverage (currently not supported) or (ii) this output type is selected explicitly through an `extraParameter`.

Efficiency: The `describe()` operator normally does not materialize the complete coverage, but determines only the coverage description making this function very efficient. A full evaluation is only required if `coverageExpression` contains a `clip()` performing a curtain, corridor, or linestring operation.

Examples

- Determine coverage description as a CIS 1.0 Rectified Grid Coverage in GML, without evaluating the range set:

```
for $c in (Cov)
return describe( $c.red[Lat(10:20), Long(30:40), "application/
↪gml+xml" )
```

- Deliver coverage description as a CIS 1.1 General Grid Coverage in GML, where range type changes in the query:

```
for $c in (Cov)
return describe( { $c.red; $c.green; $c.blue }, "application/
↪gml+xml",

↪"outputType=GeneralGridCoverage" )
```

- Deliver coverage description as a CIS 1.1 General Grid Coverage, in JSON:

```
for $c in (Cov)
return describe( $c, "application/json",
↪"outputType=GeneralGridCoverage" )
```

Specific Exceptions

- Unsupported output format
- This format is only supported for General Grid Coverage
- Illegal extra parameter

5.5.14 Flip Operator in WCPS

The non-standard `FLIP` function enables reversing values from an axis belonging to a coverage expression. The output coverage expression has *no* changes in the grid domains, base type and dimensionality, but with reversed values and geo bounds of the selected axis; if this axis is irregular then its list coeffcients is reversed as well. See more *details* in rasql.

Syntax

```
flipExp: FLIP coverageExpression ALONG axisLabel

axisLabel: identifier
```

A `FLIP` expression consists of `coverageExpression` which denotes the input coverage, and one `axisLabel` of the coverage to flip values.

Examples

The following examples illustrate the syntax of the FLIP operator.

- Flipping the 2D coverage expression on its Long axis, by using:

```
for $c in (test_mean_summer_airtemp)
return
   encode(
           FLIP $c[Lat(-30:-15), Long(125:145)] ALONG Long
         , "image/png")
```

- Flipping the 3D coverage expression on its unix time axis, by using:

```
for $c in (test_wms_3d_time_series_irregular)
return
    encode(
           FLIP $c[Lat(40:90), Long(80:140)] + 20 ALONG unix
          , "json")
```

5.5.15 Sort Operator in WCPS

The SORT operator enables the user to sort a coverage expression along an axis. The sorting is done by slicing the array of the coverage along that axis, calculating a slice rank for each of the slices, and then rearranging the slices according to their ranks, in an ascending or descending order.

The sorting causes no change in the spatial domain, base type, or dimensionality. This means that the resulting array is the original array but with its values sorted at the sorting axis. See more *details* in rasql.

Note: After sorting, the geo domains (and coefficients for irregular axis) of the sorted axis are not changed, even though the grid values associated with geo coordinates are changed.

Syntax

```
sortExp: SORT coverageExp ALONG sortAxis [listingOrder] BY cellExp

coverageExp: a general coverage expression
sortAxis: identifier.
listingOrder: ASC (default if omitted) | DESC
cellExp: an expression that produces scalar ranks for each slice
along the sortAxis.
```

Note: One should not do subset (slice/trim) on the sortAxis in the cellExp

Examples

The following examples illustrate the syntax of the SORT operator.

- Sort the 2D coverage expression on its `Lon` axis according to the coverage values at each longitude index and -40 latitude in ascending order:

```
for $c in (test_mean_summer_airtemp)
return
   encode(
           SORT $c ALONG Lon BY $c[Lat(-40)]
         , "image/png")
```

- Sort the 3D coverage expression on its `unix` time axis in descending order by the sum of each time slice along it:

```
for $c in (test_wms_3d_time_series_irregular)
return
  encode(
     SORT $c.Red + 30 ALONG unix DESC BY add($c)
  , "json")
```

5.6 OGC Web Map Service (WMS)

The OGC Web Map Service (WMS) standard provides a simple HTTP interface for requesting overlays of geo-registered map images, ready for display.

With petascope, geo data can be served simultaneously via WMS, WMTS, WCS, and WCPS. Further information:

- *How to publish a WMS layer via WCST_Import*.
- *Add WMS style queries to existing layers*.

This section mainly covers rasdaman extensions of the OGC WMS standard.

5.6.1 GetMap extensions

Transparency

By adding a parameter `transparent=true` to WMS requests the returned image will have `NoData Value=0` in the metadata indicating to the client that all pixels with value *0* value should be considered transparent for PNG encoding format. Example:

```
http://localhost:8080/rasdaman/ows?SERVICE=WMS&VERSION=1.3.0
    &REQUEST=GetMap&LAYERS=waxlake1
    &BBOX=618887,3228196,690885,3300195.0
    &CRS=EPSG:32615&WIDTH=600&HEIGHT=600&FORMAT=image/png
    &TRANSPARENT=true
```

Interpolation

If in a `GetMap` request the output CRS requested is different from the coverage's native CRS, petascope will duly reproject the map applying resampling and interpolation. The algorithm used can be controlled with the non-standard `GetMap` parameter `interpolation=${method}`; default is nearest-neighbour interpolation. See *Geographic projection* for the methods available and their meaning. Example:

```
http://localhost:8080/rasdaman/ows?SERVICE=WMS
    &VERSION=1.3.0
    &REQUEST=GetMap
    &LAYERS=test_wms_3857
    &BBOX=-44.525,111.976,-8.978,156.274
    &CRS=EPSG:4326
    &WIDTH=60&HEIGHT=60
    &FORMAT=image/png
    &INTERPOLATION=bilinear
```

Random parameter

Normally, Web Browser cache the WMS requests from a WMS client (e.g. WebWorldWind). In order to bypass that, one needs to add append extra parameter `random` with its value equals to a random number for all WMS *GetMap* requests. For example:

```
http://localhost:8080/rasdaman/ows?SERVICE=WMS&VERSION=1.3.0
    &REQUEST=GetMap&LAYERS=waxlake1
    &BBOX=618887,3228196,690885,3300195.0
    &CRS=EPSG:32615&WIDTH=600&HEIGHT=600&FORMAT=image/png
    &TRANSPARENT=true
    &RANDOM=12345678
```

In petascope, this `random` parameter is stripped when petascope receives a WMS `GetMap` request, hence, if the request is already processed, the result stored in the cache will be returned as usual.

5.6.2 nD Coverages as WMS Layers

Petascope allows to import a 3D+ coverage as a WMS layer. To this end, the ingredients file used for `wcst_import` must contain `wms_import": true`. For 3D+ coverages this works with recipes *regular_time_series*, *irregular_time_series*, and *general_coverage*. This example shows how to define an *irregular_time_series* 3D coverage from 2D TIFF files.

Once the coverage is created, `GetMap` requests can use the additional (non-horizontal) axes for subsetting according to the OGC WMS 1.3.0 standard.

Table 5.6: WMS Subset Parameters for Different Axis Types

Axis Type	Subset parameter
Time	time=…
Elevation	elevation=…
Other	dim_AxisName=… (e.g dim_pressure=…)

According to the WMS 1.3.0 specification, the subset for non-geo-referenced axes can have these formats:

- Specific value (*value1*):

```
time='2012-01-01T00:01:20Z'
dim_pressure=20
```

- Range values (*min/max*):

```
time='2012-01-01T00:01:20Z'/'2013-01-01T00:01:20Z'
dim_pressure=20/30
```

- Multiple values (*value1,value2,value3,…*):

```
time='2012-01-01T00:01:20Z','2013-01-01T00:01:20Z'
dim_pressure=20,30,60,100
```

- Multiple range values (*min1/max1,min2/max2,…*):

```
dim_pressure=20/30,40/60
```

A `GetMap` request always returns a 2D result. If a non-geo-referenced axis is omitted from the request it will be considered as a slice on the upper bound along this axis. For example, in a time-series the most recent timeslice will be delivered.

Examples:

- Multiple values on time axis of a 3D coverage
- Multiple values on time and dim_pressure axes of a 4d coverage

5.6.3 GetLegendGraphic request

WMS `GetLegendGraphic` allows to get a legend PNG/JPEG image associated with a style of a layer. Admin can set a legend image for a style via a *style creation* request.

Required request parameters:

- `format` - data format in which the legend image is returned; only `image/png` and `image/jpeg` are supported.
- `layer` - the WMS layer which contains the specified style.
- `style` - the style which contains the legend image.

Note: Any further extra parameters will be ignored by rasdaman.

This request, for example, will return the legend image for style color of layer cov1:

```
http://localhost:8080/rasdaman/ows?service=WMS&
↪request=GetLegendGraphic
    &format=image/png&layer=cov1&style=color
```

When a style of a layer has an associated legend graphic, WMS `GetCapabilities` will have an additional `<LegendURL>` XML section for this style. For example:

```
<LegendURL>
    <Format>image/jpeg</Format>
    <OnlineResource
        xlink:href="http://localhost:8080/rasdaman/ows?service=WMS&
↪amp;request=GetLegendGraphic
           &amp;format=image/jpeg&amp;layer=cov_1&amp;style=NDVI"/>
</LegendURL>
```

5.6.4 Layer Management

Non-standard API for WMS layer management are listed below.

Layers can be easily created from existing WCS coverages in two ways:

- By enabling this during coverage import in the ingredients file with the *wms_import* option;
- By manually sending an */rasdaman/admin/layer/activate* HTTP request to petascope
- Create a new WMS layer from an existing coverage `MyCoverage`:

  ```
  /rasdaman/admin/layer/activate?COVERAGEID=MyCoverage
  ```

  During coverage import this can be done with the *wms_import* option in the ingredients file.

- Remove a WMS layer directly:

  ```
  /rasdaman/admin/layer/deactivate&COVERAGEID=MyLayer
  ```

  Indirectly a layer will be removed when *deleting the associated WCS coverage*

5.6.5 Style Behavior

When a client sends `GetMap` requests, the rules below define (in conformance with the WMS 1.3 standard) how a style is applied to the requested layers:

- If no styles are defined then rasdaman returns the data as-is, encoded in the requested format.
- If some styles are defined, e.g. X, Y, and Z, then:
  - If the client specifies a style Y, then Y is applied.
  - If the client does not specify a style, then:
    * If the admin has set a style as default, e.g. Z, then Z is applied.
    * Otherwise, if no style has been set as default, then the first style from the list of styles (X) is applied.

5.6.6 Style Management

Styles can be created for layers using rasql and WCPS query fragments. This allows users to define several visualization options for the same dataset in a flexible way. Examples of such options would be color classification, NDVI detection etc. The following HTTP request will create a style with the name, abstract and layer provided in the KVP parameters below

Note: Tomcat version 7+ requires the query (WCPS/rasql fragment) to be URL-encoded correctly. This site offers such an encoding service.

Style Definition

A style of a WMS layer can be created via the `/rasdaman/admin/layer/style/add` endpoint, while an existing style can be updated via the `/rasdaman/admin/layer/style/update` endpoint. Both endpoints understand the following parameters:

- `COVERAGEID` - an existing WMS layer to which the style to be created or updated belongs (mandatory);
- `STYLEID` - the style name, must be unique among all the styles of one layer (mandatory);
- `TITLE` - an optional style title as human-understandable text;
- `ABSTRACT` - an optional description of the what the style does
- One of the following (optional):
  - `RASQLTRANSFORMFRAGMENT` - a rasql query expression applied to the map tiles before being returned to the client;
  - `WCPSQUERYFRAGMENT` - a WCPS query expression applied to the map tiles before being returned to the client;

- `COLORTABLETYPE` + `COLORTABLEDEFINITION` - an optional color table for coloring the map tiles before returning to the client.

At least a query fragment, or a color table, or both, must be specified in the request.

Additionally the updating endpoint supports the following optional parameters:

- `NEWSTYLEID` - the style specified with `STYLEID` will be renamed to the new id specified by this parameter.
- `DEFAULT` - if set to `true` then this style is set as the default of the layer (more details *here*); if not specified, it is `false` by default.
- `LEGENDGRAPHIC` - associate a PNG/JPEG legend image to this style, specified in Base64 string format; clients can get the legend with a `GetLegendGraphic` request (more details *here*). The legend can be removed by setting this parameter to empty, i.e. `LEGENDGRAPHIC=`.

Below the supported values for `COLORTABLETYPE` are explained:

- `ColorMap`: check *Coloring Arrays* for more details; the color table definition must be a JSON object, for example:

```
{
  "type": "intervals",
  "colorTable": {  "0":   [0,     0,  255,   0],
                   "100": [125, 125,  125, 255],
                   "255": [255,   0,    0, 255]
                }
}
```

- `GDAL`: The color table definition must be a JSON object containing **256 color arrays** in a `colorTable` array, example:

```
{
   "colorTable": [
                  [255,   0,   0,255],
                  [216,  31,  30,255],
                  [216,  31,  30,255],
                  ...,
                  [ 43,131,186,255]
                 ]
}
```

- `SLD`: The color table definition must be valid Styled Layer Descriptor XML and contain a `ColorMap` element. Note that rasdaman will only consider the first `sld:ColorMap` element in the SLD document, any other SLD elements will be ignored. Check *Coloring Arrays* for details about the supported types (`ramp` (default), `values`, `intervals`), example `ColorMap` with `type="values"`:

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor xmlns="http://www.opengis.net/sld"
                       xmlns:gml="http://www.opengis.net/gml"
```

(continues on next page)

(continued from previous page)

```
                          xmlns:sld="http://www.opengis.net/sld"
                          xmlns:ogc="http://www.opengis.net/ogc"
                          version="1.0.0">
  <UserLayer>
    <sld:LayerFeatureConstraints>
      <sld:FeatureTypeConstraint/>
    </sld:LayerFeatureConstraints>
    <sld:UserStyle>
      <sld:Name>sqi_fig5_crop1</sld:Name>
      <sld:FeatureTypeStyle>
        <sld:Rule>
          <sld:RasterSymbolizer>
            <sld:ColorMap type="values">
               <ColorMapEntry color="#0000FF" quantity="150" />
               <ColorMapEntry color="#FFFF00" quantity="200" />
               <ColorMapEntry color="#FF0000" quantity="250" />
            </sld:ColorMap>
          </sld:RasterSymbolizer>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </UserLayer>
</StyledLayerDescriptor>
```

Style Removal

Removing a style from an existing WMS layer can be done via the `/rasdaman/admin/layer/style/remove` endpoint, e.g.

```
/rasdaman/admin/layer/style/remove?COVERAGEID=MyCoverage&
↪STYLEID=mystyle
```

Examples

- Create a style with a WCPS query fragment and set this style as default style:

```
http://localhost:8080/rasdaman/admin/layer/style/add
    ?COVERAGEID=test_wms_4326
    &STYLEID=wcps_style
    &ABSTRACT=This style marks the areas where fires are in␣
↪progress with the color red
    &WCPSQUERYFRAGMENT=switch case $c > 1000 return {red: 107;␣
↪green:17; blue:68} default return {red: 150; green:103;␣
↪blue:14})
    &DEFAULT=true
```

Variable `$c` will be replaced by a layer name when sending a `GetMap` request containing this layer's style.

- Create a style with a rasql query fragment:

```
http://localhost:8080/rasdaman/admin/layer/style/add
    ?COVERAGEID=test_wms_4326
    &STYLEID=rasql
    &ABSTRACT=This style marks the areas where fires are in␣
↪progress with the color red
    &RASQLTRANSFORMFRAGMENT=case $Iterator when ($Iterator + 2)␣
↪> 200 then {255, 0, 0} else {0, 255, 0} end
```

  Variable `$Iterator` will be replaced with the actual name of the rasdaman collection and the whole fragment will be integrated inside the regular `GetMap` request.

- Multiple layers can be used in a style definition. Besides the iterators `$c` in WCPS query fragments and `$Iterator` in rasql query fragments, which always refer to the current layer, other layers can be referenced by name using an iterator of the form `$LAYER_NAME` in the style expression.

  Example: create a WCPS query fragment style referencing 2 layers (`$c` refers to layer *sentinel2_B4* which defines the style):

```
http://localhost:8080/rasdaman/admin/layer/style/add
    ?COVERAGEID=sentinel2_B4
    &STYLEID=BandsCombined
    &ABSTRACT=This style needs 2 layers
    &WCPSQUERYFRAGMENT=$c + $sentinel2_B8
```

  Then, in any `GetMap` request using this style the result will be obtained from the combination of the 2 layers *sentinel2_B4* and *sentinel2_B8*:

```
http://localhost:8080/rasdaman/ows?SERViCE=WMS&VERSION=1.3.0
    &REQUEST=GetMap
    &LAYERS=sentinel2_B4
    &BBOX=-44.975,111.975,-8.975,155.975&CRS=EPSG:4326
    &WIDTH=800&HEIGHT=600
    &FORMAT=image/png&transparent=true
    &STYLES=BandsCombined
```

  The WCPS query fragment must follow one of these patterns in order to allow petascope to instantiate the fragment into a full valid query for any WMS request bbox:

  - If no subsets are in the style, just use `$c` and `$otherLayerName` as usual in the fragment query.
  - If there is a subset, usually a slice on non-XY axes for 3D+ coverages, then the subsets must follow the pattern `$c[axisLabel(geoSubset),..]` or `$otherLayerName[axisLabel(geoSubset),..]`.

- WMS styling supports colorizing the result of GetMap request when the style is requested by applying a color table definition to it. A style can contain either one or both a query fragment and color table definitions. The request supports two parameters for this purpose: `COLORTABLETYPE` with valid values `ColorMap`, `GDAL` and `SLD`, and `COLORTABLEDEFINITION` containing the corresponding definition.

```
http://localhost:8080/rasdaman/admin/layer/style/add
    ?COVERAGEID=test_wms_4326
    &STYLEID=firearea
    &ABSTRACT=This style marks the areas where fires are in␣
↪progress with the color red
    &WCPSQUERYFRAGMENT=switch case $c > 1000 return {red: 107;␣
↪green:17; blue:68} default return {red: 150; green:103;␣
↪blue:14})
    &COLORTABLETYPE=ColorMap
    &COLORTABLEDEFINITION={"type": "intervals", "colorTable": {␣
↪ "0": [0, 0, 255, 0], "100": [125, 125, 125, 255], "255":␣
↪[255, 0, 0, 255] } }
```

5.6.7 Pyramid Management

The following WMS requests are used to manage downscaled coverages, which are primarily created as pyramid *levels* of a particular *base* coverage. Internally they are used for efficient zooming in/out in WMS, and downscaling when using the `scale()` function in WCPS or scaling extension in WCS.

Only regular axes, typically spatial X and Y, can be downscaled for this purpose.

Below the API for pyramid management are covered:

- Create a pyramid member coverage *c* for a base coverage *b* with given scale factors for each axis. Only regular axes can have a *scale factor > 1*. E.g. to create a downscaled coverage *cov_3D_4* of a 3D coverage *cov_3D* that is *4x smaller* for Lat and Long regular axes (Time is irregular axis, hence, scale factor must be 1):

  ```
  http://localhost:8080/rasdaman/admin/coverage/pyramid/create
      ?COVERAGEID=cov_3D
      &MEMBER=cov_3D_4
      &SCALEVECTOR=1,4,4
  ```

  `wcst_import` can execute create pyramid requests automatically when importing data with the `scale_levels` or `scale_factors` options in the ingredients file; more details *here*.

- Add a list of existing coverage *c*, *d*, *e*, . . . as pyramid member coverages of a base coverage *b*. The scale factors for each axis of the pyramid member coverage will be calculated implicitly based on axis resolutions. If *harvesting=true* (default is false), recursively collect pyramid members of *c*, *d*, *e*, . . . and add them as pyramid member of *b*. E.g. to add a downscaled coverage *cov_3D_4* (4x smaller) and its pyramid members recursively as pyramid member coverages of base coverage *cov_3D*:

```
http://localhost:8080/rasdaman/admin/coverage/pyramid/add
    ?COVERAGEID=cov_3D
    &MEMBERS=cov_3D_4
    &HARVESTING=true
```

`wcst_import` provides *several options* for conveniently adding pyramid members in the ingredients file.

- Remove a list of existing pyramid member coverage *c*, *d*, *e*, . . . from a base coverage *b*. The coverages *c*, *d*, *e*, . . . will still exist, until they are removed with a WCS-T *DeleteCoverage request*. E.g. to remove pyramid member *cov_3D_4* from base coverage *cov_3D*:

```
http://localhost:8080/rasdaman/admin/coverage/pyramid/remove
    &COVERAGEID=cov_3D
    &MEMBERS=cov_3D_4
```

- List all pyramid member coverages associated with a base coverage in JSON-formatted output. E.g. to list the pyramid members of *Sentinel2_10m*:

```
http://localhost:8080/rasdaman/admin/coverage/pyramid/list
    ?COVERAGEID=Sentinel2_10m
```

Example output:

```
{
  "coverage": "Sentinel2_10m",
  "members": [
      {
        "coverage": "Sentinel2_20m",
        "scale": [ 1, 2, 2 ]
      },
      {
        "coverage": "Sentinel2_60m",
        "scale": [ 1, 6, 6 ]
      }
    ]
}
```

5.6.8 Testing a WMS Setup

A rasdaman WMS service can be tested with any conformant client through a `GetMap` request like the following:

```
http://example.org/rasdaman/ows?service=WMS&version=1.3.0&
↪request=GetMap
    &layers=MyLayer
    &bbox=618885.0,3228195.0,690885.0,3300195.0
```

(continues on next page)

(continued from previous page)

```
    &crs=EPSG:32615
    &width=600
    &height=600
    &format=image/png
```

5.6.9 Errors and Workarounds

Cannot load new WMS layer in QGIS

In this case, the problem is due to QGIS caching the WMS GetCapabilities from the last request so the new layer does not exist (see clear cache solution).

5.7 OGC Web Map Tile Service (WMTS)

The OGC Web Map Tile Service (WMTS) standard provides a simple HTTP interface for requesting overlays of geo-registered map images as small tiles, ready for display. WMTS works like a subset of OGC Web Map Service (WMS) standard and it provides extra functionalities from the imported WMS Layer. See more *details* for processing WMS requests in rasdaman.

rasdaman supports WMTS with the following request types in key-value pairs (KVP) format:

- `GetCapabilities` for obtaining a list of layers and their associated `TileMatrixSets` offered together with an overall service description;
- `GetTile` for downloading a 2D image (called `Tile`) as a small subset from a requesting Layer. This request works mostly as same as WMS `GetMap` request with some extra parameters for selecting the requested tile.

5.7.1 GetCapabilities extension

This request is used to describe the general information (e.g. service owner, contacts,. . .) about the server, and most importantly are the advertised WMTS Layers with supported styles and associated `TileMatrixSet` objects.

- A `TileMatrixSet` contains the list of pyramid members (each member is called `TileMatrix` in WTMS standard) in a CRS (typically EPSG:4326) of an associated layer.
- A `TileMatrix` is a 2D matrix of `Tiles`, each `Tile` is a 2D image and it has the fixed size: 256 x 256 pixels. To obtain a `Tile` from a `TileMatrix`, one needs two zero-based indices: `TileRow` in the height dimension and `TileCol` in the width dimension. In case, a dimension (width/height) of a `TileMatrix` is less than 256 pixels, then, this dimension contains only 1 Tile with the number of pixels from this dimension. For example, a pyramid member has grid domains 36 x 20 pixels, then, the associated `TileMatrix` contains only 1 `Tile` with size 36 x 20 pixels.

Each layer has a mandatory reference to a `TileMatrixSet` in CRS EPSG:4326; if layer's native CRS is not EPSG:4326, then, it has an extra reference to another `TileMatrixSet` in this CRS (e.g. EPSG:32633).

The naming convention for `TileMatrixSet` is: `LayerName:EPSG:code`. For example a WMTS layer's, called `germany_temperature` has a native CRS EPSG:32633, then it has references to two `TileMatrixSets`: 1. `germany_temperature:EPSG:4326` and 2. `germany_temperature:EPSG:32633`.

The WTMS GetCapabilities parameters are described in the below table:

Table 5.7: WMTS GetCapabilities Standard Parameters

Request Parameter	Value	Description	Required
SERVICE	WMTS	Service standard	Yes
VERSION	1.0.0	WMTS version used	Yes
REQUEST	GetCapabilities	Request type to be performed	Yes

For example, a WMTS `GetCapabilities` request in KVP format:

```
http://localhost:8080/rasdaman/ows?SERVICE=WMTS&VERSION=1.0.0
    &REQUEST=GetCapabilities
```

5.7.2 GetTile extension

This request is used to get a 2D small subset (called a `Tile`; typically it has fixed size 256 x 256 pixels) of a requesting layer. The result is encoded in supported formats (`image/png` and `image/jpeg`).

Based on the result of WMTS `GetCapabilities` request, one can pick the proper `TileMatrixSet` and `TileMatrix` (pyramid member) referenced by a layer to get the best detailed result at a zoom level with good performance.

Note: Unlike WMS `GetMap` request, WMTS `GetTile` request only supports processing on a layer and an optional associated style of this layer.

The WTMS `GetTile` parameters are described in the below table:

Table 5.8: WMTS GetTile Standard Parameters

Request Parameter	Value	Description	Required
SERVICE	WMTS	service standard	Yes
VERSION	1.0.0	WMTS version used	Yes
REQUEST	GetTile	Request type to be performed	Yes
LAYER	{layerName}	A layer name to be requested	Yes
STYLE	{styleName}	A style name (can be null) of the layer to be requested	Yes
FORMAT	{format}	Encoded format (image/png or image/jpeg) of the output	Yes
TILEMATRIXSET	{tileMatrixSet}	A TileMatrixSet name to be requested	Yes
TILEMATRIX	{tileMatrix}	A TileMatrix name to be requested	Yes
TILEROW	{tileRow}	A row index of the requesting TileMatrix	Yes
TILECOL	{tileCol}	A column index of the requesting TileMatrix	Yes
Other dimensions	{value}	Value allowed for this dimension (e.g. TIME, ELEVATION. . .)	No

For example, a WMTS `GetTile` request in KVP format to get a tile, encoded in image/png from a `TileMatrixSet` in CRS EPSG:4326:

```
http://localhost:8080/rasdaman/ows?SERVICE=WMTS&VERSION=1.0.0
&REQUEST=GetTile
&LAYER=germany_temperature
&STYLE=colorized
&FORMAT=image/png
&TILEMATRIXSET=germany_temperature:EPSG:4326
&TILEMATRIX=germany_temperature
&TILEROW=0
&TILECOL=0
```

5.8 Data Import

Raster data in a variety of formats, such as TIFF, netCDF, GRIB, etc. can be imported in rasdaman through the `wcst_import.sh` utility. Internally it is based on `WCS-T` requests, but hides the complexity and maintains the geo-related metadata in its so-called `petascopedb` while the raster data get imported into the rasdaman array store.

Building large *time-series / datacubes*, *mosaics*, etc. and keeping them up-to-date as new data become available is supported for a large variety of data formats and file/directory organizations.

The systemtest contains many examples for importing different types of data. Note that the ingest.template.json are template files which cannot be directly imported, as several variables need to be set.

5.8.1 Introduction

The wcst_import.sh tool is based on two concepts:

- **Recipe** - A recipe defines how a set of data files can be combined into a well-defined coverage (e.g. a 2-D mosaic, regular or irregular 3-D timeseries, etc.);
- **Ingredients** - A JSON file that configures how the recipe should build the coverage (e.g. the server endpoint, the coverage name, which files to consider, etc.).

To execute an ingredients file in order to import some data:

```
$ wcst_import.sh path/to/my_ingredients.json
```

Alternatively, wcst_import.sh can be started in the background as a daemon:

```
$ wcst_import.sh path/to/my_ingredients.json --daemon start
```

or as a daemon that is "watching" for new data at some interval (in seconds):

```
$ wcst_import.sh path/to/my_ingredients.json --watch <interval>
```

For further informations regarding the usage of wcst_import.sh:

```
$ wcst_import.sh --help
```

The workflow behind is depicted approximately on Figure 5.1.

An ingredients file showing all possible options (across all recipes) can be found here in the same directory there are several examples of different recipes.

The following recipes are provided in the rasdaman repository:

- *Mosaic map*
- *Regular timeseries*
- *Irregular timeseries*
- *General coverage*
- *Import from external WCS*
- Specialized recipes
  - *Sentinel 1*
  - *Sentinel 2*

For each one of these there is an ingredients example under the ingredients/ directory, together with an example for the available parameters Further on each recipe type is described in turn, starting with the common options shared by all recipes.

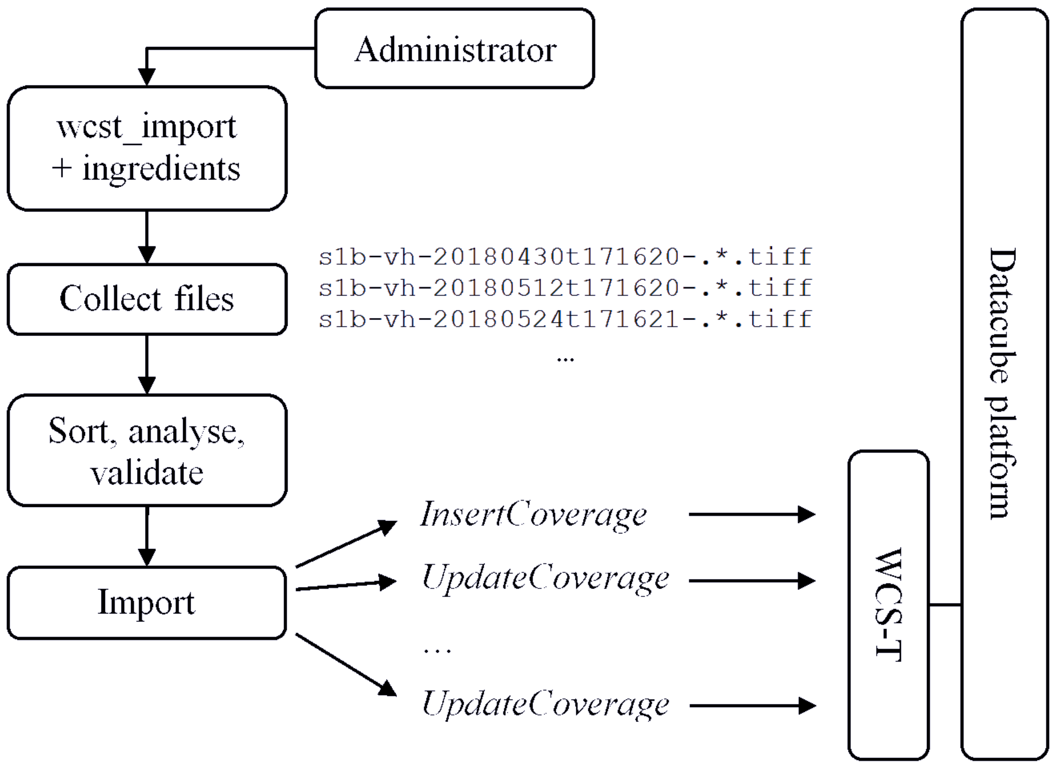


Figure 5.1: Data importing process with `wcst_import.sh`

Note: It is required to run only one `wcst_import.sh` process for registering / importing files to one specific coverage. Running multiple `wcst_import.sh` processes for building multiple different coverages are allowed (the maximum number of processes is equivalent to the number of rasservers configured in `rasmgr.conf` file).

5.8.2 Common Options

Some options are commonly applicable to all recipes. We describe these options for each top-level section of an ingredient file: config, input, recipe, and hooks.

config section

- `service_url` - The endpoint of the WCS service with the WCS-T extension enabled
- **`service_is_local` - `true` if the WCS service endpoint runs locally on the same machine,** `false` otherwise. When set to `false`, the data to be imported will be uploaded to the remote host. This may also be done even when the WCS service endpoint runs locally but has no read permissions on the data files, in which case the only way to import the data is by uploading it to the server; note, however, that this adds a performance penalty, so it should be avoided whenever possible. By default this setting is `true`.

- `mock` - Print WCS-T requests but do not execute anything if set to `true`. Set to `false` by default.
- `automated` - Set to `true` to avoid any interaction during the data import process. Useful in production environments for automated deployment for example. By default it is `false`, i.e. user confirmation is needed to execute the actual import.
- `blocking` (since v9.8) - Set to `false` to analyze and import each file separately (*non-blocking mode*). By default blocking is set to `true`, i.e. wcst_import will analyze all input files first to create corresponding coverage descriptions, and only then import them. The advantage of non-blocking mode is that the analyzing and importing happens incrementally (in blocking mode the analyzing step can take a long time, e.g. days, before the import can even begin).

  Note: When importing in *non-blocking* import mode for coverages with irregular axes, it will *only rely on sorted files by filenames* and it can fail if these axes' coefficients are collected from input files' metadata (e.g: DateTime value in TIFF's tag or GRIB metadata) as they might not be consecutive. wcst_import will not analyze all files to collect metadata to be sorted by DateTime as in default *blocking* import mode.

- `default_null_values` - This parameter adds default null values for bands that do *not* have a null value provided by the file itself. The value for this parameter should be an array containing the desired null value either as a closed interval `low:high` or single values. Example:

  ```
  "default_null_values": [ 9.96921e+36, "9.96921e+35:*" ],
  ```

  Note:

  – If set this parameter will override the null/nodata values present in the input files.

  – If this parameter is not set, wcst_import will try to detect these values for bands implicity from the first input file.

  – If set this parameter to: `[]`, then, wcst_import will create a coverage without any null values.

  Note: If a null value interval is specified, e.g `"9.96921e+35:*"`, during encode it will not be preserved as-is because null value intervals are not supported by most formats. In this case it is recommended to first specify a non-interval null value, followed by the interval, e.g. `[9.96921e+35, "9.96921e+35:*"]`.

- `tmp_directory` - Temporary directory in which gml and data files are created; should be readable and writable by rasdaman, petascope and current user. By default this is `/tmp`.
- `crs_resolver` - The crs resolver to use for generating WCS-T request. By default it

is determined from the `petascope.properties` setting.

- `url_root` - In case the files are exposed via a web-server and not locally, you can specify the root file url here; the default value is `"file://"`.
- `skip` - Set to `true` to ignore files that failed to import; by default it is `false`, i.e. the import process is terminated when a file fails to import.
- `retry` - Set to `true` to retry a failed request. The number of retries is either 5, or the value of setting `retries` if specified. This is set to `false` by default.
- `retries` - Control how many times to retry a failed WCS-T request; set to 5 by default.
- `retry_sleep` - Set number of seconds to wait before retrying after an error; a floating-point number can also be specified for sub-second precision. Default values is 1.
- `track_files` - Set to `true` to allow input files to be tracked in a JSON file `<coverage_id>.resume.json` containing a list of imported file paths, in order to avoid reimporting them when wcst_import.sh is subsequently executed again. The JSON file is generated in the directory set by the `resumer_dir_path` setting. This setting is enabled by default. Example content of a resume file `S2_L2A_32633_B01.resume.json` of a coverage `S2_L2A_32633_B01`:

```
["/tmp/s2_l2A_32633_B01_1.tiff",
 "/tmp/s2_l2A_32633_B01_2.tiff",
 ...
 "/tmp/s2_l2A_32633_B01_10.tiff"]
```

- `resumer_dir_path` - The directory in which to store the resume file generated when `track_files` is set to `true`. The user invoking wcst_import.sh must have permissions to write in this directory. By default the resume file will be stored in the same directory as the ingredients file.
- `slice_restriction` - Limit the slices that are imported to the ones that fit in a specified bounding box. Each subset in the bounding box should be of form `{ "low": 0, "high": <max> }`, where low/high are given in the axis format. Example:

```
"slice_restriction": [
  { "low": 0, "high": 36000 },
  { "low": 0, "high": 18000 },
  { "low": "2012-02-09", "high": "2012-12-09", "type": "date" }
]
```

- `description_max_no_slices` - Maximum number of slices (files) to show for preview before starting the actual data import.
- `subset_correction` (*deprecated* since v9.6) - In some cases the resolution is small enough to affect the precision of the transformation from domain coordinates to grid coordinates. To allow for corrections that will make the import possible, set this parameter to `true`.

input section

- `coverage_id` - The name of the coverage to be created; if the coverage already exists, it will be updated with the new files collected by `paths`.
- `paths` - List of absolute or relative (to the ingredients file) paths or regex patterns in format acceptable by the Python glob function. Multiple paths separated by commas can be specified. The collected file paths are by default sorted in ascending order before import, either by calculated datetime in time-series recipes, or by lexicographic comparison of the file path strings otherwise. The ordering can be changed to descending or disabled completely with the *import_order* option.

  Note: wcst_import analyzes each input file from `paths` and maximum time to open one file to analyze is 60 seconds. If during this time, the file cannot be opened, then, wcst_import will try to open it 3 more times. If the file is still not possible to open, then, it will:

  – Throw exception and stop the importing process if `skip` setting is `False`

  – Ignore the input file and continue with these other input files if `skip` setting is `True`

- `inspire` section contains the settings for importing INSPIRE coverage:
  – `metadata_url` - If set to non-empty string, then the importing coverage will be marked as INSPIRE coverage, see more details *here*. If set to empty string or omitted, then the coverage will be updated as non-INSPIRE coverage.

recipe section

- `import_order` - Indicate in which order the input files collected with the `paths` setting should be imported. In time-series recipes, the ordering is based on the datetime calculated for each file. In other recipes, e.g. `map_mosaic`, the ordering is based on lexicographic comparison of the file paths. Possible values are:
  – `ascending` (default) - import files in ascending order;
  – `descending` - import files in descending order;
  – `none` - do not order files in any particular way before import.

  Example:

```
"import_order": "descending"
```

- `tiling` - Specifies the tile structure to be created for the coverage in rasdaman. You can set arbitrary tile sizes for the tiling option only if the tile name is `ALIGNED`. Example:

```
"tiling": "ALIGNED [0:0, 0:1023, 0:1023] TILE SIZE 5000000"
```

  For more information on tiling check the *Storage Layout Language*

- `wms_import` - If set to `true`, after importing data to coverage, it will also create a WMS layer from the imported coverage and populate metadata for this layer. After that, this layer will be available from *WMS GetCapabilties request*. Example:

```
"wms_import": true
```

- `scale_levels` - Enable the *WMS pyramids* feature. Level must be positive number and greater than 1 (note: only spatial geo axes, e.g. Lat and Long are scaled down in the pyramid member coverage). A new coverage as pyramid member of the importing coverage will be created with *this pattern*. Syntax:

```
"scale_levels": [ 1.5, 2, 4, ... ]
```

- `scale_factors` - Enable the *WMS pyramids* feature. It is a more flexible variant of the `scale_levels` setting. The two settings are exclusive, either `scale_levels` or `scale_factors` can exist in the ingredient file. The *coverage_id* of each factor must be unique in rasdaman and manually set by the user. The *factors* is a list of decimal values corresponding to the coverage axes according to its CRS order; a scale value for an irregular axis must be 1, while for a regular axis it should be greater than 1; see more details *here*. For example, you can create two pyramid member 2D coverages which are 2x smaller (*cov_level_2*) and 4x smaller (*cov_level_4*) on the regular *Lat* and *Long* axes:

```
"scale_factors": [
  {
    "coverage_id": "cov_level_2",
    "factors": [2, 2]
  },
  {
    "coverage_id": "cov_level_4",
    "factors": [4, 4]
  }
 ]
```

- `import_overviews` - If specified with indices (0-based), wcst_import will import the corresponding overview levels defined in the input files as separated coverages with *this naming pattern*. The selected overview coverages are then added as pyramid memberds to the base importing coverage. For example, to import overview levels 0 and 3 from a tiff file which has 4 overview levels in total

```
gdalinfo 20100101.tif
...
Band 1 Block=89x71 Type=Byte, ColorInterp=Gray
Overviews: 45x36, 23x18, 12x9, 6x5
```

  you can specify `"import_overviews":  [0, 3]` in the ingredients.

  By default this setting is set to an empty array, i.e. no overview levels will be imported. Only GDAL recipes and gdal version 2+ are supported.

- `import_all_overviews` - If specified with `true`, all overview levels which exist in the input files will be imported. For example, to import all 4 overview levels from a tiff file you can specify `"import_all_overviews": true` in the ingredient file.

This setting and `import_overviews` are exclusive, only one can be specified. By default it is set to *false*. Only GDAL recipes and gdal version 2+ are supported.

- `import_overviews_only` - If specified with `true`, input files are not imported to the *base* coverage specified with `coverage_id`, but only to the *overview* coverages as specified in the ingredients file by either `import_all_overviews` or `import_overviews`. This setting is set to `false` by default if not specified explicitly.

Note: If the input files were already imported to the *base* coverage and they were tracked in `<base_coverage_id>.resume.json`, it is necessary to remove this resume file in order to import only the overview coverages. Alternatively the ingredients file can be copied to another directory and adapted to set `import_overviews_only` to `true`.

- `pyramid_members` - List of existing coverages which can be added as pyramid members of the importing coverage, see *request*. Syntax:

```
"pyramid_members": [ "cov_level_2",  "cov_level_4"]
```

- `pyramid_bases` - List of existing coverages to which the importing coverage will be added as a pyramid member. This parameter has the opposite effect of `pyramid_members`, see corresponding *request*. Syntax:

```
"pyramid_bases": [ "cov_A",  "cov_B"]
```

- `pyramid_harvesting` - If set to `true`, recursively add all nested pyramid members of the pyramid *member* coverage to the target *base* coverage. The pyramid member coverage depends on which of these two settings is used:
  - If `pyramid_bases` is specified, then the currently importing coverage is the pyramid member of the the base coverages listed in `pyramid_bases`;
  - Otherwise, if `pyramid_members` is specified, then the currently importing coverage is the base coverage of the pyramid member coverages listed in `pyramid_members`;
  - Otherwise, if neither of the above options is specified, an error is throws.

  See *request* for more details on the underlying request sent to petascope when this option is set to `true`. By default this option is set to `false`.

Image pyramids

Since v9.7 it is possible to create downscaled versions of a given coverage, eventually achieving something like an image pyramid, in order to enable faster WMS requests when zooming in/out.

By using the *scale_levels* option of wcst_import when importing a coverage with WMS enabled, petascope will create downscaled collections in rasdaman following this pattern: `coverageId_<level>`. If level is a float, then *the dot* is replaced with an *underscore*, as dots are not permitted in a collection name. Some examples:

- MyCoverage, level 2 -> MyCoverage_2
- MyCoverage, level 2.45 -> MyCoverage_2_45

Example ingredients specification to create two downscaled levels which are *8x* and *32x* smaller than the original coverage:

```
"options": {
  "scale_levels": [8, 32],
  ...
}
```

Two new WCS-T non-standard requests are utilized by wcst_import for this feature, see *here for more information*.

hooks section

Since v9.8, it is possible to run shell commands *before/after data import* by adding optional `hooks` top-level configuration in an ingredient file (on the same level as the `config`, `input`, and `recipe` sections).

There are 2 types of hooks:

- `before_import` - Run shell commands before analyzing the input files, e.g. reproject input files from EPSG:3857 to EPSG:4326 with gdalwarp and import the *reprojected* files only.
- `after_import` - Run shell commands after importing the input files, e.g. clean all projected files from running gdalwarp above.

When import mode is set to non-blocking (`"blocking": false`), wcst_import will run before/after hook(s) for the file which is being used to update coverage, while the default blocking importing mode will run before/after hooks for *all input files* before/after they are updated to a coverage.

Multiple before/after hooks can be specified, and they will be evaluated in the order in which they are specified. Each hook is a JSON object in the `"hooks"` JSON array, with parameters as follows:

- `description` - Describe what this hook does and wcst_import prints this message when processing this hook.

- `when` - mandatory parameter. Run a command before (set to `before_import`) or after (set to `after_import`) importing files to a coverage.
- With one of the following options either Bash or Python code must be specified, which will be run for each input file.
  - `cmd` - specify Bash commands; standard error is redirected to standard output, which wcst_import prints while executing the command. Note that the code is executed in a new Bash process newly forked for every file; if there are many files, this can be costly in terms of performance and memory usage, and it may be better to use `python_cmd`.
  - `python_cmd` - specify Python code, which is evaluated in the same Python instance already running wcst_import with the exec() method. It may be preferable to Bash `cmd` when there are many files to import, or more complex tasks need to be performed with advance math calculations, for example.

    Note: As an ingredients file can contain arbitrary Python or Shell code which wcst_import will execute before/after importing files or during the evaluation of sentence expressions, it can pose a security issue if untrusted users are allowed to write ingredients files to be executed with wcst_import. In this case, it is recommended to make sure the user executing wcst_import is properly restricted on their ingredients files.

- `abort_on_error` - Only valid for `before_import` hook. If set to `true`, when a `cmd` bash command returns an error or when a `python_cmd` raises an `Exception`, wcst_import terminates immediately.
- `replace_path` - Only valid for `before_import` hook. wcst_import considers the specified absolute paths (globbing is allowed) as the actual absolute file paths to be imported after running a hook, rather than the original input file paths configured in `paths` setting, under `input` section.

Example: Import GDAL subdatasets

The example ingredients below contains a pre-hook which replaces the collected file path into a GDAL subdataset form; in this particular case, with the GDAL driver for NetCDF a single variable from the collected NetCDF files is imported.

```
"hooks": [
    {
      "description": "Import one variable for netCDF with subdataset
↪",
      "when": "before_import",
      "cmd": "",
      "abort_on_error": true,
      // GDAL netCDF subdataset variable file path
      "replace_path": ["NETCDF:${file:path}:area"]
    }
 ]
```

Example: Preprocessing GDAL files before importing

This example ingredients below contains one `before_import` hook and one `after_import` hook. The `before_import` hook runs a bash command to project each input tiff file to a temp tiff file in *EPSG:4326* CRS, then, it collects these temp file paths to import. The `after_import` hook runs a bash command after importing to remove the temp file paths above.

```
"hooks": [
    {
      "description": "reproject input files.",
      "when": "before_import",
      "cmd": "gdalwarp -t_srs EPSG:4326 -tr 0.02 0.02 -overwrite \"$
↪{file:path}\" \"${file:path}.projected\"",
      "abort_on_error": true,
      "replace_path": ["${file:path}.projected"]
    },

    {
      "description": "Remove projected files.",
      "when": "after_import",
      "cmd": "rm -rf \"${file:path}\""
    }
    ...
]
```

5.8.3 Recipe map_mosaic

Well suited for importing a tiled map, not necessarily continuous; it will place all input files given under a single coverage and deal with their position in space. Parameters are explained below.

```
{
  "config": {
    // The endpoint of the WCS service with the WCS-T extension␣
↪enabled
    "service_url": "http://localhost:8080/rasdaman/ows",
    // If set to true, it will print the WCS-T requests and will not
    // execute them. To actually execute them set it to false.
    "mock": true,
    // If set to true, the process will not require any user␣
↪confirmation.
    // This is useful for production environments when deployment␣
↪is automated.
    "automated": false
  },
  "input": {
    // The name of the coverage; if the coverage already exists,
    // it will be updated with the new files
```

(continues on next page)

(continued from previous page)

```
    "coverage_id": "MyCoverage",
    // Absolute or relative (to the ingredients file) path or regex␣
↪that
    // would work with the ls command. Multiple paths separated by␣
↪commas
    // can be specified.
    "paths": [ "/var/data/*" ]
  },
  "recipe": {
    // The name of the recipe
    "name": "map_mosaic",
    "options": {
      // The tiling to be applied in rasdaman
      "tiling": "ALIGNED [0:511, 0:511]"
    }
  }
}
```

5.8.4 Recipe time_series_regular

Well suited for importing multiple 2-D slices created at regular intervals of time (e.g sensor data, satelite imagery etc) as 3-D cube with the third axis being a temporal one. Parameters are explained below

```
{
  "config": {
    // The endpoint of the WCS service with the WCS-T extension␣
↪enabled
    "service_url": "http://localhost:8080/rasdaman/ows",
    // If set to true, it will print the WCS-T requests and will not
    // execute them. To actually execute them set it to false.
    "mock": true,
    // If set to true, the process will not require any user␣
↪confirmation.
    // This is useful for production environments when deployment␣
↪is automated.
    "automated": false
  },
  "input": {
    // The name of the coverage; if the coverage already exists,
    // it will be updated with the new files
    "coverage_id": "MyCoverage",
    // Absolute or relative (to the ingredients file) path or regex␣
↪that
    // would work with the ls command. Multiple paths separated by␣
↪commas
    // can be specified.
```

(continues on next page)

(continued from previous page)

```
    "paths": [ "/var/data/*" ]
  },
  "recipe": {
    // The name of the recipe
    "name": "time_series_regular",
    "options": {
      // Starting date for the first spatial slice
      "time_start": "2012-12-02T20:12:02",
      // Format of the time provided above: `auto` to try to guess␣
↪it,
      // otherwise use any combination of YYYY:MM:DD HH:mm:ss
      "time_format": "auto",
      // Distance between each slice in time, granularity seconds␣
↪to days
      "time_step": "2 days 10 minutes 3 seconds",

      // CRS to be used for the time axis
      "time_crs": "http://localhost:8080/def/crs/OGC/0/AnsiDate",
      // The tiling to be applied in rasdaman
      "tiling": "ALIGNED [0:1000, 0:1000, 0:2]"
    }
  }
}
```

5.8.5 Recipe time_series_irregular

Well suited for importing multiple 2-D slices created at irregular intervals of time into a 3-D cube with the third axis being a temporal one. There are two types of time parameters in "options", one needs to be choosed according to the particular use case:

- `tag_name` - e.g. `TIFFTAG_DATETIME` in the image's metadata; the metadata should be checked with `gdalinfo <file>`, as not every image may have the tag. Below is an example:

```
{
  "config": {
    // The endpoint of the WCS service with the WCS-T extension␣
↪enabled
    "service_url": "http://localhost:8080/rasdaman/ows"
  },
  "input": {
    // The name of the coverage; if the coverage already exists,
    // it will be updated with the new files
    "coverage_id": "CoverageExampleTagName",
    // Absolute or relative (to the ingredients file) path or␣
↪regex that
    // would work with the ls command. Multiple paths separated␣
↪by commas
```

(continues on next page)

(continued from previous page)

```
    // can be specified.
    "paths": [ "/home/rasdaman/images/tag_name/*.tif" ]
  },
  "recipe": {
    // The name of the recipe
    "name": "time_series_irregular",
    "options": {
      // Get the date for the slice from a tag that can be read␣
↪by GDAL
      "time_parameter": {
        // The name of such a tag
        "metadata_tag": { "tag_name": "TIFFTAG_DATETIME" },
        // The format of the datetime value in the tag
        // Y = Year, e.g. to match TIFFTAG_DATETIME=2005
        "datetime_format": "YYYY"
      },
      // CRS to be used for the time axis
      "time_crs": "http://localhost:8080/rasdaman/def/crs/OGC/0/
↪AnsiDate",
      // The tiling to be applied in rasdaman
      "tiling": "ALIGNED [0:10, 0:1000, 0:500]"
    }
  }
}
```

- `filename` allows an arbitrary pattern to extract the time information from the data file paths. Below is an example:

```
{
  "config": {
    // The endpoint of the WCS service with the WCS-T extension␣
↪enabled
    "service_url": "http://localhost:8080/rasdaman/ows"
  },
  "input": {
    // The name of the coverage; if the coverage already exists,
    // it will be updated with the new files
    "coverage_id": "CoverageExampleFilename",
    // Absolute or relative (to the ingredients file) path or␣
↪regex that
    // would work with the ls command. Multiple paths separated␣
↪by commas
    // can be specified.
    "paths": [ "/home/rasdaman/images/filename/*" ]
  },
  "recipe": {
    // The name of the recipe
    "name": "time_series_irregular",
    "options": {
```

(continues on next page)

(continued from previous page)

```
      // Extract the date/time from the file name
      "time_parameter" :{
        "filename": {
          // The regex has to contain groups of tokens in␣
↪parentheses
          "regex": "(.*)_(.*)_(.+?)_(.*)",
          // Which regex group to use for retrieving the time␣
↪value
          "group": "2"
        },
      }
      // CRS to be used for the time axis
      "time_crs": "http://localhost:8080/def/crs/OGC/0/AnsiDate
↪",
      // The tiling to be applied in rasdaman
      "tiling": "ALIGNED [0:2, 0:1000, 0:1000]"
    }
  }
}
```

5.8.6 Recipe general_coverage

This is a highly flexible recipe that can handle any kind of data files (be it 2D, 3D or n-D) and model them in coverages of any dimensionality. It does that by allowing users to define their own coverage models with any number of bands and axes and fill the necesary coverage information through the so called ingredient sentences inside the ingredients.

Coverage parameters

Using ingredient sentences we can define any coverage model directly in the options of the ingredients file. Each coverage model contains the following parts:

- `crs` - Indicates the crs of the coverage to be constructed. Either a CRS url can be used e.g. http://localhost:8080/rasdaman/def/crs/EPSG/0/4326 or the shorthand notation `CRS1@CRS2@CRS3`, e.g. `OGC/0/AnsiDate@EPSG/0/4326` for indicating a time/date + spatial CRS.
- `metadata` - A group of options controlling metadata extraction and consolidation; more detailed information follows *below*
- `slicer` - A group of options controlling the data decoding and placement into the overall datacube; more detailed information follows *below*.

metadata section

The `metadata` section specifies in which format you want the metadata (json or xml). It can only contain characters and is limited in size by the backend database limit for CLOB columns; for postgresql (the default backend for petascope) the maximum size is 2GB (source).

- `type` - Specifies the format for storing the coverage metadata; `xml` and `json` are supported, and it is set to `xml` by default.
- `global` - Specifies fields which should be saved once for the whole coverage (e.g. the data licence, the creator etc). For example a "Title" metadata value can be set with `"global": { "Title": "'Drought code'", ... }`. Global metadata is collected automatically (only for netCDF / gdal recipe) from the first input file, if the `"global"` setting is omitted, or it is set to `"auto"`. This automatic collection is *not* done when additional global metadata needs to be added on top of the metadata present in the input file; in this case both the metadata from the file and the additional metadata have to be specified explicitly.
- `local` - Specifies fields which are fetched from each input file to be stored in coverage's metadata. When subsetting in the output coverage only *local* metadata associated to the subsetted areas will be added to the result. E.g., `"local": { "LocalMetadataKey": "${netcdf:metadata:LOCAL_METADATA}" }` sets LocalMetadataKey to a metadata value extracted from the input data; the `${..}` is explained in *Data expressions*. For a more detailed explanation of local metadata see the dedicated *Local metadata from input files* section.
- `colorPaletteTable` - Controls collection of color palette table for the created coverage, which can then be used internally when encoding coverage to, e.g. PNG, to colorize the result. Currently only GDAL-style `colorTable` with 256 color entries is supported.

  A path to an explicit Color Palette Table file can be specified, see example file; such a file can be referenced in the ingredients file with, e.g., `"colorPaletteTable": "PATH/TO/table.cpt"`.

  If `colorPaletteTable` is set to `"auto"` or not specified at all, and the slicer is set to `gdal` (see next section for info on slicers), then the color table will be read automatically from the first input file if its metadata contains one.

  If `colorPaletteTable` is set to an empty string `""`, any color table metadata will be ignored when creating coverage's global metadata.
- `bands` and `axes` - Allow specifying metadata for the coverage bands and/or axes; more details can be found in *Band and axis metadata in global metadata*.

slicer section

The `slicer` subsection specifies the driver to use to read from the data files, the required bands from data files and for each axis from the CRS how to obtain the bounds and resolution corresponding to each file.

- `type` - Specifies the decoding driver to be used; currently the following are supported:
  - `gdal` - for TIFF, PNG, and other encoding format that can be read with GDAL (check with `gdalinfo <file>`);
  - `netcdf` - for importing NetCDF data. If a netCDF file is flipped on Lat axis (South -> North coordinates increase in the output of `ncdump -c`) instead of GDAL style (North -> South coordinates decrease), then it is necessary to flip it before importing as rasdaman, e.g. with `cdo invertlat input.nc output.nc`.
  - `grib` - for GRIB data. Currently, rasdaman only supports GRIB files with `gridType` format of regular lat long `regular_ll`. If the format is different, it is necessary to preprocess the input files into regular grid type. The grid type can be retreived with `grib_dump file.grib | grep 'gridType'`.

    If a GRIB file is flipped on Lat axis (South -> North with `jScansPositively = 0` in the output of `grib_dump`) instead of GDAL style (North -> South with `jScansPositively = 1`), then it is necessary to flip it before importing to rasdaman, e.g. with `cdo invertlat input.grib output.grib`.
- `pixelIsPoint` - Only valid if `type` is `netcdf` or `grib`. In some cases, by convention in the input files, the coordinates are set in the middle of grid pixels, hence, set to `true` to extend the lower and upper bounds of each regular axis by half grid pixel to be able to import. By default it is set to `false`.
- `bands` - A list of bands/chanels/variables from the input files which should be imported to the importing coverage. Each entry is a JSON object with the following options, of which `identifier` and `name` are mandatory to specify while the rest are optional:
  - `identifier` - The name of the band in the input file. With GRIB data, only one band can be specified in the ingredients file, and the band identifier must match the `shortName` field in the GRIB messsages so only those messages will be imported. If no messages matched the band identifier, then all GRIB messages will be imported; this only works for input GRIB files with only one band.
  - `name` - The name of the band which will be used in the created coverage; this can be set to different from the `indentifier`;
  - `description` - Metadata description of the band;
  - `nilValue`` - Metadata null value of the band;
  - `nilReason` - Metadata reason for the null value of the band;
  - `uomCode` - Set the Unit of measurement (uom) code of the band. Besides setting it directly, it can also be derived from the input file metadata, with e.g. `${netcdf:variable:NAME:units}` for NetCDF or `${grib:unitsOfFirstFixedSurface}` for GRIB.

  - `filterMessagesMatching` - Default is empty. If not-empty (a dictionary of user input GRIB keys:values; keys (e.g. `shortName`) must exist in the input GRIB files), then it filters any GRIB message which has a GRIB value not contain a user input value of a GRIB key.
  - Further `"key": "value"` entries can be specified to add customized band metadata to the global coverage metadata.

- `axes` - A JSON object which configures the properties of each axis of the created coverage with `"axisLabel": { properties... }`. The possible properties are listed below; generally, `gridOrder`, `min`, `max`, and `resolution` have to be specified, except for irregular axes where `resolution` is not applicable.
  - `gridOrder` - specify the grid order of axes defined by the coverage CRS. If not specified, wcst_import will try to automatically derive the gridOrder according to the documentation below. That may fail with unusual data, in which case it will be necessary to set this setting manually for each axis.

    Axes of a CRS which is not part of the file CRS have gridOrder that is same as the order in the CRS definition. For example, if the coverage CRS is a compound CRS `OGC/0/AnsiDate@EPSG/0/4326` and data files themselves have CRS `EPSG/0/4326`, then gridOrder for the ansi axis in `OGC/0/AnsiDate` will be 0, and the gridOrder of the `EPSG/0/4326` axes will follow with 1 and 2. If the CRS order was reversed to `EPSG/0/4326@OGC/0/AnsiDate`, then the gridOrder of 4326 axes (Long/Lat) would be 0 and 1, and of AnsiDate (ansi) would be 2. Usually axes of non-file CRS (AnsiDate in this example) will also have setting `dataBound: false`.

    Below we give hints on how to determine the gridOrder of axes in the file CRS.
    * When data is imported with the `gdal` or `grib` slicer, generally the gridOrder is `n` for X axes (Longitude, E, . . .), and `n+1` for Y axes (Latitude, N, . . .).
    * When importing data with the `netcdf` slicer, the gridOrder should usually match the dimension order of the imported variable, which can be checked with `ncdump -h`; e.g. a variable `float dc(time, lat, lon)` will have gridOrder `n` for time, `n+1` for lat, and `n+2` for lon. This will work well as long as the data conforms to the *CF-conventions <https://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/cf-conventions.html#dimensions>*, and may otherwise need adjustments if the spatial dimensions are not in Y/X order.
  - `crsOrder` - The index of the geo axis in the coverage's CRS (0-based). Note: By default it is not required. Only set when one specifies a different name for this axis, than the one configured in the CRS's definition; more details can be found *here*; In this case, each axis must have an unique index `crsOrder` specified.
  - `min` - The lower bound of the axis (coordinates in the axis CRS);
  - `max`- The upper bound of the axis (coordinates in the axis CRS);
  - `resolution` - The resolution of the axis from the input file; if this axis is irregular, the resolution is set to `1`;

- `statements` - Import python utility libraries (e.g. `datetime` / `timedelta`) to support calculating `min`, `max`, `resolution`, etc; covered in more detail in a subsequent *section*;

A few additional options are specific to *irregular axes*:

- `irregular` - Set to `true` to specify that this axis is irregular, e.g. a time axis with irregular datetime indexes; if not specified, it is set to `false` by default;
- `directPositions` - A list of coefficients which are extracted and calculated based on the axis lower bound from the irregular axis values specified in the input netCDF/GRIB file.

  For example, let's consider a netCDF file that has a `time` dimension with attribute `units: "days since 1970-01-01 00:00:00"`. All stored values of the `time` axis must be converted to datetime based on the lower bound value (`"1970-01-01"`) as an origin. See this ingredients file for a full example.

```
"axes": {
    "time": {
        "statements": "from datetime import datetime,␣
↪timedelta",
        "min": "(datetime(1970,12,31,12,0,0) +␣
↪timedelta(days=${netcdf:variable:time:min})).strftime(\"
↪%Y-%m-%dT%H:%M\")",
        "max": "(datetime(1970,12,31,12,0,0) +␣
↪timedelta(days=${netcdf:variable:time:max})).strftime(\"
↪%Y-%m-%dT%H:%M\")",
        "directPositions": "[(datetime(1970,12,31,12,0,0) +␣
↪timedelta(days=x)).strftime(\"%Y-%m-%dT%H:%M\") for x in $
↪{netcdf:variable:time}]",
        "irregular": true,
            "resolution": 1,
        "gridOrder": 0
     }
}
```

- `dataBound` - Set to `false` to specify that this axis should be imported as a slicing point instead of a subset with lower and upper bounds; typical use case for this is when extracting irregular datetime values from the input file names. When not specified it is set to `true` by default.

  For example, the indexes of an irregular axis `ansi` could be extracted from dates in the file names of input netCDF files (e.g. `GlobLAI-20030101-20030110-H01V06-1.0_MERIS-FR-LAI-HA.nc`) through a regular expression.

```
"axes": {
   "ansi": {
       "min": "datetime(regex_extract('${file:name}',
↪'(GlobLAI-)(.+?)(-.+?)\\.(.*)', 2), 'YYYYMMDD')",
       "gridOrder": 0,
```

(continues on next page)

(continued from previous page)

```
        "irregular": true,
        "dataBound": false
    }
}
```

– `sliceGroupSize` - Group multiple input slices into a single slice in the created coverage, e.g., multiple daily data files onto a single week index on the coverage time axis; explained in more detail *here*;

Examples

The examples below illustrate importing data in different formats with the `general_coverage` recipe; many more can be found in the rasdaman test suite.

- Commented example for importing GRIB data (only the `recipe` section is shown for brevity):

```
"recipe": {
  "name": "general_coverage",
  "options": {
    // Provide the coverage description and the method of␣
↪building it
    "coverage": {
      // The coverage has 4 axes by combining 3 CRSes (Lat,␣
↪Long, ansi, ensemble)
      "crs": "EPSG/0/4326@OGC/0/AnsiDate@OGC/0/Index1D?axis-
↪label=\"ensemble\"",

      // specify metadata in json format
      "metadata": {
        "type": "json",

        "global": {
          // We will save the following fields from the input␣
↪file
          // for the whole coverage
          "MarsType": "'${grib:marsType}'",
          "Experiment": "'${grib:experimentVersionNumber}'"
        },

        // or automatically import metadata, netcdf/gdal only (!
↪)
        "global": "auto"

        "local": {
          // and the following field for each file that will␣
↪compose the final coverage
          "level": "${grib:level}"
```

(continues on next page)

(continued from previous page)

```
        }
      },

      // specify the "driver" for reading each file
      "slicer": {
        // Use the grib driver, which gives access to grib and␣
↪file expressions.
        "type": "grib",
        // The pixels in grib are considered to be 0D in the␣
↪middle of the cell,
        // as opposed to e.g. GeoTiff, which considers pixels␣
↪to be intervals
        "pixelIsPoint": true,
        // Define the bands to create from the files (1 band in␣
↪this case)
        "bands": [
          {
            "name": "temp2m",
            "definition": "The temperature at 2 meters.",
            "description": "We measure temperature at 2 meters␣
↪using sensors and
                            then we process the values using a␣
↪sophisticated algorithm.",
            "nilReason": "The nil value represents an error in␣
↪the sensor."
            "uomCode": "${grib:unitsOfFirstFixedSurface}",
            "nilValue": "-99999"
          }
        ],
        "axes": {
          // For each axis specify how to extract the spatio-
↪temporal position
          // of each file that is imported
          "Latitude": {
            // E.g. to determine at which Latitude the nth file␣
↪will be positioned,
            // we will evaluate the given expression on the file
            "min": "${grib:latitudeOfLastGridPointInDegrees} +
                    (${grib:jDirectionIncrementInDegrees}
                     if bool(${grib:jScansPositively})
                     else -${grib:jDirectionIncrementInDegrees})
↪",
            "max": "${grib:latitudeOfFirstGridPointInDegrees}",
            "resolution": "${grib:jDirectionIncrementInDegrees}
                           if bool(${grib:jScansPositively})
                           else -$
↪{grib:jDirectionIncrementInDegrees}",

            // This optional configuration is added since␣
↪version 9.8.
```

(continues on next page)

(continued from previous page)

```
            // The crs order specifies the order of the CRS␣
↪axis in coverage
            // that will be created and allows to change␣
↪standard abbreviation for axis label
            // from EPSG database to a different name (e.g: "Lat
↪" -> "Latitude").
            "crsOrder": 0
            // The grid order specifies the order of the axis␣
↪in the raster
            // that will be created
            "gridOrder": 3
          },
          "Long": {
            "min": "${grib:longitudeOfFirstGridPointInDegrees}",
            "max": "${grib:longitudeOfLastGridPointInDegrees} +
                     (-${grib:iDirectionIncrementInDegrees}
                      if bool(${grib:iScansNegatively})
                      else ${grib:iDirectionIncrementInDegrees})
↪",
            "resolution": "-${grib:iDirectionIncrementInDegrees}
                           if bool(${grib:iScansNegatively})
                           else $
↪{grib:iDirectionIncrementInDegrees}",
            "crsOrder": 1
            "gridOrder": 2
          },
          "ansi": {
            "min": "grib_datetime(${grib:dataDate}, $
↪{grib:dataTime})",
            "resolution": "1.0 / 4.0",
            "type": "ansidate",
            "crsOrder": 2,
            "gridOrder": 1,
            // In case and axis does not natively belong to a␣
↪file (e.g. as time),
            // then this property must set to false; by default␣
↪it is true otherwise.
            "dataBound": false
          },
          "ensemble": {
            "min": "${grib:localDefinitionNumber}",
            "resolution": 1,
            "crsOrder": 3,
            "gridOrder": 0
          }
        }
      },

      "tiling": "REGULAR [0:0, 0:20, 0:1023, 0:1023]"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

- Example for importing NetCDF data (full ingredients file here):

```
"recipe": {
  "name": "general_coverage",
  "options": {
    "coverage": {
      "crs": "OGC/0/UnixTime@EPSG/0/3577",
      "metadata": {
        "type": "xml",
        "global": {
          "date_created": "'${netcdf:metadata:date_created}'",
          "Conventions": "'${netcdf:metadata:Conventions}'",
          "history": "\"${netcdf:metadata:history}\"",
          "title": "'${netcdf:metadata:title}'",
          "summary": "'${netcdf:metadata:summary}'",
          "product_version": "'${netcdf:metadata:product_
↪version}'",
          "test_empty_attribute": "",
          "source": "'${netcdf:metadata:source}'"
        },
        "bands": {
          "band_1": {
            "product_version": "'${netcdf:metadata:product_
↪version}'",
            "test_empty_attribute": ""
          },
          "band_7": {
            "date_created": "'${netcdf:metadata:date_created}'",
            "Conventions": "'${netcdf:metadata:Conventions}'"
          }
        },
        "axes": {
          "unix": {
            "min": "${netcdf:variable:unix:min}",
            "max": "${netcdf:variable:unix:max}",
            "directPositions": "${netcdf:variable:E:min}"
          }
        }
      },
      "slicer": {
        "type": "netcdf",
        "pixelIsPoint": true,
        "bands": [
          {
            "name": "band_1",
            "description": "Nadir BRDF Adjusted Reflectance 0.
↪43-0.45 microns (Coastal Aerosol)",
```

(continues on next page)

(continued from previous page)

```
            "identifier": "band_1",
            "nilValue": "-999"
          },
          {
            "name": "band_2",
            "identifier": "band_2",
            "nilValue": "-999"
          },
          {
            "name": "band_3",
            "identifier": "band_3",
            "nilValue": "-999"
          },
          {
            "name": "band_4",
            "identifier": "band_4",
            "nilValue": "-999"
          },
          {
            "name": "band_5",
            "identifier": "band_5",
            "nilValue": "-999"
          },
          {
            "name": "band_6",
            "identifier": "band_6",
            "nilValue": "-999"
          },
          {
            "name": "band_7",
            "identifier": "band_7",
            "nilValue": "-999"
          }
        ],
        "axes": {
          "unix": {
            "min": "${netcdf:variable:unix:min}",
            "max": "${netcdf:variable:unix:max}",
            "directPositions": "${netcdf:variable:unix}",
            "gridOrder": 0,
            "irregular": true
          },
          "E": {
            "min": "${netcdf:variable:E:min}",
            "max": "${netcdf:variable:E:max}",
            "gridOrder": 2,
            "resolution": 25
          },
          "N": {
```

(continues on next page)

(continued from previous page)

```
                "min": "${netcdf:variable:N:min}",
                "max": "${netcdf:variable:N:max}",
                "gridOrder": 1,
                "resolution": -25
              }
            }
          }
        },
        "tiling": "ALIGNED [0:13, 0:999, 0:999] TILE SIZE 4000000"
      }
    }
```

- Example for importing TIFF data with the `gdal` driver (full ingredients file here):

```
"recipe": {
  "name": "general_coverage",
  "options": {
    "import_order": "ascending",
    "coverage": {
      "crs": "OGC/0/AnsiDate@EPSG/0/4326",
      "metadata": {
        "type": "xml",
        "global": {
          "Title": "'This is a test coverage'"
        }
      },
      "slicer": {
        "type": "gdal",
        "bands": [
          {
            "name": "Gray",
            "identifier": "0"
          }
        ],
        "axes": {
          "MyTimeAxis": {
            "min": "datetime(regex_extract('${file:name}', '(.
↪*)\\.(.*)',1), 'YYYYMMDD')",
            "crsOrder": 0,
            "gridOrder": 0,
            "type": "ansidate",
            "irregular": true,
            "sliceGroupSize": 7,
            "dataBound": false
          },
          "long": {
            "min": "${gdal:minX}",
            "max": "${gdal:maxX}",
            "crsOrder": 2,
```

(continues on next page)

(continued from previous page)

```
                "gridOrder": 1,
                "resolution": "${gdal:resolutionX}"
              },
              "lat": {
                "min": "${gdal:minY}",
                "max": "${gdal:maxY}",
                "crsOrder": 1,
                "gridOrder": 2,
                "resolution": "${gdal:resolutionY}"
              }
            }
          }
        },
        "tiling": "ALIGNED [0:0, 0:1023, 0:1023]"
  }
}
```

Ingredient sentences

An *ingredient sentence* can be of multiple types:

- *Numeric* - e.g. `2`, `4.5`
- *Strings* - e.g. `'Some information'`
- *Functions* - e.g. `datetime('2012-01-01', 'YYYY-mm-dd')`
- *Data expressions* - Allow to collect information from the data file being imported with a specific format driver. An expression is of form `${driverName:driverOperation}` - e.g. `${gdal:minX}` or `${netcdf:variable:time:min`. All possible expressions are documented in *Data expressions*.
- *Python expressions* - The types above can be combined into any valid Python expression; this allows to do mathematical operations, string parsing, date/time manipulation, etc. E.g. `${gdal:minX} + 1/2 * ${gdal:resolutionX}` or `datetime(${netcdf:variable:time:min} * 24 * 3600)`. Expressions can use functions from any Python library which just needs to be explicitly imported as explained in *Using libraries in sentences*.

Data expressions

Each driver allows expressions to extract information from input files. We will mark with capital letters things that vary in the expression. E.g. `${gdal:metadata:FIELD}` means that you can replace `FIELD` with any valid gdal metadata tag such as `TIFFTAG_DATETIME`. Example ingredients where data expressions are used can be found in *Examples*.

NetCDF

Type	Description	Examples
Metadata information	`${netcdf:metadata:YOUR_METADATA_FIELD}`	`${netcdf:metadata:title}`
Variable information	`${netcdf:variable:VAR_NAME:MODIFIER}` where `VAR_NAME` can be any variable in the file and `MODIFIER` can be one of: first|last|max|min; Any extra modifiers will return the corresponding metadata field on the given variable	`${netcdf:variable:t:min}` `${netcdf:variable:t:unit`
Dimension information	`${netcdf:dimension:DIM_NAME}` where `DIM_NAME` can be any dimension in the file. This will return the value on the selected dimension.	`${netcdf:dimension:time}`

GDAL

Relevant for TIFF, PNG, JPEG, and other 2D data formats.

Type	Description	Examples
Metadata information	`${gdal:metadata:METADATA_FIELD}`	`${gdal:metadata:TIFFTAG}`
Geo Bounds	`${gdal:BOUND_NAME}` where `BOUND_NAME` can be one of the minX|maxX|minY|maxY	`${gdal:minX}`
Geo Resolution	`${gdal:RESOLUTION_NAME}` where `RESOLUTION_NAME` can be one of the resolutionX|resolutionY	`${gdal:resolutionX}`
Origin	`${gdal:ORIGIN_NAME}` where `ORIGIN_NAME` can be one of the originX|originY	`${gdal:originY}`

GRIB

Type	Description	Examples
GRIB Key	`${grib:KEY}` where `KEY` can be any of the keys contained in the GRIB file `${grib:messagenumber}` is the special value to get the current processed GRIB message index (starting from 1)	`${grib:experimentVersi`

File

Type	Description	Examples
File Information	`${file:PROPERTY}` where property can be one of path|name|dir_path|original_path|original_dir_path original_* allows to get the original input file's path/directory. Used only in `before_import` hooks with `replace_path` to replace original input file paths with customized file paths.	`${file:path}`
Imported File Information	`${imported_file:PROPERTY}` where property can be one of path|name|dir_path|original_path|original_dir_path Files which were imported to rasdaman (excluding *skipped files*). This variable is used only in `after_import` hooks.	`${imported_file:`

BBox

Type	Description	Examples
Coverage axis information	`${bbox:AXIS_LABEL:PROPERTY}` where axis_label is one of coverage's axis name and property can be one of `min|max` (return the lower/upper geo bound of the selected axis). Used only in `after_import` hooks where each bbox containing the multi-dimensional bounding box of the data region affected by the update of an input file	`${bbox:Lat:min`

Special functions

A couple of special functions are available to help with more complicated expressions:

Function and Arguments	Description	Examples
`grib_datetime` • date • time	This function helps to deal with the usual grib date and time format. It returns back a datetime string in ISO format.	`grib_datetime(${grib:dataDate}, ${grib:dataTime})`
`datetime` • date • format	This function helps to deal with strange date time formats. It returns back a datetime string in ISO format.	`datetime("20120101:1200", "YYYYMMDD:HHmm")`
`regex_extract` • string • regex • group	This function extracts information from a string using regex; input is the string you parse, regex is the regular expression, group is the regex group you want to select	`datetime( regex_extract('${file:name}', '(.*)_(\\d*-\\d\\d)(.*)', 2), 'YYYY-MM')`
`replace` • str • old • new	Replaces all occurrences of a substring with another substring in the input string	`replace('${file:path}', '.tiff', '.xml')`

Using libraries in sentences

In case the ingredient sentences require functionality from extra Python libraries, they can be imported with a `statements` option. For example, to calculate the lower bound and upper bound for the time axis `ansi` (starting days from `1978-12-31T12:00:00`) one could use `datetime` and `timedelta` from the `datatime` library.

```
"ansi": {
  "statements": "from datetime import datetime, timedelta",

  "min": "(datetime(1978,12,31,12,0,0) + timedelta(days=$
↪{netcdf:variable:time:min})).strftime(\"%Y-%m-%dT%H:%M\")",
  "max": "(datetime(1978,12,31,12,0,0) + timedelta(days=$
↪{netcdf:variable:time:max})).strftime(\"%Y-%m-%dT%H:%M\")",
  "directPositions": "[(datetime(1978,12,31,12,0,0) +␣
↪timedelta(days=x)).strftime(\"%Y-%m-%dT%H:%M\") for x in $
↪{netcdf:variable:time}]",
```

(continues on next page)

(continued from previous page)

```
  "irregular": true,
      "resolution": "1",
  "gridOrder": 0,
  "crsOrder": 0,
  "type": "ansidate"
},
```

Python functions imported in this way override the *special functions* provided by wcst_import. For example, the special utility function `datetime(date_time_string, format)` to convert a string of datetime to an ISO date time format will be overridden when the `datetime` module is imported with a `statements` setting.

Note: See *details* about potential issue for running python code in the ingredients file.

Local metadata from input files

Beside the *global metadata* of a coverage, you can add *local metadata* for each file which is a part of the whole coverage (e.g. a 3D time-series coverage mosaiced from 2D GeoTiff files).

Under the metadata section add a "local" object with keys and values extracted by using format type expression. Example of extracting an attribute from a netCDF input file:

```
"metadata": {
  "type": "xml",
  "global": {
    ...
  },
  "local": {
    "LocalMetadataKey": "${netcdf:metadata:LOCAL_METADATA}"
  }
}
```

Each file's envelope (geo domain) and its local metadata will be added to the coverage metadata under `<slice>...</slice>` element if coverage metadata is imported in XML format. Example of a coverage containing local metadata in XML from 2 netCDF files:

```
<slices>

  <!--- Begin Local Metadata from netCDF file 1 -->
  <slice>
    <boundedBy>
      <Envelope>
        <axisLabels>Lat Long ansi forecast</axisLabels>
        <srsDimension>4</srsDimension>
        <lowerCorner>34.4396675 29.6015625
                    "2017-01-10T00:00:00+00:00" 0</lowerCorner>
```

(continues on next page)

(continued from previous page)

```
        <upperCorner>34.7208095 29.8828125
                     "2017-01-10T00:00:00+00:00" 0</upperCorner>
      </Envelope>
    </boundedBy>
    <LocalMetadataKey>FROM FILE 1</LocalMetadataKey>
    <fileReferenceHistory>
    /tmp/wcs_local_metadata_netcdf_in_xml/20170110_0_ecfire_fwi_dc.
↪nc
    </fileReferenceHistory>
  </slice>
  <!--- End Local Metadata from netCDF file 1 -->

  <!--- Begin Local Metadata from netCDF file 2 -->
  <slice>
    <boundedBy>
      <Envelope>
        <axisLabels>Lat Long ansi forecast</axisLabels>
        <srsDimension>4</srsDimension>
        <lowerCorner>34.4396675 29.6015625
                     "2017-02-10T00:00:00+00:00" 3</lowerCorner>
        <upperCorner>34.7208095 29.8828125
                     "2017-02-10T00:00:00+00:00" 3</upperCorner>
      </Envelope>
    </boundedBy>
    <LocalMetadataKey>FROM FILE 2</LocalMetadataKey>
    <fileReferenceHistory>
    /tmp/wcs_local_metadata_netcdf_in_xml/20170210_3_ecfire_fwi_dc.
↪nc
    </fileReferenceHistory>
  </slice>
  <!--- End Local Metadata from netCDF file 2 -->

</slices>
```

Since v10.0, local metadata for input files can be also fetched from corresponding external text files with the optional `metadata_file` option. For example:

```
"local": {
   "local_metadata_key": "${gdal:metadata:local_metadata_key}",
   "metadata_file": {
      // The metadata from the external XML file will be created
      // as a child element of this root element
      "root_element": "INSPIRE",
      // Path to the external XML file corresponding to
      // the importing input file
      "path": "replace('${file:path}', '.tiff', '.xml')"
   }
 }
```

When subsetting a coverage which contains a local metadata section from input files (via

WC(P)S requests), if the geo domains of subsetted coverage intersect with some input files' envelopes, only local metadata of these files will be added to the output coverage metadata.

For example: a `GetCoverage` request with a trim such that crs axis subsets are within netCDF file 1:

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1
       &request=GetCoverage
       &subset=ansi("2017-01-10T00:00:00+00:00")
       &subset=Lat(34.4396675,34.4396675)
       &subset=Long(29.6015625,29.6015625)
       &subset=forecast(0)
```

The coverage's metadata result will contain local metadata *only* from netCDF file 1:

```
<slices>
   <!--- Begin Local Metadata from netCDF file 1 -->
   <slice>
     <boundedBy>
       <Envelope>
         <axisLabels>Lat Long ansi forecast</axisLabels>
         <srsDimension>4</srsDimension>
         <lowerCorner>34.4396675 29.6015625
                      "2017-01-10T00:00:00+00:00" 0</lowerCorner>
         <upperCorner>34.7208095 29.8828125
                      "2017-01-10T00:00:00+00:00" 0</upperCorner>
       </Envelope>
     </boundedBy>
     <LocalMetadataKey>FROM FILE 1</LocalMetadataKey>
     <fileReferenceHistory>
     /tmp/wcs_local_metadata_netcdf_in_xml/20170110_0_ecfire_fwi_dc.
↪nc
     </fileReferenceHistory>
   </slice>
   <!--- End Local Metadata from netCDF file 1 -->
<slices>
```

Customized axis labels

By default, the axes to be configured must be matched by their name as defined by the coverage CRS. For example, a CRS `OGC/0/AnsiDate@EPSG:4326` defines three axes with labels ansi, Long, and Lat. To configure them, we would have a section as bellow:

```
"axes": {
  "AnsiDate": { ... },
  "Long":     { ... },
  "Lat":      { ... }
}
```

Since v9.8, one can change the default axis label defined by the CRS through indicating the axis index in the CRS (0-based) with the `"crsOrder"` setting. For example, to change the axis labels to MyDateTimeAxis, MyLatAxis, and MyLongAxis:

```
"axes": {
  "MyDateTimeAxis": {
    // Match ansi axis in AnsiDate CRS
    "crsOrder": 0,
    ...
  },
  "MyLongAxis": {
    // Match Long axis in EPSG:4326
    "crsOder": 2,
    ...
  },
  "MyLatAxis": {
    // Match Lat axis in EPSG:4326
    "crsOder": 1,
    ...
  }
}
```

Group coverage slices

Since v9.8, wcst_import allows to group input files on irregular axes (with `"dataBound": false`) through the `sliceGroupSize` option, which would specify the group size as a positive number. E.g:

```
"time": {
    "min": "datetime(regex_extract('${file:name}', '(.*)\\.(.*)',1),
↪ 'YYYYMMDD')",
    "gridOrder": 0,
    "type": "ansidate",
    "irregular": true,
    "sliceGroupSize": 7,
    "dataBound": false
}
```

If each input slice corresponds to index *X*, and one wants to have slice groups of size *N*, then the index would be translated with this option to `X - (X % N)`.

Typical use case is importing 3D coverage from 2D satellite imagery where the time axis is irregular and its values are fetched from input files by regex expression. Then, all input files which belong to the same time window (e.g 7 days in AnsiDate CRS with `"sliceGroupSize": 7`) will have the same value, which is the first date of the week.

Band and axis metadata in global metadata

Metadata can be individually specified for each *band* and *axis* in the ingredient file. Example:

```
"metadata": {
  "type": "xml",
  "global": {
    "description": "'3-band data.'",
    "resolution": "'1'"
  },
  "bands": {
    "red": {
      "metadata1": "metadata_red1",
      "metadata2": "metadata_red2"
    },
    "green": {
      "metadata3": "metadata_green3",
      "metadata4": "metadata_green4"
    },
    "blue": {
      "metadata5": "metadata_blue5"
    }
  },
  "axes": {
    "i": {
      "metadata_i_1": "metadata_1",
      "metadata_i_2": "metadata_2"
    },
    "j": {
      "metadata_j_1": "metadata_3"
    }
  }
}
```

Since v9.7, the following metadata can also be automatically derived from the input netCDF files.

band metadata

- For netCDF: If `"bands"` is set to `"auto"` or does not exist under `"metadata"` in the ingredient file, all user-specified bands will have metadata which is fetched directly from the netCDF file. Metadata for one band is collected automatically if the band is not added or it is set to `"auto"`.
- Otherwise, the user could specify metadata explicitly by a dictionary of keys/values. Example:

  ```
  "metadata": {
    "type": "xml",
  ```

(continues on next page)

(continued from previous page)

```
  "global": {
    "description": "'3-band data.'",
    "resolution": "'1'"
  },
  "bands": {
    "red": {
      "metadata1": "metadata_red1",
      "metadata2": "metadata_red2"
    },
    "green": {
      "metadata3": "metadata_green3",
      "metadata4": "metadata_green4"
    }
  }
}
```

axis metadata

- For netCDF: If `"axes"` is set to `"auto"` or does not exist under `"metadata"` in the ingredient file, all user-specified axes will have metadata which is fetched directly from the netCDF file. Metadata for one axis is collected automatically if: 1) the axis is not specified, 2) the axis is set to `"auto"`, or 3) the axis is set to `${netcdf:variable:Name:metadata}`. The axis label for variable is detected from the `min` or `max` value of CRS axis configuration under `"slicer/axes"` section. For example:

```
"slicer": {
   //...
   "axes": {
      "Long": {
         // 'lon' is variable name in netCDF file for CRS axis
↪'Long'.
         "min": "${netcdf:variable:lon:min}"
         // ...
      }
   }
 }
```

- Otherwise, the user could specify metadata explicitly as a dictionary of keys/values.

```
"metadata": {
  "type": "xml",
  "global": {
    "description": "'3-band data.'",
    "resolution": "'1'"
  },
  "axes": {
```

(continues on next page)

(continued from previous page)

```
    "i": {
      "metadata_i_1": "metadata_1",
      "metadata_i_2": "metadata_2"
    },
    "j": {
      "metadata_j_1": "metadata_3"
    }
  }
}
```

Rotated CRS support

If rasdaman is compiled with GDAL v3.4.1+, importing and querying data with rotated CRS COSMO:101 is supported. The netCDF data usually has to be preprocessed before import:

1. Invert the latitude axis when it is south to north order (lower to upper coordinates):

```
cdo invertlat input.nc inverted_input.nc
```

2. Swap the order of the rotated latitude (*rlat*) and rotated longitude (*rlon*) axes when the data variable has *rlat,rlon* order. For example, the `float CAPE_ML(time, rlat, rlon)` variable can be transformed to `float CAPE_ML(time, rlon, rlat)` with the following command:

```
ncpdq --rdr=time,rlon,rlat inverted_input.nc correct_lon_lat.nc
```

Example ingredient file for importing the CAPE_ML variable from preprocessed COSMO netCDF data:

```
{
   "config":{
      "service_url":"http://localhost:8080/rasdaman/ows",
      "tmp_directory":"/tmp/",
      "automated":true,
      "mock":false,
      "track_files":false
   },
   "input":{
      "coverage_id":"rotated_crs_coverage",
      "paths":[
         "correct_lon_lat.nc"
      ]
   },
   "recipe":{
      "name":"general_coverage",
      "options":{
         "wms_import":false,
         "coverage":{
```

(continues on next page)

(continued from previous page)

```
            "crs":"OGC/0/AnsiDate@COSMO/0/101",
            "metadata":{
               "type":"json",
               "global":"auto"
            },
            "slicer":{
               "type":"netcdf",
               "pixelIsPoint":true,
               "bands":[
                  {
                     "name":"CAPE_ML",
                     "identifier":"CAPE_ML",
                     "description":"Count of the number of␣
↪observations from the SeaWiFS sensor contributing to this bin cell
↪",
                     "nilReason":"The nil value represents an error␣
↪in the sensor."
                  }
               ],
               "axes":{
                  "ansi":{
                     "min":"(datetime(2016,12,1,0,0,0) +␣
↪timedelta(hours=${netcdf:variable:time:min})).strftime(\"%Y-%m-%dT
↪%H:%M\")",
                     "max":"(datetime(2016,12,1,0,0,0) +␣
↪timedelta(hours=${netcdf:variable:time:max})).strftime(\"%Y-%m-%dT
↪%H:%M\")",
                     "directPositions":"[(datetime(2016,12,1,0,0,0)␣
↪+ timedelta(hours=x)).strftime(\"%Y-%m-%dT%H:%M\") for x in $
↪{netcdf:variable:time}]",
                     "statements":"from datetime import datetime,␣
↪timedelta",
                     "resolution":1,
                     "gridOrder":0,
                     "type":"ansidate",
                     "crsOrder":0,
                     "irregular":true
                  },
                  "rlat":{
                     "min":"${netcdf:variable:rlat:min}",
                     "max":"${netcdf:variable:rlat:max}",
                     "gridOrder":2,
                     "crsOrder":1,
                     "resolution":"$
↪{netcdf:variable:rlat:resolution}"
                  },
                  "rlon":{
                     "min":"${netcdf:variable:rlon:min}",
                     "max":"${netcdf:variable:rlon:max}",
```

(continues on next page)

(continued from previous page)

```
                            "gridOrder":1,
                            "crsOrder":2,
                            "resolution":"$
↪{netcdf:variable:rlon:resolution}"
                        }
                    }
                }
            },
            "tiling":"ALIGNED [0:0, 0:1023, 0:1023]"
        }
    }
}
```

wcst_import automatically checks if the specified band variables (*CAPE_ML* in the above example) have a `grid_mapping` metadata entry (e.g. `CAPE_ML:grid_mapping = "rotated_pole"`), and adds all metadata from the grid mapping variable (`rotated_pole`) to the global metadata of the imported coverage. With the added `grid_mapping` section, the global metadata of the coverage might look as below, for example:

```
.. more global metadata

"CDO": "Climate Data Operators version 1.9.6 (http://mpimet.mpg.de/
↪cdo)",
"nco_openmp_thread_number": "1",

"grid_mapping": {
  "identifier": "rotated_pole",
  "grid_mapping_name": "rotated_latitude_longitude",
  "grid_north_pole_longitude": "-170.0",
  "grid_north_pole_latitude": "40.0",
  "semi_major_axis": "6371229.0",
  "semi_minor_axis": "6371229.0"
}
```

When encoding to netCDF in WCS or WCPS requests with the same `COSMO:101` CRS, rasdaman will add this grid mapping metadata as a non-dimension variable in the output, so that it has the correct CRS information. The name of the non-dimension variable in the output is set from the `identifier` value (`rotated_pole` above).

5.8.7 Recipe wcs_extract

Allows to import a coverage from a remote petascope endpoint into the local petascope. Parameters are explained below.

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "default_crs": "http://localhost:8080/def/crs/EPSG/0/4326",
    "automated": true
  },
  "input": {
    "coverage_id": "test_wcs_extract"
  },
  "recipe": {
    // name of recipe
    "name": "wcs_extract",
    "options": {
      // remote coverage id in remote petascope
      "coverage_id": "test_time3d",
      // remote petascope endpoint
      "wcs_endpoint" : "http://localhost:8080/rasdaman/ows",
      // the partitioning scheme as a list of the maximum number of␣
↪pixels on each
      // axis dimension e.g. [500, 500, 1] will split the 3-D␣
↪coverage in 2-D slices
      // of 500 by 500.
      "partitioning_scheme" : [0, 0, 500],
      // The tiling to be applied in rasdaman
      "tiling": "ALIGNED [0:2000, 0:2000]"
    }
  }
}
```

5.8.8 Recipe sentinel1

This is a convenience recipe for importing Sentinel 1 data in particular; currently only GRD/SLC product types are supported, and only geo-referenced tiff files. Below is an example:

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "automated": true,
    "track_files": false
  },
  "input": {
    "coverage_id": "S1_GRD_${modebeam}_${polarisation}",
```

(continues on next page)

(continued from previous page)

```
     // (e.g: a geo-referenced tiff file to CRS: EPSG:4326, mode␣
↪beam IW,
     //  singler polarisation VH:
     // s1a-iw-grd-vh-20190226t171654-20190326t171719-026512-02f856-
↪002.tiff)
    "paths": [ "*.tiff" ],

    // If not specified, default product is "GRD"
    "product": "SLC"

    "modebeams": ["EW", "IW"],
    "polarisations": ["HH", "HV", "VV", "VH"]
  },
  "recipe": {
    "name": "sentinel1",
    "options": {
      "coverage": {
        "metadata": {
          "type": "xml",
          "global": {
            "Title": "'Sentinel-1 GRD data served by rasdaman'"
          }
        }
      },
      "tiling": "ALIGNED [0:0, 0:1999, 0:1999] TILE SIZE 32000000",
      "wms_import": true
    }
  }
}
```

The recipe extends *general_coverage* so the `"recipe"` section has the same structure. However, a lot of information is automatically filled in by the recipe now, so the ingredients file is much simpler as the example above shows.

The other obvious difference is that the `"coverage_id"` is templated with several variables enclosed in `${` and `}` which are automatically replaced to generate the actual coverage name during import:

- `modebeam` - the mode beam of input files, e.g. `IW/EW`.
- `polarisation` - single polarisation of input files, e.g: `HH/HV/VV/VH`

If the files collected by `"paths"` are varying in any of these parameters, the corresponding variables must appear somewhere in the `"coverage_id"` (as for each combination a separate coverage will be constructed). Otherwise, the data import will either fail or result in invalid coverages. E.g. if all data's mode beam is `IW`, but still different polarisations, the `"coverage_id"` could be `"MyCoverage_${polarisation}"`;

In addition, the data to be imported can be optionally filtered with the following options in the `"input"` section:

- `modebeams` - specify a subset of mode beams to import from the data, e.g. only the `IW` mode beam; if not specified, data of all supported mode beams will be ingested.

- `polarisations` - specify a subset of polarisations to import, e.g. only the `HH` polarisation; if not specified, data of all supported polarisations will be imported.

Limitations:

- Only GRD/SLC products are supported.
- Data must be geo-referenced.
- Filenames are assumed to be of the format: `s1[ab]-(.*?)-grd(.?)-(.*?)-(.*?)-(.*?)-(.*?)-(.*?)-(.*?).tiff` or `s1[ab]-(.*?)-slc(.?)-(.*?)-(.*?)-(.*?)-(.*?)-(.*?)-(.*?).tiff`.

5.8.9 Recipe sentinel2

This is a convenience recipe for importing Sentinel 2 data in particular. It relies on support for Sentinel 2 in more recent GDAL versions. Importing zipped Sentinel 2 is also possible and automatically handled.

Below is an example:

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "automated": true
  },
  "input": {
    "coverage_id": "S2_${crsCode}_${resolution}_${level}",
    "paths": [ "S2*.zip" ],
    // Optional filtering settings
    "resolutions": ["10m", "20m", "60m", "TCI"],
    "levels": ["L1C", "L2A"],
    "crss": ["32757"] // remove or leave empty to import any CRS
  },
  "recipe": {
    "name": "sentinel2",
    "options": {
      "coverage": {
        "metadata": {
          "type": "xml",
          "global": {
            "Title": "'Sentinel-2 data served by rasdaman'"
          }
        }
      },
      "tiling": "ALIGNED [0:0, 0:1999, 0:1999] TILE SIZE 32000000",
      "wms_import": true
    }
  }
}
```

The recipe extends *general_coverage* so the `"recipe"` section has the same structure. However, a lot of information is automatically filled in by the recipe now, so the ingredients file is much simpler as the example above shows.

The other obvious difference is that the `"coverage_id"` is templated with several variables enclosed in `${` and `}` which are automatically replaced to generate the actual coverage name during import:

- `crsCode` - the CRS EPSG code of the imported files, e.g. `32757` for WGS 84 / UTM zone 57S.
- `resolution` - Sentinel 2 products bundle several subdatasets of different resolutions:
  - `10m` - bands B4, B3, B2, and B8 (base type unsigned short)
  - `20m` - bands B5, B6, B7, B8A, B11, and B12 (base type unsigned short)
  - `60m` - bands B1, B8, and B10 (base type unsigned short)
  - `TCI` - True Color Image (red, green, blue char bands); also 10m as it is derived from the B2, B3, and B4 10m bands.
- `level` - `L1C` or `L2A`

If the files collected by `"paths"` are varying in any of these parameters, the corresponding variables must appear somewhere in the `"coverage_id"` (as for each combination a separate coverage will be constructed). Otherwise, the import will either fail or result in invalid coverages. E.g. if all data is level `L1C` with CRS `32757`, but still different resolutions, the `"coverage_id"` could be `"MyCoverage_${resolution}"`; the other variables can still be specified though, so `"MyCoverage_${resolution}_${crsCode}"` is valid as well.

In addition, the data to be imported can be optionally filtered with the following options in the `"input"` section:

- `resolutions` - specify a subset of resolutions to import from the data, e.g. only the “10m” subdataset; if not specified, data of all supported resolutions will be ingested.
- `levels` - specify a subset of levels to import, so that files of other levels will be fully skipped; if not specified, data of all supported levels will be ingested.
- `crss` - specify a list of CRSs (EPSG codes as strings) to import; if not specified or empty, data of any CRS will be imported.

5.8.10 Creating your own recipe

The recipes above cover a frequent but limited subset of what is possible to model using a coverage. WCSTImport allows to define your own recipes in order to fill these gaps. In this tutorial we will create a recipe that can construct a 3D coverage from 2D georeferenced files. The 2D files that we want to target have all the same CRS and cover the same geographic area. The time information that we want to retrieve is stored in each file in a GDAL readable tag. The tag name and time format differ from dataset to dataset so we want to take this information as an option to the recipe. We would also want to be flexible with the time crs that we require so we will add this option as well.

Based on this usecase, the following ingredient file seems to fulfill our need:

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "mock": false,
    "automated": false
  },
  "input": {
    "coverage_id": "MyCoverage",
    "paths": [ "/var/data/*" ]
  },
  "recipe": {
    "name": "my_custom_recipe",
    "options": {
      "time_format": "auto",
      "time_crs": "http://localhost:8080/def/crs/OGC/0/AnsiDate",
      "time_tag": "MY_SPECIAL_TIME_TAG",
    }
  }
}
```

To create a new recipe start by creating a new folder in the recipes folder. Let's call our recipe `my_custom_recipe`:

```
$ cd $RMANHOME/share/rasdaman/wcst_import/recipes_custom/
$ mkdir my_custom_recipe
$ touch __init__.py
```

The last command is needed to tell python that this folder is containing python sources, if you forget to add it, your recipe will not be automatically detected. Let's first create an example of our ingredients file so we get a feeling for what we will be dealing with in the recipe. Our recipe will just request from the user two parameters Let's now create our recipe, by creating a file called `recipe.py`

```
$ touch recipe.py
$ editor recipe.py
```

Use your favorite editor or IDE to work on the recipe (there are type annotations for most WCSTImport classes so an IDE like PyCharm would give out of the box completion support). First, let's add the skeleton of the recipe (note that in this tutorial, we will omit the import section of the files (your IDE will help you auto import them)):

```
class Recipe(BaseRecipe):
    def __init__(self, session):
        """
        The recipe class for my_custom_recipe.
        :param Session session: the session for the import tun
        """
        super(Recipe, self).__init__(session)
```

(continues on next page)

(continued from previous page)

```
        self.options = session.get_recipe()['options']

    def validate(self):
        super(Recipe, self).validate()
        pass

    def describe(self):
        """
        Implementation of the base recipe describe method
        """
        pass

    def ingest(self):
        """
        Imports the input files
        """
        pass

    def status(self):
        """
        Implementation of the status method
        :rtype (int, int)
        """
        pass

    @staticmethod
    def get_name():
        return "my_custom_recipe"
```

The first thing you need to do is to make sure the `get_name()` method returns the name of your recipe. This name will be used to determine if an ingredient file should be processed by your recipe. Next, you will need to focus on the constructor. Let's examine it. We get a single parameter called `session` which contains all the information collected from the user plus a couple more useful things. You can check all the available methods of the class in the session.py file, for now we will just save the options provided by the user that are available in `session.get_recipe()` in a class attribute.

In the `validate()` method, you will validate the options for the recipe provided by the user. It's generally a good idea to call the super method to validate some of the general things like the WCST Service availability and so on although it is not mandatory. We also want to validate our custom recipe options here. This is how the recipe looks like now:

```
class Recipe(BaseRecipe):
    def __init__(self, session):
        """
        The recipe class for my_custom_recipe.
        :param Session session: the session for the import tun
        """
        super(Recipe, self).__init__(session)
```

(continues on next page)

(continued from previous page)

```
        self.options = session.get_recipe()['options']

    def validate(self):
        super(Recipe, self).validate()
        if "time_crs" not in self.options:
            raise RecipeValidationException(
                "No valid time crs provided")

        if 'time_tag' not in self.options:
            raise RecipeValidationException(
                "No valid time tag parameter provided")

        if 'time_format' not in self.options:
            raise RecipeValidationException(
                "You have to provide a valid time format")

    def describe(self):
        """
        Implementation of the base recipe describe method
        """
        pass

    def ingest(self):
        """
        Imports the input files
        """
        pass

    def status(self):
        """
        Implementation of the status method
        :rtype (int, int)
        """
        pass

    @staticmethod
    def get_name():
        return "my_custom_recipe"
```

Now that our recipe can validate the recipe options, let's move to the `describe()` method. This method allows you to let your users know any relevant information about the data import before it actually starts. The `irregular_timeseries` recipe prints the timestamp for the first couple of slices for the user to check if they are correct. Similar behaviour should be done based on what your recipe has to do.

Next, we should define the import behaviour. The framework does not make any assumptions about how the correct method of data import is, however it offers a lot of utility functionality that help you do it in a more standardized way. We will continue this tutorial by describing how to take advantage of this functionality, however, note that this is not required for the recipe to work. The first thing that you need to do is to define an *importer* object. This importer object, takes a *coverage* object and imports it using WCST requests. The object has two public methods, `ingest()`, which imports the coverage into the WCS-T service (note: this can be an insert operation when the coverage was not defined, or update if the coverage exists. The importer will handle both cases for you, so you don't have to worry if the coverage already exists.) and `get_progress()` which returns a tuple containing the number of imported slices and the total number of slices. After adding the importer, the code should look like this:

```
class Recipe(BaseRecipe):
    def __init__(self, session):
        """
        The recipe class for my_custom_recipe.
        :param Session session: the session for the import tun
        """
        super(Recipe, self).__init__(session)
        self.options = session.get_recipe()['options']
        self.importer = None

    def validate(self):
        super(Recipe, self).validate()
        if "time_crs" not in self.options:
            raise RecipeValidationException(
                "No valid time crs provided")

        if 'time_tag' not in self.options:
            raise RecipeValidationException(
                "No valid time tag parameter provided")

        if 'time_format' not in self.options:
            raise RecipeValidationException(
                "You have to provide a valid time format")

    def describe(self):
        """
        Implementation of the base recipe describe method
        """
        pass

    def ingest(self):
        """
        Imports the input files
        """
        self._get_importer().ingest()

    def status(self):
        """
        Implementation of the status method
        :rtype (int, int)
        """
        pass

```

(continues on next page)

(continued from previous page)

```
    def _get_importer():
      if self.importer is None:
        self.importer = Importer(self._get_coverage())
      return self.importer

    def _get_coverage():
      pass

    @staticmethod
    def get_name():
        return "my_custom_recipe"
```

In order to build the importer, we need to create a coverage object. Let's see how we can do that. The coverage constructor requires a

- `coverage_id`: the id of the coverage
- `slices`: a list of slices that compose the coverage. Each slice defines the position in the coverage and the data that should be defined at the specified position
- `range_fields`: the range fields for the coverage
- `crs`: the crs of the coverage
- `pixel_data_type`: the type of the pixel in gdal format, e.g. Byte, Float32 etc

The coverage object can be built in many ways, we will present one such method. Let's start from the crs of the coverage. For our recipe, we want a 3D crs, composed of the CRS of the 2D images and a time CRS as indicated. The following lines of code give us exactly this:

```
# Get the crs of one of the images using a GDAL helper class.
# We are assuming all images have the same CRS.
gdal_dataset = GDALGmlUtil(self.session.get_files()[0].get_
↪filepath())
# Get the crs of the coverage by compounding the two crses
crs = CRSUtil.get_compound_crs([gdal_dataset.get_crs(), self.
↪options['time_crs']])
```

Let's also get the range fields for this coverage. We can extract them again from the 2D image using a helper class that can use GDAL to get the relevant information:

```
fields = GdalRangeFieldsGenerator(gdal_dataset).get_range_fields()
```

Let's also get the pixel base type, again using the gdal helper:

```
pixel_type = gdal_dataset.get_band_gdal_type()
```

Let's see what we have so far:

```
class Recipe(BaseRecipe):
    def __init__(self, session):
```

(continues on next page)

(continued from previous page)

```
        """
        The recipe class for my_custom_recipe.
        :param Session session: the session for the import tun
        """
        super(Recipe, self).__init__(session)
        self.options = session.get_recipe()['options']
        self.importer = None

    def validate(self):
        super(Recipe, self).validate()
        if "time_crs" not in self.options:
            raise RecipeValidationException(
                "No valid time crs provided")

        if 'time_tag' not in self.options:
            raise RecipeValidationException(
                "No valid time tag parameter provided")

        if 'time_format' not in self.options:
            raise RecipeValidationException(
                "You have to provide a valid time format")

    def describe(self):
        """
        Implementation of the base recipe describe method
        """
        pass

    def ingest(self):
        """
        Import the input files
        """
        self._get_importer().ingest()

    def status(self):
        """
        Implementation of the status method
        :rtype (int, int)
        """
        pass

    def _get_importer(self):
      if self.importer is None:
        self.importer = Importer(self._get_coverage())
      return self.importer

    def _get_coverage(self):
      # Get the crs of one of the images using a GDAL helper class.
      # We are assuming all images have the same CRS.
```

(continues on next page)

(continued from previous page)

```
        gdal_dataset = GDALGmlUtil(self.session.get_files()[0].get_
↪filepath())
        # Get the crs of the coverage by compounding the two crses
        crs = CRSUtil.get_compound_crs(
          [gdal_dataset.get_crs(), self.options['time_crs']])
        fields = GdalRangeFieldsGenerator(gdal_dataset).get_range_
↪fields()
        pixel_type = gdal_dataset.get_band_gdal_type()
        coverage_id = self.session.get_coverage_id()
        slices = self._get_slices(crs)
        return Coverage(coverage_id, slices, fields, crs, pixel_type)

    def _get_slices(self, crs):
        pass

    @staticmethod
    def get_name():
        return "my_custom_recipe"
```

As you can notice, the only thing left to do is to implement the _get_slices() method. To do so we need to iterate over all the input files and create a slice for each. Here's an example on how we could do that

```
def _get_slices(self, crs):
  # Let's first extract all the axes from our crs
  crs_axes = CRSUtil(crs).get_axes()
    # Prepare a list container for our slices
    slices = []
    # Iterate over the files and create a slice for each one
    for infile in self.session.get_files():
      # We need to create the exact position in time and space in␣
↪which to
      # place this slice # For the space coordinates we can use the␣
↪GDAL
      # helper to extract it for us, which will return a list of␣
↪subsets
      # based on the crs axes that we extracted # and will fill the
      # coordinates for the ones that it can (the easting and␣
↪northing axes)
      subsets = GdalAxisFiller(
        crs_axes, GDALGmlUtil(infile.get_filepath())).fill()
      # fill the time axis as well and indicate the position in time
      for subset in subsets:
        # Find the time axis
        if subset.coverage_axis.axis.crs_axis.is_future():
          # Set the time position for it. Our recipe extracts it␣
↪from
          # a GDAL tag provided by the user
          subset.interval.low = GDALGmlUtil(infile).get_datetime(
```

(continues on next page)

(continued from previous page)

```
            self.options["time_tag"])
        slices.append(Slice(subsets, FileDataProvider(tpair.file)))
    return slices
```

And we are done we now have a valid coverage object. The last thing needed is to define the status method. This method need to provide a status update to the framework in order to display it to the user. We need to return the number of finished work items and the number of total work items. In our case we can measure this in terms of slices and the importer can already provide this for us. So all we need to do is the following:

```
def status(self):
    return self._get_importer().get_progress()
```

We now have a functional recipe. You can try the ingredients file against it and see how it works.

```
class Recipe(BaseRecipe):
    def __init__(self, session):
        """
        The recipe class for my_custom_recipe.
        :param Session session: the session for the import tun
        """
        super(Recipe, self).__init__(session)
        self.options = session.get_recipe()['options']
        self.importer = None

    def validate(self):
        super(Recipe, self).validate()
        if "time_crs" not in self.options:
            raise RecipeValidationException(
                "No valid time crs provided")

        if 'time_tag' not in self.options:
            raise RecipeValidationException(
                "No valid time tag parameter provided")

        if 'time_format' not in self.options:
            raise RecipeValidationException(
                "You have to provide a valid time format")

    def describe(self):
        """
        Implementation of the base recipe describe method
        """
        pass

    def ingest(self):
        """
        Import the input files
```

(continues on next page)

(continued from previous page)

```
        """
        self._get_importer().ingest()

    def status(self):
        """
        Implementation of the status method
        :rtype (int, int)
        """
        pass

    def _get_importer(self):
      if self.importer is None:
        self.importer = Importer(self._get_coverage())
      return self.importer

    def _get_coverage(self):
      # Get the crs of one of the images using a GDAL helper class.
      # We are assuming all images have the same CRS.
      gdal_dataset = GDALGmlUtil(self.session.get_files()[0].get_
↪filepath())
      # Get the crs of the coverage by compounding the two crses
      crs = CRSUtil.get_compound_crs(
        [gdal_dataset.get_crs(), self.options['time_crs']])
      fields = GdalRangeFieldsGenerator(gdal_dataset).get_range_
↪fields()
      pixel_type = gdal_dataset.get_band_gdal_type()
      coverage_id = self.session.get_coverage_id()
      slices = self._get_slices(crs)
      return Coverage(coverage_id, slices, fields, crs, pixel_type)

    def _get_slices(self, crs):
      # Let's first extract all the axes from our crs
      crs_axes = CRSUtil(crs).get_axes()
      # Prepare a list container for our slices
      slices = []
      # Iterate over the files and create a slice for each one
      for infile in self.session.get_files():
        # We need to create the exact position in time and space in␣
↪which to
        # place this slice # For the space coordinates we can use␣
↪the GDAL
        # helper to extract it for us, which will return a list of␣
↪subsets
        # based on the crs axes that we extracted # and will fill␣
↪the
        # coordinates for the ones that it can (the easting and␣
↪northing axes)
        subsets = GdalAxisFiller(
            crs_axes, GDALGmlUtil(infile.get_filepath())).fill()
```

(continues on next page)

(continued from previous page)

```
        # fill the time axis as well and indicate the position in␣
↪time
        for subset in subsets:
            # Find the time axis
            if subset.coverage_axis.axis.crs_axis.is_future():
            # Set the time position for it. Our recipe extracts it␣
↪from
            # a GDAL tag provided by the user
            subset.interval.low = GDALGmlUtil(infile).get_datetime(
                self.options["time_tag"])
        slices.append(Slice(subsets, FileDataProvider(tpair.file)))
      return slices

    @staticmethod
    def get_name():
        return "my_custom_recipe"
```

5.8.11 Importing many files

When an ingredient contains many paths to be imported, usually more than 1000, this may lead to hitting some system limits during the import.

In particular when data is imported with the GDAL driver, wcst_import has a cache of open GDAL datasets to avoid reopening files, which is costly. With too many open GDAL datasets limit on max open files can be reached, which is often 1024 (see `ulimit -n`). wcst_import handles this case by clearing its cache; however, this may degrade import performance, so increasing the limit on open files should be considered.

Furthermore, limits on maximum number of threads may be reached as well, as each open GDAL dataset creates several threads. This will lead to errors such as `fork: retry: Resource temporarily unavailable`. The maximum allowed number can be observed with `cat /sys/fs/cgroup/pids/user.slice/user-<id>.slice/pids.max`, where `<id>` can be found with `id -u <user>` for the user with which wcst_import is executed. Increasing to a larger value, e.g. 4194304, should solve this issue.

Finally, wcst_import.sh allows to control the gdal cache size with the `-c`, `--gdal-cache-size <size>` option. The specified value can be one of: `-1` (no limit, cache all files), `0` (fully disable caching), `N` (clear the cache whenever it has more than `N` datasets, `N` should be greater than 0). The default value is `-1` if this option is not specified.

5.8.12 Logging and error handling

wcst_import outputs log messages to the console, as well as to a log file if the user that executed wcst_import has write permissions to it. The log file name is created from settings `resumer_dir_path` and `coverage_id` in the ingredients file in format `resumer_dir_path/coverage_id.json`. If `resumer_dir_path` is not set in the ingredients file, by default the log file will be written in the folder containing the imported ingredients file with file name `coverage_id.json`.

Errors that occur while wcst_import is running are handled in the following way:

- The error message is written in the terminal console;
- The error message and the full stack trace are written to the log file.

5.9 Data export

WCS formats are requested via the **format** KVP key (`<gml:format>` elements for XML POST requests), and take a valid **MIME type** as value. Output encoding is passed on to the the GDAL library, so the limitations on output formats are devised accordingly by the supported raster formats of GDAL. The valid MIME types which Petascope may support can be checked from the WCS 2.0.1 GetCapabilities response:

```
<wcs:formatSupported>application/gml+xml</wcs:formatSupported>
<wcs:formatSupported>image/jpeg</wcs:formatSupported>
<wcs:formatSupported>image/png</wcs:formatSupported>
<wcs:formatSupported>image/tiff</wcs:formatSupported>
<wcs:formatSupported>image/bmp</wcs:formatSupported>
<wcs:formatSupported>image/jp2</wcs:formatSupported>
<wcs:formatSupported>application/netcdf</wcs:formatSupported>
<wcs:formatSupported>text/csv</wcs:formatSupported>
<wcs:formatSupported>application/json</wcs:formatSupported>
<wcs:formatSupported>application/dem</wcs:formatSupported>
...
```

In case of *encode* processing expressions, besides MIME types **WCPS** (and *rasql*) can also accept GDAL format identifiers or other commonly-used format abbreviations like “CSV” for Comma-Separated-Values for instance.

5.9.1 Support for time in netCDF output

If the global metadata of a coverage contains `"units"` and `"calendar"` settings for the time axis, when encoding to netCDF rasdaman will adjust the coordinates of the time variable based on the origin specified in the `"units"` and `"calendar"` setting instead of the time CRS. Only `standard` and `proleptic_gregorian` calendars are currently supported. More details on these standard attributes of time variables can be found in the CF conventions docs.

For example, a coverage might have this metadata for the ansi time axis:

```
<axes>
    <ansi>
        <standard_name>time</standard_name>
        <units>hours since 2016-12-01 00:00:00</units>
        <calendar>proleptic_gregorian</calendar>
        <axis>T</axis>
    </ansi>
    ...
</axes>
```

The values of `ansi` variable in the output netCDF file will be based on the origin `2016-12-01 00:00:00` as specified by the `<units>` above, instead of `1600-12-31`, the origin of the `AnsiDate` CRS associated with this axis.

5.10 rasdaman / petascope Geo Service Administration

The petascope conpoment, which geo services contact through its OGC APIs, uses rasdaman for storing the raster arrays; geo-related data parts (such as geo-referencing), as per coverage standard, are maintained by petascope itself.

Petascope is implemented as a war file of Java servlets. Internally, incoming requests requiring coverage evaluation are translated by petascope, with the help of the coverage metadata, into rasql queries executed by rasdaman as the central workhorse. Results returned from rasdaman are forwarded by petascope to the client.

Note: rasdaman can maintain arrays not visible via petascope (such as non-geo objects like human brain images). Data need to be imported via *Data Import*, not rasql, for being visible as coverages.

For further internal documentation on petascope see Developer introduction to petascope and its metadata database.

5.10.1 Service Startup and Shutdown

Depending of how `java_server` is configured in `petascope.properties`, starting the petascope Web application is different as follows:

- If set to `external`, then managing the petascope Web application is done via the system Tomcat in which it is deployed, e.g.

```
$ systemctl start tomcat
$ systemctl stop tomcat
$ systemctl restart tomcat
```

- If set to *embedded* then petascope is managed along with rasdaman; see *this section* for more details.

5.10.2 Configuration

The rasdaman-geo frontend (petascope) can be configured via changing settings in `/opt/rasdaman/etc/petascope.properties`. For changes to take effect, system Tomcat (if *deployment* is `external`) or rasdaman (if *deployment* is `embedded`) needs to be restarted after editing this file.

Database

- `spring.datasource.url` set the connectivity string to the database administered by rasdaman-geo. Supported databases are PostgreSQL, H2, HSQLDB; for more details, see *this section*.
  - Default: `jdbc:postgresql://localhost:5432/petascopedb`
  - Need to change: **YES** when DMBS other than PostgreSQL is used
- `spring.datasource.username` set the username for connecting to the above database.
  - Default: `petauser`
  - Need to change: **YES** when changed in the above database
- `spring.datasource.password` set the password for the user specified by `spring.datasource.username`.
  - Default: `randomly generated password`
  - Need to change: **YES** when changed in the above database
- `spring.datasource.jdbc_jar_path` absolute path to the JDBC jar file for connecting to the database configured in setting `spring.datasource.url`. If left empty, the default PostgreSQL JDBC driver will be used. To use a different DBMS (e.g. H2), please download the corresponding JDBC driver, and set the path to it.
  - Default: empty
  - Need to change: **YES** when a DMBS other than PostgreSQL is used
- `spring.datasource.tomcat.initial-size` set the initial size for JDBC connections in pool.
  - Default: `30 connections`
  - Need to change: NO
- `spring.datasource.tomcat.max-active` set the maximum number of active JDBC connections in pool.
  - Default: `70 connections`

  - Need to change: NO

- `spring.datasource.tomcat.max-idle` set the maximum number of idle JDBC connections in pool.
  - Default: `30 connections`
  - Need to change: NO
- `metadata_url` set the connectivity string to the database administered by rasdaman-geo. This setting is only used for *database migration* from one DBMS to another (e.g. PostgreSQL to H2) with *migrate_petascopedb.sh*; in this case `metadata_url` is used to connect to the *source database*, while `spring.datasource.url` is used to connect to the *target database*.
  - Default: `jdbc:postgresql://localhost:5432/petascopedb`
  - Need to change: **YES** when migrating from a DMBS different from PostgreSQL
- `metadata_user` set the username for the above database
  - Default: `petauser`
  - Need to change: **YES** when different in the above database
- `metadata_pass` set the password for the user specified by `metadata_user`
  - Default: `petapasswd`
  - Need to change: **YES** when different in the above database
- `metadata_jdbc_jar_path` absolute path to the JDBC jar file for connecting to the database configured in setting `metadata_url`. If left empty, the default PostgreSQL JDBC driver will be used. To use a different DBMS (e.g. H2), please download the corresponding JDBC driver, and set the path to it.
  - Default: empty
  - Need to change: **YES** when a DMBS other than PostgreSQL is used

General

- `server.contextPath` when rasdaman-geo is running in embedded mode (setting *java_server*), this setting allows to control the prefix in the deployed web application URL, e.g. the `/rasdaman` in `http://localhost:8080/rasdaman/ows`.
  - Default: `/rasdaman`
  - Need to change: NO
- `secore_urls` set SECORE endpoints to be used by rasdaman-geo. Multiple endpoints for fail-safety can be specified as a comma-separated list, attempted in order as listed. By default, `internal` indicates that rasdaman-geo should use its own `SECORE`, which is more efficient as it avoids external HTTP requests.
  - Default: `internal`

  - Need to change: NO

- `xml_validation` if set to `true`, WCS `POST/SOAP` XML requests will be validated against `OGC WCS 2.0.1` schema definitions; when starting Petascope it will take around 1-2 minutes to load the schemas from the OGC server.

  Note: Passing the *OGC CITE* tests requires this parameter to be set to `false`.

  - Default: `false`
  - Need to change: NO

- `ogc_cite_output_optimization` if `true`, rasdaman-geo will optimize responses in order to pass a couple of invalid *OGC CITE* test cases. Indentation of `WCS GetCoverage` and `WCS DescribeCoverage` results, for example, will be trimmed.

  - Default: `false`
  - Need to change: NO, except when executing *OGC CITE* tests

- `petascope_servlet_url` set the service endpoint in `<ows:HTTP>` elements of the result of `GetCapabilities`. Change to your public service URL if rasdaman-geo runs behind a proxy; if not set then it will be automatically derived, usually to `http://localhost:8080/rasdaman/ows`.

  - Default: empty
  - Need to change: **YES** when rasdaman-geo runs behind a proxy

- `max_wms_cache_size` set the maximum amount of memory (in bytes) to use for caching WMS `GetMap` requests. This setting speeds up repeating WMS operaions over similar area/zoom level. It is recommended to consider increasing the parameter if the system has more RAM, but make sure to correspondingly update the `-Xmx` option for Tomcat as well. The cache evicts least recently inserted data when it reaches the maximum limit specified here.

  - Default: `100000000` (100 MB)
  - Need to change: NO

- `uploaded_files_dir_tmp` set an absolute path to a server directory where files uploaded to rasdaman-geo by a request will be temporarily stored; the user running rasdaman-geo (either tomcat or rasdaman) should have write permissions on the specified directory.

  - Default: `/tmp/rasdaman_petascope/upload`
  - Need to change: NO

- `full_stacktraces` log only stacktraces generated by rasdaman (`false`), or full stacktraces including all external libraries (`true`). It is recommended to keep this setting to `false` for shorter exception stacktraces in `petascope.log`.

  - Default: `false`

  - Need to change: NO

- `inspire_common_url` set the URL to an external catalog service for the INSPIRE standard, to be provided by the user. If not set then it will be automatically derived from the *petascope_servlet_url* setting.
  - Default: empty
  - Need to change: NO

Deployment

- `java_server` specify how is rasdaman-geo deployed: `embedded` starts the Web application standalone with embedded Tomcat, listening on the `server.port` setting as configured below, while `external` indicates that `rasdaman.war` is deployed in the `webapps` dir of external Tomcat.

  It is recommended to set `embedded`, as there is no dependency on external Tomcat server, `petascope.log` can be found in the rasdaman log directory `/opt/rasdaman/log`, and start/stop of rasdaman-geo is in sync with starting/stopping the rasdaman service. Setting to `external` on the other hand can be preferred when there is already an existing Tomcat server running other Web applications.
  - Default: `embedded`
  - Need to change: NO, unless rasdaman-geo is deployed in external Tomcat
- `server.port` set the port on which `embedded` rasdaman-geo (`java_server=embedded` above) will listen when rasdaman starts. This setting has no effect when `java_server=external`.
  - Default: `8080`
  - Need to change: **YES** when port `8080` is occupied by another process, e.g. external Tomcat
- `static_html_dir_path` absolute path to a directory containing static demo Web pages (html/css/javascript). If set, rasdaman-geo will serve the `index.html` in this directory at the `/rasdaman` endpoint, e.g. `http://localhost:8080/rasdaman/`. Changes of files in this directory do not require a rasdaman-geo restart. The system user running Tomcat (if `java_server=external`) or rasdaman (if `java_server=embedded`) must have read permission on this directory.
  - Default: empty
  - Need to change: **YES** when demo web pages required under radaman-geo's endpoint

Rasdaman

- `rasdaman_url` set the connection URL to the rasdaman database. Normally rasdaman is installed on the same machine, so the bellow needs no changing (unless the default `rasmgr` port 7001 has changed).
  - Default: `http://localhost:7001`
  - Need to change: NO, unless changed in rasdaman (not recommended)
- `rasdaman_database` set the name of the rasdaman database (configured in `/opt/rasdaman/etc/rasmgr.conf`).
  - Default: `RASBASE`
  - Need to change: NO, unless changed in rasdaman (not recommended)
- `rasdaman_user` set the user for **unauthenticated** read-only access to rasdaman. Any request which does not provide credentials for a rasdaman user in basic authentication format in the HTTP Authorization header, will entail executing read-only operations with this user in rasdaman. It is best to limit this user to read-only access in rasdaman by granting the `R` permission to it.
  - Default: `rasguest`
  - Need to change: **YES** when changed in rasdaman
- `rasdaman_pass` set the password for the user set for `rasdaman_user`. It is recommended to change the default password for `rasguest` user in rasdaman and update the value here.
  - Default: `rasguest`
  - Need to change: **YES** when changed in rasdaman
- `rasdaman_admin_user` this user is used to map updating OGC requests (e.g. during data import, or deleting coverages) to updating rasql queries, for any request which does not provide credentials for a rasdaman user in basic authentication format in the HTTP Authorization header. Additionally, these credentials are used internally for various tasks which require admin access rights in rasdaman.

  Generally, this user should be granted full admin permissions.
  - Default: `rasadmin`
  - Need to change: **YES** when changed in rasdaman
- `rasdaman_admin_pass` set the password for the user set for `rasdaman_admin_user`. It is recommended to change the default password for `rasadmin` user in rasdaman and update the value here.
  - Default: `rasadmin`
  - Need to change: **YES** when changed in rasdaman
- `rasdaman_retry_attempts` set the number of re-connect attempts to a rasdaman server in case a connection fails.

  - Default: `5`
  - Need to change: NO
- `rasdaman_retry_timeout` set the wait time in seconds between re-connect attempts to a rasdaman server.
  - Default: `10` (seconds)
  - Need to change: NO
- `rasdaman_bin_path` absolute path to the *rasdaman executables directory*.
  - Default: `/opt/rasdaman/bin`
  - Need to change: NO

Security

- `allow_write_requests_from` configure from which IP addresses (as a comma-separated list) should the server accept write requests such as WCS-T `InsertCoverage`, `UpdateCoverage` and `DeleteCoverage`. `127.0.0.1` will allow locally generated requests, usually needed to import data with `wcst_import.sh`; setting to empty will block all requests, while `*` will allow any IP address.

  Note: This setting (i.e. the origin IP) is ignored when a request contains basic auth credentials for a valid rasdaman user with `RW` rights in the HTTP Authorization header.

  - Default: `127.0.0.1`
  - Need to change: NO, unless more IP addresses should be allowed to execute write requests
- `security.require-ssl` allow *embedded* petascope to work with HTTPS requests from its endpoint.
  - Default: `false`
  - Need to change: NO

Logging

rasdaman-geo uses the `log4j` library version `1.2.17` provided by Spring Boot version `1.5.2` to log information/errors in a `petascope.log` file. See the log4j 1.2 document for more details.

- Configuration for petascope logging; by default only level `INFO` or higher is logged to a file. The valid logging levels are `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR` and `FATAL`.

```
log4j.rootLogger=INFO, rollingFile
```

- Configuration for reducing logs from external libraries: Spring, Hibernate, Liquibase, GRPC and Netty.

```
log4j.logger.org.springframework=WARN
log4j.logger.org.hibernate=WARN
log4j.logger.liquibase=WARN
log4j.logger.io.grpc=WARN
log4j.logger.io.netty=WARN
log4j.logger.org.apache=WARN
```

- Configure `file` logging. The paths for `file` logging specified below should be write-accessible by the system user running Tomcat. If running embedded Tomcat, then the files should be write accessible by the system user running rasdaman, which is normally `rasdaman`.

```
log4j.appender.rollingFile.layout=org.apache.log4j.PatternLayout
log4j.appender.rollingFile.layout.ConversionPattern=%6p [%d
↪{yyyy-MM-dd HH:mm:ss}] %c{1}@%L: %m%n
```

- Select one strategy for rolling files and comment out the other. Default is rolling files by time interval.

```
# 1. Rolling files by maximum size and index
#log4j.appender.rollingFile.File=@LOG_DIR@/petascope.log
#log4j.appender.rollingFile.MaxFileSize=10MB
#log4j.appender.rollingFile.MaxBackupIndex=10
#log4j.appender.rollingFile=org.apache.log4j.RollingFileAppender

# 2. Rolling files by time interval (e.g. once a day, or once a␣
↪month)
log4j.appender.rollingFile.rollingPolicy.ActiveFileName=@LOG_
↪DIR@/petascope.log
log4j.appender.rollingFile.rollingPolicy.FileNamePattern=@LOG_
↪DIR@/petascope.%d{yyyyMMdd}.log.gz
log4j.appender.rollingFile=org.apache.log4j.rolling.
↪RollingFileAppender
log4j.appender.rollingFile.rollingPolicy=org.apache.log4j.
↪rolling.TimeBasedRollingPolicy
```

5.10.3 Security

By default only local IP addresses are allowed to make *write requests* to petascope (e.g. `InsertCoverage` and `UpdateCoverage` when importing data, or `DeleteCoverage`, etc). This is configured through the `allow_write_requests_from` setting in `petascope.properties`.

Any write requests from a non-listed IP address will be blocked. However, if one has a rasdaman user credentials with `RW` rights (see *user rights*), then one can send write requests with these credentials via basic authentication header. This authentication mechanism is used by the

WSClient for example when logged in with the petascope admin credentials, to enable deleting coverages, updating metadata, styles, etc.

Note: Since v10+, the *petascope admin user* configured in `petascope.properties` by settings `petascope_admin_user` and `petascope_admin_pass` has no effect. One must use credentials of a rasdaman user with `RW` rights to perform a request with the basic header authentication method.

5.10.4 Meta Database Connectivity

Non-array data of coverages (here loosely called metadata) are stored in another database, separate from the rasdaman database. This backend is configured in `petascope.properties`.

As a first action it is highly recommended to substitute {db-username} and {db-password} by some safe settings; keeping obvious values constitutes a major security risk.

Note that the choice is exclusive: only one such database can be used at any time. Changing to another database system requires a database migration which is entirely the responsibility of the service operator and involves substantially more effort than just changing these entries; generally, it is strongly discouraged to change the meta database backend.

If necessary, add the path to the JDBC jar driver to `petascope.properties` using `metadata_jdbc_jar_path` and `spring.datasource.jdbc_jar_path`.

Several different systems are supported as metadata backends. Below is a list of `petascope.properties` settings for different systems that have been tested successfully with rasdaman.

Postgresql (default)

The following configuration in `petascope.properties` enables PostgreSQL as metadata backend:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/petascopedb
spring.datasource.username={db-username}
spring.datasource.password={db-password}
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

HSQLDB

The following configuration in `petascope.properties` enables HSQLDB as metadata backend:

```
spring.datasource.url=jdbc:hsqldb:file://{path-to-petascopedb}/
↪petascopedb.db
spring.datasource.username={db-username}
spring.datasource.password={db-password}
```

H2

The following configuration in `petascope.properties` enables H2 as metadata backend:

```
spring.datasource.url=jdbc:h2:file://{path-to-petascopedb}/
↪petascopedb.db;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username={db-username}
spring.datasource.password={db-password}
```

5.10.5 petascope Standalone Deployment

The petascope Web application can be deployed through any suitable servlet container, or (recommended) can be operated standalone using its built-in embedded container. The embedded variant is activated through setting `java_server=embedded` in `$RMANHOME/etc/petascope.properties`.

To configure embedded mode, the following options will need to be checked and adjusted:

- `petascope.properties`

  ```
  java_server=embedded
  server.port=8080
  # a path writable by the rasdaman user
  log4j.appender.rollingFile.File=/opt/rasdaman/log/
  ↪petascope.log
  # or
  log4j.appender.rollingFile.rollingPolicy.
  ↪ActiveFileName=/opt/rasdaman/log/petascope.log
  ```

- `secore.properties`

  ```
  # paths writable by the rasdaman user
  secoredb.path=/opt/rasdaman/data/secore
  log4j.appender.rollingFile.File=/opt/rasdaman/log/
  ↪secore.log
  log4j.appender.rollingFile.rollingPolicy.
  ↪ActiveFileName=/opt/rasdaman/log/secore.log
  ```

In the standalone mode petascope can be started individually using the central startup/shutdown scripts of rasdaman:

```
$ sudo -u rasdaman start_rasdaman.sh --service petascope
$ sudo -u rasdaman stop_rasdaman.sh --service petascope
```

The Web application can be even be started from the command line:

```
$ java -jar rasdaman.war [ --petascope.confDir={path-to-etc-dir} ]
```

The port required by the embedded tomcat will be fetched from the `server.port` setting in `petascope.properties`. Assuming the port is set to 8080, petascope can be accessed via

URL `http://localhost:8080/rasdaman/ows`.

5.10.6 Serving Static Content

Serving external static content (such as HTML, CSS, and Javascript) residing outside `rasdaman.war` through petascope can be enabled with the following setting in `petascope.properties`:

```
static_html_dir_path={absolute-path-to-index.html}
```

with an absolute path to a directory readable by the user running petascope. The directory must contain an `index.html`, which will be served as the root, ie: at URL `http://localhost:8080/rasdaman/`.

5.10.7 Logging

Configuration file `petascope.properties` also defines logging. The log level can be adjusted in verbosity, log file path can be set, etc. Tomcat restart is required for new settings to become effective.

The user running Tomcat (`tomcat` or so) must have write permissions to the `petascope.log` file specified if `java_server=external`; usually the file should be placed in the Tomcat log directory in this case, e.g. `/var/log/tomcat/petascope.log`.

Otherwise, if `java_server=embedded`, then the user running rasdaman must have write permissions to the specified log file; usually the file would be placed in the rasdaman log directory in this case, e.g. `/opt/rasdaman/log/petascope.log`.

5.11 Geo Service Standards Compliance

rasdaman community is OGC WCS reference implementation and supports the following conformance classes:

- OGC CIS:
  - CIS 1.0:
  - Class GridCoverage
  - Class RectifiedGridCoverage
  - Class ReferenceableGridCoverage
  - Class gml-coverage
  - Class multipart-coverage
  - CIS 1.1:
  - Class grid-regular

  - Class grid-irregular
  - Class gml-coverage
  - Class json-coverage
  - Class other-format-coverage
- OGC WCS
  - WCS 2.0:
  - WCS Core
  - WCS Range Subsetting
  - WCS Processing (supporting WCPS 1.0)
  - WCS Transaction
  - WCS CRS
  - WCS Scaling
  - WCS Interpolation
  - WMS 1.3.0:
    - all raster functionality, including SLD `ColorMap` styling

Note: With WCS 2.1, petascope provides an additional proprietary parameter to request CIS 1.0 coverages to be returned as CIS 1.1 coverages. This is specified by adding parameter `outputType=GeneralGridCoverage`.

CHAPTER

SIX

CRS MANAGEMENT

6.1 Introduction

SECORE (Semantic Coordinate Reference System Resolver) is a server which resolves CRS URLs into full CRS definitions represented in GML 3.2.1. The implementation constitutes the official resolver of OGC, accessible under `http://www.opengis.net/def/crs/`

SECORE accepts axis, CRS, and CRS template identifiers as input URLs in GET-KVP and RESTful syntax. Further, it accepts general XQuery requests on its CRS database. It is accessible at the following service endpoint:

- `http://www.opengis.net/def/axis` for Axis Identifier URLs
- `http://www.opengis.net/def/crs` for CRS Identifier URLs and CRS Template URLs
- `http://www.opengis.net/def/crs-compound` for Compound CRS URLs
- `http://www.opengis.net/def/equal` for semantic CRS URL comparison
- `http://www.opengis.net/def/crs-query` for general XQuery requests

If deployed locally, then substitute the official opengis.net part with localhost, or your own domain.

6.2 Service

SECORE stores and queries XML data in a BaseX XML database. On the disk this database is stored in `$CATALINA_HOME/webapps/secoredb`, this is the directory where the Tomcat process will typically have write access. The database is created and maintained automatically, so no action by the user is required regarding this.

There are two types of definition collections:

- `gml` collection which is fixed and cannot be modified; this is based on the EPSG dictionary.

Note: SECORE (`def.war`) is bundled with several EPSG dataset version, however they may get outdated. For this reason it is possible to manually download the latest EPSG dataset and add it to SECORE as follows:

1. Create a new directory in the `secoredb` directory (usually in the same directory as `def.war`, e.g. `/var/lib/tomcat/webapps/secoredb`) named `gml_number.number(.number)?).` 2. Extract the EPSG GML dictionary in this folder. 3. Restart Tomcat to allow SECORE to load the new EPSG version.

For example, for EPSG version 9.5.2 it's necessary to create directory `secoredb/gml_9.5.2` and put the extracted `GmlDictionary.xml` file into it.

- `user` collection where users can add/update/delete definitions.

Each definition has an **identifier** which is checked when updating/deleting a particular definition. When inserting a definition, its identifier must not exist in SECORE already.

```
<gml:identifier codeSpace="EPSG">
  http://www.opengis.net/def/crs/EPSG/0/4326
</gml:identifier>
```

Note: Until v9.7, version 0 in CRS identifiers referred to EPSG dataset version 8.5. From v9.7, version 0 in CRS identifiers refers to the **latest** EPSG dataset version. This means that the CRS definition corresponding to a URL such as `http://localhost:8080/def/crs/EPSG/0/4326` is dynamic and may change with new releases of the EPSG dataset. For this reason it is now recommended to specify an exact version, e.g. 9.4.2 instead of 0.

6.2.1 User interface

The SECORE database tree can be viewed and (upon login) modified via graphical web interface at `http://your.server/def/index.jsp`.

More generally, any folder and definition can turn to EDIT mode by appending a `/browse.jsp` to its URI; e.g.

- `http://your.server/def/uom/EPSG/0/9001/browse.jsp` will let you view/edit EPSG:9001 unit of measure, whereas
- `"http://your.server/def/uom/EPSG/0/browse.jsp"` will let you either remove EPSG UoM definitions or add a new one, not necessarily under the EPSG branch: the `"gml:identifier"` of the new definition will determine its position in the tree.

In this document you can find hints on how to to define new GML definitions of CRSs. Mind that compounding is achieved at resolve-time by querying SECORE with a `"crs-compound"` path, so that only single CRS definitions should be added.

With regard to parametrized CRSs, you should mind that relative XPaths are not allowed (either start with `/` or `//` when selecting nodes); non-numeric parameters must be embraced by single or double *quotes* both when setting optional default values in the definition or when setting custom values in the URI.

6.2.2 Configuration

The SECORE configuration can be found in `$RMANHOME/etc/secore.properties`; editing this file requires restarting Tomcat.

Security

You should set the `secore_admin_user`, `secore_admin_pass` options to prevent unauthorized users from editing CRS definitions in the `userdb` CRS collection. If these are not set or commented out, then the admin pages have public access.

Standalone deployment

Instead of running SECORE in an external Tomcat (the default way), you can run it through its embedded Tomcat which is included inside the SECORE java web application (`def.war`). To do this, you need to change the `java_server` option to `embedded`, and change the `server.port` to a port which is not used in your system (e.g. `server.port=8082`).

Then restart rasdaman and you can access SECORE at `http://localhost:8082/def` (if `server.port=8082` has been set).

Logging

At the end of `secore.properties` you will find the logging configuration. It is recommended to adjust this, and make sure that Tomcat has permissions to write the secore.log file.

6.3 Concepts

6.3.1 CRS templates

CRS templates are concrete definitions targeted by parameterized CRSs where one or more named parameters allow the customization of one or more elements in the template itself. As such, they describe (possibly infinite) sets of concrete CRSs.

Note: The term "parametrized" is generally avoided because it may lead to confusion with the term "parametric" in OGC Abstract Topic 2 / ISO 19111-2:2009 which has a significantly different meaning.

Parameters can be resolved through values provided in the CRS URI, or through defaults defined in the CRS Template definition. Additionally, expressions ("formulae") can be associated with a CRS Template which evaluate to values when instantiated with parameter values. All values, whether instantiated in a URL request or coming from a default or a formula, can be substituted in one or several places in the concrete CRS definition associated with the CRS Template.

Example

The following URI defines the Auto Orthographic CRS 42003 specified in sub clauses 6.7.3.4 and B.9 of WMS 1.3.0 for "meter" as unit of measure and centred at 100? West longitude and 45? North latitude:

```
http://www.opengis.net/def/crs?
  authority=OGC&
  version=1.3&
  code=AUTO42003&
  UoM=m&
  CenterLongitude=-100&
  CenterLatitude=45&
```

Note: Additional examples of not-completely-specified objects can be found in sub clauses B.7, B.8, B.10, and B.11 of the`WMS 1.3.0 spec <http://portal.opengeospatial.org/files/?artifact_id=14416>`__, and in sub clauses 10.1 through 10.3 of OGC 05-096r1 (GML 3.1.1 grid CRSs profile).

Structure

Formally, a CRS Template is a GML document with root `crsnts:AbstractCRSTemplate`. It contains an element `crsnts:CrsDefinition` of some instantiatable subtype of `gml:AbstractCRS` together with a list of formal parameters.

Parameters are `crsnts:Parameter` elements listed in the `crsnts:Parameters` section. A formal parameter consists of a locally unique name, an XPath target expression indicating one or a set of substitution points relative to the CRS subnode, optionally a default value, and optionally a formula. Further, each parameter has a type associated.

The `crsnts:value` element contains a well-formed formula adhering to the JSR scripting syntax as specified in JSR-233 [5]. The type associated in the formula's `crsnts:Parameters` element denotes the result type of the expression. Names are enclosed in `${` and `}`; when used in a formula they shall contain only references to parameter names defined in the same CRS Template, and no (direct or indirect) recursive references across formulae.

Note: In particular, a formula cannot have its own parameter name as a free parameter. The target expression in crsnts:target indicates the places where, during request evaluation, the resulting parameter (obtained from URL input, or formula evaluation, or by using the default) gets applied to the CRS definition, assuming crsnts:CrsDefinition as the relative document root for XPath evaluation.

Example

The following XML snippet defines a geodetic Parametrized CRS with formal parameter x substituting parameter values in all (fictitious) axisName elements appearing the GeodeticCRS root of the CRS definition:

```
<crsnts:ParameterizedCRS>
  <gml:identifier>...</gml:identifier>
  <gml:scope>...</gml:scope>
  <crsnts:parameters>
    <crsnts:parameter name="lon" >
      <crsnts:value>90</crsnts:value>
      <crsnts:target>//longitude | //Longitude</crsnts:target>
    </crsnts:parameter>
    <crsnts:parameter name="zone">
      <crsnts:target>//greenwichLongitude</crsnts:target>
      <crsnts:value>
        min(floor((${lon} + 180.0) / 6.0) + 1,60)
      </crsnts:value>
    </crsnts:parameter>
  </crsnts:parameters>
  <crsnts:targetReferenceSystem
    xlink:href="http://www.opengis.net/def/crs/EPSG/0/4326"/>
</crsnts:ParameterizedCRS>
```

Resolution

The result of a URI request against a Parametrized CRS depends on the degree of parameter matching; it is GML document with its root being an instantiatable subtype of either `gml:AbstractCRS` or `crsnts:AbstractCRSTemplate`. The response is:

- In case all formal parameters in the Parametrized CRS addressed are matched: a CRS definition where all parameters matched are resolved.

  Example. Assuming that the name of the above Parametrized CRS example is my-own-crs, a possible instantiation of this CRS to a concrete CRS Identifier is

  ```
  http://www.opengis.net/def/crs/my-own-crs?lon=47.6
  ```

  The response to this instantiation is

  ```
  <gml:GeodeticCRS>
    ...
  <gml:GeodeticCRS>
  ```

- In case not all parameters are matched: a Parametrized CRS where all parameters matched are resolved, their corresponding crsnts:Parameter is removed, and only the non-matched parameters remain in the template.

Example. Assuming the same example as above, the CRS itself can be obtained through

```
http://www.opengis.net/def/crs/my-own-crs
```

The response to this request is

```
<crsnts:ParameterizedCRS>
  <gml:identifier>...</gml:identifier>
  <gml:scope>...</gml:scope>
  <crsnts:parameters>
    ...
  </crsnts:parameters>
  <crsnts:targetReferenceSystem xlink:href="..."/>
</crsnts:ParameterizedCRS>
```

6.3.2 CRS equality

It is possible that one and the same CRS, axis, etc. is identified by a number of syntactically different URLs, and it is not straightforward for applications to decide about equivalence of two given URIs. To remedy this, a comparison predicate is available in SECORE. A request sent to URL

```
http://www.opengis.net/def/crs-equal?1=A&2=B
```

containing two URLs A and B listed as GET/KVP parameters with names 1 and 2, respectively, will result in a response of true if and only if both URLs identify the same concept, and false otherwise; the response is embedded in an XML document.

Example

Comparing EPSG codes 4327 and 4326 can be done with this URL:

```
http://www.opengis.net/def/equal?
  1=http://www.opengis.net/def/crs/EPSG/0/4327
 &2=http://www.opengis.net/def/crs/EPSG/0/4326
```

The response will look like this:

```
<crsnts:comparisonResult xmlns='http://www.opengis.net/crs-nts/1.0'>
  <crsnts:equal>false</crsnts:equal>
  <crsnts:reason>
    <![CDATA[ ...description text... ]]>
  </crsnts:reason>
</crsnts:comparisonResult>
```

6.3.3 Directly Querying SECORE

An XQuery GET or POST request sent to URL http://www.opengis.net/def/crs-query will result in a document obtained from evaluating the XQuery request according to the XQuery standard.

CHAPTER

SEVEN

RRASDAMAN GUIDE

RRasdaman is an R package providing database interface for rasdaman.

7.1 Prerequisites

- Installed and running rasdaman (see *Installation and Administration Guide*).
- *Installed RRasdaman package*
- Familiarity with R. This tutorial is an introduction into the RRasdaman package, not R itself. To learn more about the R language, start from the official Introduction to R.
- Familiarity with rasql, the rasdaman query language. Please refer to the *Query Language Guide*.

7.2 Connect to rasdaman

The first thing to do is to establish a connection to the rasdaman database from R:

```
library(RRasdaman)
conn <- dbConnect(Rasdaman())
```

Connection is the object which allows to send queries to database and to manage transactions. One can also specify host, port, username and password which should be used for the connection. Additionally, connections could be read-only and writable. The full signature of the dbConnect method and its default arguments are the following:

```
dbConnect(drv, host = "localhost", port = 7001,
        dbName = "RASBASE", user = "rasguest",
        password = "rasguest", mode = CONN_READ_ONLY)
```

Do not forget to close your connection with dbDisconnect(conn) after the work is done! You always can list all open connections with . Here is an example on how to create several connections and close them in one command:

```
> conn1 <- dbConnect(Rasdaman())
> conn2 <- dbConnect(Rasdaman())
> dbListConnections(Rasdaman())
[[1]]
An object of class "RasdamanConnection"
Slot "jObj":
[1] "Java-Object{RasConnection{ host=localhost, port=7001,␣
↪db=RASBASE, user=rasguest, mode=READ_ONLY, alive=true }}"

[[2]]
An object of class "RasdamanConnection"
Slot "jObj":
[1] "Java-Object{RasConnection{ host=localhost, port=7001,␣
↪db=RASBASE, user=rasguest, mode=READ_ONLY, alive=true }}"

> lapply(dbListConnections(Rasdaman()), dbDisconnect)
> dbListConnections(Rasdaman())
list()
```

7.3 Handles

After the connection is established, we can send our first query and read some data from rasdaman. Let's compute the average temperature over the whole observation period from the "climate_temperature" dataset:

```
handles <- dbGetQuery(conn, "select avg_cells(x) from climate_
↪temperature as x")
length(handles)
# [1] 1
simplify(handles[[1]])
# [1] 272.1054
```

First, we send a query to rasdaman with `dbGetQuery` in rasql language. The result is a list of *handles* for each result of query execution over arrays in the collection. A handle represents the query execution result which is not yet converted into R data representation. In our case there is only one array in the collection "climate_temperature", so we have only one result, so we have only one handle.

Then we convert the returned handle into an R object with the `simplify` method and can see the scalar result.

7.4 Arrays representation

Now let's look what happens if we retrieve some array, not the single scalar value.

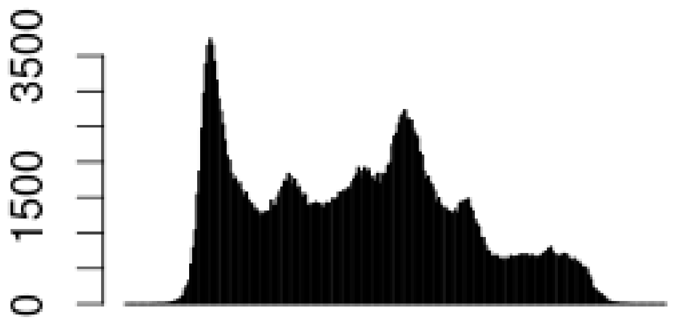
```
> handles <- dbGetQuery(conn, "select (RGBPixel) x[0:1,0:1] from␣
↪rgb as x")
> obj <- simplify(handles[[1]])
> obj
An object of class "RasdamanArray"
Slot "array":
$red
  [,1] [,2]
[1,]  169  168
[2,]  169  168
$green
  [,1] [,2]
[1,]  210  209
[2,]  210  209
$blue
  [,1] [,2]
[1,]  212  211
[2,]  216  213

Slot "origin":
[1] 0 0
```

As we can see, the result is an object of S4 class `RasdamanArray`. It has two slots: array and origin, which could be accessed as `obj@array` and `obj@origin`. The origin of an array is the coordinates of its lowermost leftmost cell. The `obj@array` slot is a list of N-D matrices, each matrix represents one of N attribute. For example, one can use both `obj@array$green` and `obj@array[[2]]` to access the green channel values, as green is the second component of the RGB structure.

7.5 Displaying data

Rasdaman has a flexible query language, so in many cases it is possible to compute sophisticated expressions on the server side rather than doing them in R. For example, one can fetch the full dataset into R session and build the histogram over its values, but it is also possible compute histogram with rasql. The second approach requires less data to be transmitted between the rasdaman server and the client application, and the client needs less memory to store it. In the following example we build the histogram of pixel intensities:

```
handles <- dbGetQuery(conn, "select marray n in [0:255] values␣
↪count_cells(c = n[0]) from lena as c")
data <- simplify(handles[[1]])
values <- data@array[[1]]
barplot(values)
```



7.6 Writing data

If one wants to write some data to rasdaman, a connections with write permissions needs to be created:

```
conn <- dbConnect(Rasdaman(), user="rasadmin", password="rasadmin",␣
↪mode=CONN_READ_WRITE)
```

We can create a collection with regular rasql syntax. Let's create a collection `images` of 2-D RGB data:

```
dbGetQuery(conn, "create collection images RGBSet")
```

Now we need data to be inserted into the newly created collection. Let's generate some image of size 20x20 with smooth gradient from red to blue. The origin point (i.e. leftmost lowermost coordinate of the array) will be `[0,0]`.

```
n <- 20
m <- 20
red <- array(seq(247,0, length.out=n), c(n, m))
green <- array(0, c(n, m))
blue <- array(seq(0, 247, length.out=n), c(n, m))
origin <- as.integer(c(0,0))
arr <- RasdamanArray(list(red=red, green=green, blue=blue), origin)
```

This generated array can be inserted into the collection with method `dbInsertCollection`:

```
dbInsertCollection(conn, name="images", value=arr, typename=
↪"RGBImage")
# [1] 420865
```

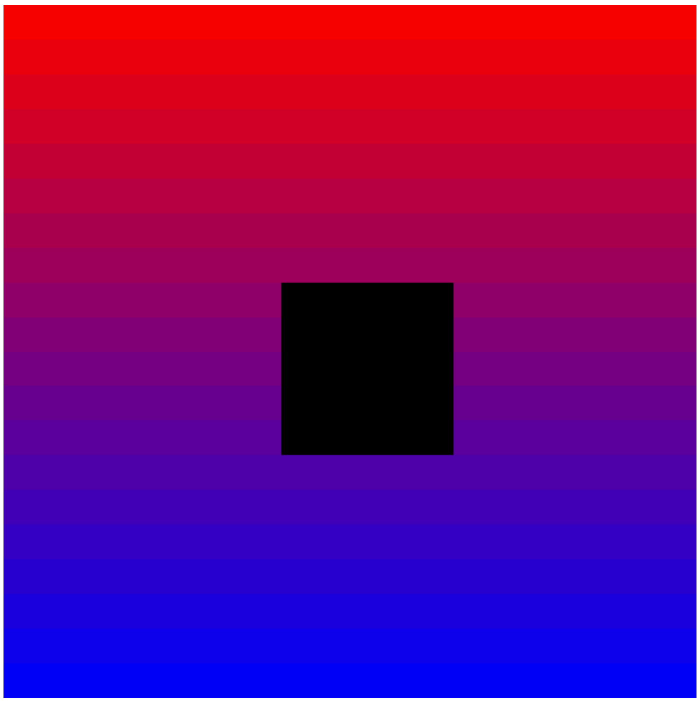
We need to specify collection name, array to be inserted and the type of the array. The method returns one number – the object identifier of the newly inserted array.

Now let's update the inserted data. We will set to zero all values in the middle of the newly inserted array. For this we need to create an array with new values.

```
zeroes <- array(0, c(5,5))
origin = as.integer(c(8,8))
updArray <- RasdamanArray(list(red=zeroes, green=zeroes,␣
↪blue=zeroes), origin)
dbUpdateCollection(conn, name="images", value=updArray, typename=
↪"RGBImage")
```

After the collection is updated, we can read the values, display the result and close the connection:

```
handles <- dbReadCollection(conn, "images")
result <- simplify(handles[[1]])

library(grid)
picture <- rgb(result@array[[1]], result@array[[2]],␣
↪result@array[[3]], max=255)
dim(picture) <- dim(result)
grid.raster(picture, interpolate=F)
dbDisconnect(conn)
```



The list of allowed type names could be obtained with a rasql query, see *Type Definition Using rasql*.

7.7 Transaction management

The methods `dbCommit` and `dbRollback` could be used for transaction management. There is no need to explicitly specify beginning of the transaction. The method `dbDisconnect` commits the transaction before closing the connection.

7.8 Further reading

You can type `?RRasdaman::RRasdaman` and `help.search("RRasdaman")` at any time from R prompt to see more package documentation.

CHAPTER

EIGHT

C++ DEVELOPERS GUIDE

8.1 Preface

8.1.1 Overview

This guide provides information about how to use the rasdaman database system (in short: rasdaman). The booklet explains usage of raslib, the rasdaman API, through its C++ binding.

Follow the instructions in this guide as you develop your application which makes use of rasdaman services. Explanations detail how, from within a C++ program, to create databases, collections, and instances; how to retrieve from databases; how to manipulate and delete instances within databases; how to influence physical storage parameters; how to do transaction handling and other administrative tasks.

The rasdaman interfaces are available on different operating system platforms. Although there are some differences in the way rasdaman appears in these different versions, the functionality is the same.

8.1.2 Audience

The information in this manual is intended primarily for application developers and for database administrators.

8.1.3 Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as *raswct*, release notes, and additional information on the rasdaman wiki.

The rasdaman Documentation Set consists of the following documents:

- Installation and Administration Guide
- Query Language Guide

- C++ Developer's Guide
- Java Developer's Guide
- raswct Developer's Guide
- the rasdaman wiki, accessible at www.rasdaman.org

8.2 Introduction

See the corresponding *Introduction* Section in the *rasdaman Query Language Guide*.

8.3 Terminology

See the corresponding *Terminology* Section in the *rasdaman Query Language Guide*.

8.4 Application Examples

The following sections contain three examples of using the rasdaman API. Every example contains a code fragment including variable declarations and definitions, database open/close and transaction begin/commit statements. The numbers at the beginning of the code lines are used as references in the explaining text. Code segments which are in more than one example are explained where they occur first. For more clarity, error handling was omitted.

As raslib heavily makes use of templates, some platform specifics have to be considered when compiling and linking application programs. These are collected in *Compilation and Linkage of Client Programs*.

For details on the operational semantics of the rasdaman data model the reader is strongly encouraged to study the *rasdaman Query Language Guide*.

8.4.1 Basic Application Program Structure

Operation sequence

In order to access data in a database, variables have to be first defined and initialized, the database has to be opened and a transaction started. In the end, the transaction has to be committed and the database closed. Hence, an application basically consists of the following components (sample C++ code interspersed as far as rasdaman access is concerned):

- Declaration and definition of database and transaction variables and other data like images or image sets

```
r_Database database;
r_Transaction transaction{&database};
```

- Set the server name using the default port 7001.

```
database.set_servername( "ServerName" );
```

- Set user identification.

```
database.set_useridentification( "me", "myPassword" );
```

- Open the database.

```
database.open( "DatabaseName" );
```

- Begin the transaction.

```
transaction.begin();
```

- Work with the database.
- Commit the transaction.

```
transaction.commit();
```

- Close the database.

```
database.close();
```

Synchronous query execution

When a query is sent to the rasdaman server it will be executed in completeness - a running query cannot be aborted[1]. Care should be taken therefore not to start queries requiring resources beyond the capability of the server hardware and software environment, as the rasdaman service may be blocked for an indefinite time period.

8.4.2 Insertion of MDD

The following example creates a new MDD set with the name ULongSet and inserts two images into this set. The first image is initialized with zero, and the second one by way of some assumed initialization function.

(1) The variable declaration part includes one instance of type `r_Database` to represent the database and one instance of type `r_Transaction` to serve for the transaction handling. The domain of type `r_Minterval` is used for specifying the spatial domain of the images. In order to hold a persistent image, image has to be declared as an `r_Ref` pointer on the `r_Marray` structure. The same applies for `image_set` which is an `r_Ref` to the set of images.

[1] This has nothing to do with transactions - after each completion of a query, the embracing transaction can be aborted indeed.

```
r_Database database;
r_Transaction transaction{&database};
r_Minterval domain;
r_Ref< r_Marray<r_ULong> > image;
r_Ref< r_Set< r_Ref< r_Marray<r_ULong> > > > image_set;
```

(2) Server name and password are set (see *Class r_Database*).

```
database.set_servername( "MyServer" );
database.set_useridentification( "me", "myPassword" );
```

(3) An open message with the database name is sent to the database object.

```
database.open( "MyDatabase" );
```

(4) The transaction is opened using the transaction object.

```
transaction.begin();
```

(5) Memory for the image set is allocated using the new operator of class `r_Object`. As additional arguments, the new operator gets the database object in which it is to be inserted and the type name which was created in the database (see *Type Definition Using rasql*).

```
image_set = new( &database, "ULongSet" )
            r_Set< r_Ref< r_Marray<r_ULong> > >(&transaction);
```

(6) To give a name to the set for later retrieval, a `set_object_name` message is sent to the database object.

```
database.set_object_name(*image_set, "MyCollection");
```

(7) The spatial domain `domain` of the first is initialized with `[1:91,1:91]`. For doing so, a temporary two-dimensional object of type `r_Minterval` is filled with `r_Sintervals` specifying lower and upper bounds per dimension and then gets assigned to `domain`.

```
domain = r_Minterval(2) << r_Sinterval( 01, 91 )
                        << r_Sinterval( 01, 91 );
```

(8) Memory for a persistent object of type `r_Marray` is allocated using the new operator of `r_Ref`. Again, the new operator gets the current database and the type name of the MDD object (creation of types is described in *Type Definition Using rasql*). The constructor of `r_Marray` gets the value zero which is used for initializing the whole MDD.

```
image = new( &database, "ULongImage" )
        r_Marray<r_ULong>( domain, 0ul, &transaction );
```

(9) The image created in (7) is now inserted into the set. From now on, the persistent object is accessible via the collection.

```
image_set->insert_element( image );
```

(10) The second image is created with a function pointer as second argument for the `r_Marray` constructor. The function must be of type `r_ULong (*initFunction)(const r_Point& pt)`. The function is invoked for each cell of the MDD with the current coordinates of the cell passed as the pt argument. The result value of type `r_ULong` is taken for the initial value of the cell.

```
image = new( &database, "ULongImage" )
        r_Marray<r_ULong>( domain, &initWithCoordinates, &
↪transaction );
```

(11) The image created in (9) is inserted into the set.

```
image_set->insert_element( image );
```

(12) The transaction is committed. At this time, the image set is created in the database and the images are inserted. The data is made persistent and becomes visible to other transactions. The transient memory used to store the image on client side is freed and pointers to these objects (`image_set` and `image`) become invalid.

```
transaction.commit();
```

(13) The last statement closes and disconnects the database.

```
database.close();
```

For completeness, the following code segment describes the function used for initializing each cell of an MDD with the coordinates `x+256*y`:

```
r_ULong initWithCoordinates( const r_Point& pt )
{
    r_ULong value = pt[0] + pt[1] * 0x100;
    return value;
}
```

8.4.3 Lookup of an MDD set by its name

This example demonstrates retrieval of a set containing MDD objects as elements and iteration through the retrieved result set using raslib.

(1) An iteration variable named `iterator` is defined. It needs the element type of the collection being iterated as template argument.

```
r_Database database;
r_Transaction transaction{&database};
r_Ref< r_Set< r_Ref< r_GMarray > > > image_set;
r_Ref< r_GMarray > image;
r_Iterator< r_Ref< r_GMarray > > iter;
```

(2) A read-only transaction is started for the retrieval query. Read-only transactions should be used whenever possible, i.e., when no update operations occur within this transaction, in order to have maximal performance.

```
database.set_servername( "ServerName" );
database.set_useridentification( "me", "myPassword" );
database.open( "DatabaseName" );
transaction.begin( r_Transaction::read_only );
```

(3) The set is retrieved by sending a `lookup_object` message with the set name to the database object. At this moment, just a set of object identifiers is sent back to the client.

```
image_set = database.lookup_object("CollectionName");
```

(4) The statement creates an iteration variable pointing to the first element of the set.

```
iter = image_set->create_iterator();
```

(5) A simple `for` loop is used for iterating through the collection. An element of the collection, which is an r_Ref pointer to the MDD object, can be accessed by dereferencing the iteration variable `iter`. The image itself is retrieved from the server when the r_Ref pointer is dereferenced for the first time.

```
for (iter.reset(); iter.not_done(); iter++)
{
    image = (*iter);
    // work with the image
    // for example print its spatial domain
    cout << image->spatial_domain() << endl;
}
```

(6) The query result is valid only until transaction end.

```
transaction.commit();
database.close();
```

8.4.4 Invocation of RasML statements

This example shows the creation and invocation of RasML queries using the raslib classes:

(1) Two domains, a collection name, and a threshold value are defined to use them at creation stage of the RasML query.

```
r_Minterval select_domain = r_Minterval("[0:4,0:4]");
r_Minterval where_domain = r_Minterval("[8:9,8:9]");
char collection_name[] = "CollectionName";
r_ULong threshold_value = 10;

r_Database database;
```

(continues on next page)

(continued from previous page)

```
r_Transaction transaction{&database};
r_Set< r_Ref< r_GMarray > > image_set;
r_Ref< r_GMarray > image;
r_Iterator< r_Ref< r_GMarray > > iter;

database.set_servername( "ServerName" );
database.set_useridentification( "me", "myPassword" );
database.open( "DatabaseName" );
```

(2) A read-only transaction is started for the retrieval query. Read-only transactions should be used whenever possible, i.e., when no update operations occur within this transaction, in order to have maximal performance.

```
transaction.begin( r_Transaction::read_only );
```

(3) The query object of type `r_OQL_Query` is created and initialized with the parameterized query string.

```
r_OQL_Query query( "select a$1 from $2 as a where some_cells( a
↪$3 > $4 )" );
```

(4) The query parameters are filled using stream operators on the query object. First, the domain of type `r_Minterval` for the select part is applied, then the collection name, the domain for the where clause, and the threshold value are inserted.

```
query << select_domain << collection_name << where_domain <<␣
↪threshold_value;
```

The resulting query string looks like follows:

```
select a[0:4,0:4]
from CollectionName as a
where some_cells( a[8:9,8:9] > 10 )
```

(5) Finally, the query is executed using the global function `r_oql_execute`. The query result is returned in the call-by-reference parameter `image_set`. As query results are transient, the data of the whole result is sent to the client at this point.

```
r_oql_execute( query, image_set, &transaction );
iter = image_set.create_iterator();
for( iter.reset(); iter.not_done(); iter++ )
{
    image = (*iter);
    // work with the image
}
transaction.commit();
database.close();
```

8.5 Raslib Classes

8.5.1 Overview

The raslib classes represent the rasdaman programming interface. It relies on the ODMG standard with some extensions supporting a smooth integration of the rasdaman-specific array structures into the conventional C++ programming model.

raslib classes are categorized in

- *Type Classes* providing type information for MDD objects,
- *Object Classes* for handling persistent MDD objects,
- *System Classes* for general tasks such as session maintenance and database querying,
- *Schema Access Classes* to get runtime type information,
- *Storage Layout Classes* for handling the storage structure, and
- *Error Classes* for error handling.

8.5.2 Type Classes

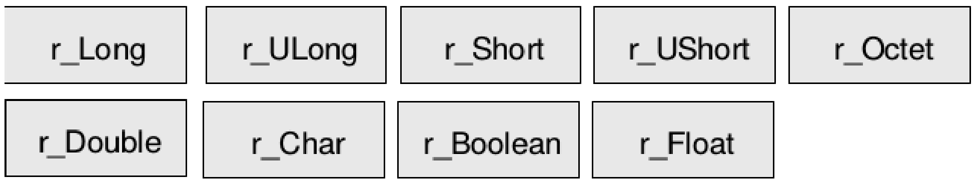


Figure 8.1: Primitive Types

The types `r_Long`, `r_ULong`, `r_Short`, `r_UShort`, `r_Octet`, `r_Char`, `r_Boolean`, `r_Float`, and `r_Double` are atomic, serving as base types for MDD objects (Figure 8.1). Composite types built from atomic (primitive) or other complex (structured) types are built using the record (struct) constructor.

Complex numbers, while by nature equivalent to a record structure `{float re,im;}`, are provided as a built-in type. Type complex implements complex numbers on single-precision float components while `complexd` implements double-precision.

Null values, i.e., values of cells which have not been assigned a value yet, always are the numerical zero value of the corresponding type. This extends in the obvious way to composite cells.

Table 8.1: Correspondence between rasql and C++ types

rasql	C++ binding	Length	Description
`octet`	`r_Octet`	8 bit	signed integer
`char`	`r_Char`	8 bit	unsigned integer
`short`	`r_Short`	16 bit	signed integer
`unsigned short`	`r_Ushort`	16 bit	unsigned integer
`long`	`r_Long`	32 bit	signed integer
`unsigned long`	`r_Ulong`	32 bit	unsigned integer
`float`	`r_Float`	32 bit	single precision floating point
`double`	`r_Double`	64 bit	double precision floating point
`boolean`	`r_Boolean`	1 bit[2]	true (nonzero value) false (zero value)
`complex`	`r_Complex`	64 bit	Single precision complex number
`complexd`	`r_Complex`	128 bit	Double precision complex number

8.5.3 Object Classes

Object Classes are used for the data exchange with the database. They consist of classes able to generate and handle persistent arrays, i.e., arrays stored in a database, intervals, multidimensional intervals, multidimensional points, and scalar data which can either be atomic (primitive) or complex (structured). Figure 8.2 shows the object classes provided by rasdaman.

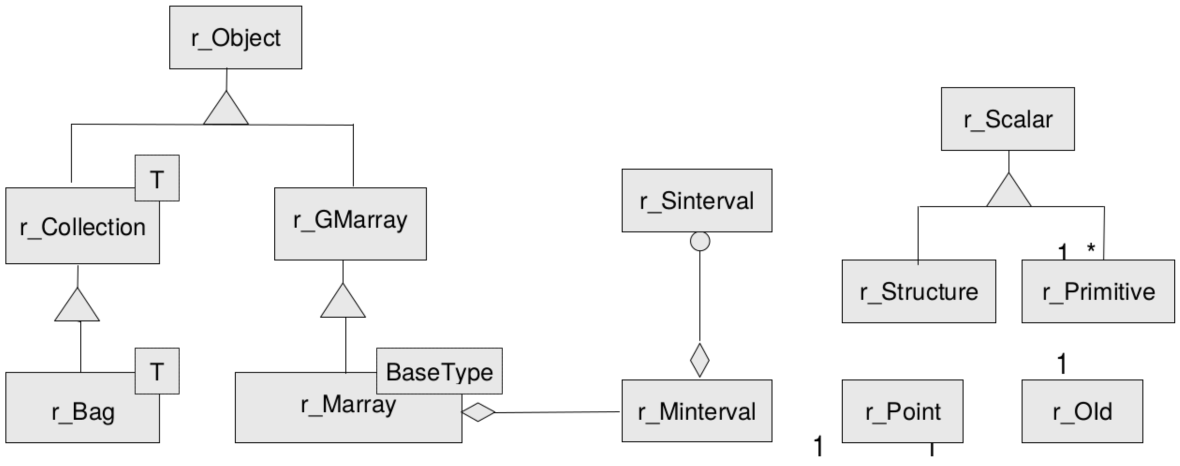


Figure 8.2: Object Classes

[2] memory usage is one byte per pixel

Class `r_Point`

Class `r_Point` handles multidimensional points.

Example

```
r_Point pointname( 5, 4 );
```

Class `r_Sinterval`

Class `r_Sinterval` represents a one-dimensional interval with lower and upper bound. Both bounds can either be fixed or variable (indicated by an asterisk '*'). Operations on intervals are defined following conventional interval arithmetics.

Example

```
r_Sinterval(100L, 200L)
```

specifies the interval [100:200].

Class `r_Minterval`

The spatial domain of an MDD is represented by an object of class `r_Minterval` ("multidimensional interval"). It specifies lower and upper bound of the point set for each dimension of an MDD. Internally, the class is implemented through an array of intervals of type `r_Sinterval`.

Example

`r_Minterval intervalname("[0:100, 0:300]");`

The object generated by the above expression yields the following output:

```
intervalname.dimension()  = 2
intervalname[0].low()     = 0
intervalname[0].high()    = 100
```

Class `r_OId`

This handles object identifiers. Every array has a unique object identifier it can be addressed with.

Class r_Object

r_Object is an abstract class. Instances can only be generated from the non abstract classes inheriting from this class, that is r_Set, r_GMarray and r_Marray<T>. All these subclasses are capable of having persistent as well as transient instances and therefore are called persistent capable classes.

Objects of these classes can be generated using the overloaded new operator:

```
void* operator new( size_t size )                           // (1)
void* operator new( size_t size, r_Database* database,
                    const char* type_name = 0 )             // (2)
void* operator new( size_t size, const char* type_name )    // (3)
```

(1) is used to create transient objects. The only argument is the size of the new object.

(2) To generate persistent instances one also has to specify the database the object is to be inserted in.

(3) is the new operator for transient objects carrying type information.

Calling the delete operator

```
void operator delete( void* obj_ptr )
```

removes the object from memory and, in case it is a persistent object, from the database.

Classes r_Marray<T> and r_GMarray

The template class r_Marray<T> represents an MDD object over cell type T. Class r_GMarray is more generic in that it is able to represent MDD objects of any base type. This is necessary, firstly, for having a generic class for query results where the base type is not known at compile time and, secondly, for composite (multi-band) types.

The template class r_Marray<T> for specific base types inherits from r_GMarray; the constructor r_Marray<T>(r_GMarray&) is provided for easy transformation to cell type safe m-arrays where the base type is known at compile time. Operations for accessing single cells are only available for r_Marray<T>.

Class r_Collection

r_Collection is an abstract class. It is the basic class of a collection. Possible subclasses are r_Set , r_Bag and r_List. The protected members isOrdered and allowsDuplicates are not initialized here, they have to be initialized in the respective subclasses. The method

```
virtual void insert_element ( const T& element, int no_modification␣
↪= 0 )
```

inserts an element into the collection. If `no_modification` is set, the `mark_modified()` method of `r_Object` is not invoked and, therefore, a modification will not be recognized at the transaction commit point.

Class `r_Set`

The class implements a set container. It inherits most of the functionality from `r_Collection`. The set can not have any duplicates and it is not ordered. The method

```
virtual void insert_element ( const T& element, int no_modification␣
↪= 0 )
```

inserts an element into the collection. If `no_modification` is set, the `mark_modified()` method of `r_Object` is not invoked and, therefore, a modification will not be recognized at the commit point.

Classes `r_Scalar`, `r_Primitive` and `r_Structure`

The subclasses of `r_Scalar` are used to represent query results of the primitive types `r_Boolean`, `r_Char`, `r_Octet`, `r_Short`, `r_UShort`, `r_Long`, `r_ULong`, `r_Float`, `r_Double` and types composed of the primitive ones.

Class `r_Primitive` supports type-safe value access methods. `r_Structure` allows to access its elements by the subscript operator [].

Examples

The following line shows access to an unsigned short value:

```
r_Primitive primitive;
// ...
r_UShort value = primitive.get_ushort();
```

A structured value consisting of three long values can be accessed as follows:

```
r_Structure structuredValue;
// ...
for( int i=0; i<structuredValue.count_elements(); i++ )
{
    value = ((r_Primitive&)structuredValue[i]).get_long();
    //...
}
```

8.5.4 System Classes

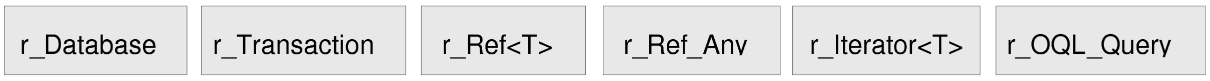


Figure 8.3: System Classes

Class `r_Database`

Class `r_Database` allows to open and close connections to a specific database. The database name and the address of a running server manager must be indicated. Further optional parameters are

- port number (default: 7001),
- access mode (read/write or read-only; by default: read-only),
- login (default: `"rasguest"`)
- password (default: `"rasguest"`).

A database object must be instantiated and opened before starting any transaction on the database, and closed after ending these transactions (with a commit or abort).

Which Server to Contact?

Note that the server/port to be indicated must address the rasdaman server *manager* (not a particular rasdaman server); if in doubt, consult your system administrator.

Example

```
r_Database database;
database.set_servername( "Server Name" );
database.set_useridentification( "login name", "passwd" );
database.open( "Database Name" );
// ...
database.close( );
```

Storage Format

The `r_Database` class also allows to set the storage format, both for storage in MDD objects in the server and for their transfer between client and server. See *Class r_Convertor and Subclasses* for details.

Class r_Transaction

To use a transaction, an object of type r_Transaction has to be instantiated with an optional r_Database object as an argument (*not* thread-safe if the database parameter is not specified). Transactions can be started either in read/write or read-only mode, committed, aborted, and checkpointed. It is important to note that all access, creation, modification, and deletion of persistent objects must be done within a transaction. In order to achieve maximal performance, read-only transactions should be used whenever possible, i.e., when no update operations occur within this transaction. Right now checkpointing is not supported.

```
r_Transaction transaction{&database};
transaction.begin( );
// ...
transaction.commit( );
```

Classes r_Ref<T> and r_Ref_Any

An instance of template class r_Ref<T> is a reference to an instance of type T and is used to reference persistent sets (r_Set<T>) and MDD objects (r_GMarray and r_Marray<T>). It behaves like a normal C++ pointer but is capable of managing persistent data of type T within a transaction. In case the r_Ref<T> pointer is dereferenced (using the operator ->) and the object it is pointing to is not in the client memory yet, it is retrieved from the server.

The class r_Ref_Any is defined to support a reference to any type. Its primary purpose is to handle generic references and allow conversions of r_Ref<T> in the type hierarchy. A r_Ref_Any object can be used as an intermediary between any two types r_Ref<X> and r_Ref<Y> where X and Y are different types. A r_Ref<T> can always be converted to a r_Ref_Any; there is a function to perform the conversion in the r_Ref<T> template. Each r_Ref<T> class has a constructor and assignment operator that takes a reference to a r_Ref_Any.

Class r_Iterator<T>

The template class r_Iterator<T> defines the generic behavior for iteration. An object of this class can be used within a for loop for iterating through a collection of MDD objects. All iterators use a consistent protocol for sequentially returning each element from the collection over which the iteration is defined. When an iterator is constructed, it is either initialized with another iterator or is set to null. When an iterator is constructed via the method r_Collection<T>::create_iterator(), the iterator is initialized to point to the first element, if there is one.

Class r_OQL_Query and the freestanding function r_oql_execute()

A query statement is represented through an object of class r_OQL_Query (see *Invocation of RasML statements*). The r_OQL_Query constructor gets a query string which optionally can be parametrized. In this case, $i indicates the i-th parameter. The $i do not have to appear in a strict order - for example, $3 may appear before $2 in the statement.

The overloaded stream operator inserts the corresponding parameter values into the query, at the same time preserving their respective types. The query object expects parameters in the sequence of $1, $2, and so on. If any of the $i is not followed by a parameter at the point r_oql_execute() is called, an r_Error exception object of kind r_Error_QueryParameterCountInvalid will be thrown.

A query is executed against an open database through invocation of the freestanding function r_oql_execute(). This overloaded function exists in four variants:

```
void r_oql_execute( r_OQL_Query & query,
                    r_Transaction* transaction = nullptr )

void r_oql_execute( r_OQL_Query & query, r_Set<r_Ref_Any>& result,␣
↪int dummy,
                    r_Transaction* transaction = nullptr );

void r_oql_execute( r_OQL_Query & query, r_Set<r_Ref<r_GMarray>> &␣
↪result_set,
                    r_Transaction* transaction = nullptr )

void r_oql_execute( r_OQL_Query & query, r_Set<r_Ref<r_Any>> &␣
↪result_set,
                    r_Transaction* transaction = nullptr )
```

The first version is used for insert (until v9.1), update, and delete statements where no result is passed back. The second version is used for insert queries, where the result contains the unique OID of the inserted object; the third parameter has no function and is there to distinguish this from the next two versions. The third version is for executing select statements where an MDD is returned; in this case, the second parameter receives the query result. The final case is for general query results which may also contain non-MDD return values, e.g., resulting from select oid(...) or select sdom(...) statements. This version will also be used when the result type of a query is not known in advance (i.e., at compile time). In this case, an r_Ref_Any object is returned, and the application is responsible for decoding the proper type. In support of this, r_Ref_Any objects contain their type information (see *Example: Dynamic Type Information of a Query Result*).

In all cases, the result_set parameter does not have to be initialised, and any previous contents is discarded by r_oql_execute().

Note: The transaction parameter is optional. If not specified, there is no guarantee on thread-safety (in fact queries will likely fail). The same holds for all other public API: if there is an r_Database or r_Transaction parameter, it is best to specify it to ensure correct usage in concurrent code.

Once a query has been executed via `r_oql_execute()`, the arguments associated with the $i parameters are cleared and new arguments must be supplied.

Example

The following code fragment creates a query string with two parameters `$1` and `$2`.

```
r_OQL_Query query1( "select a$1 from $2 as a" );
```

Now two query parameters are generated:

```
r_Minterval select_domain = r_Minterval( 2 )
                         << r_Sinterval( 100L, 199L )
                         << r_Sinterval( 0L, 149L );
char collection_name[] = "mr";
```

Next, the parameters are attached to the query using the stream operator:

```
query1 << select_domain << collection_name;
```

The resulting query string is

```
select a[ 100:199, 0:149 ] from mr as a
```

Example

The following code shows how to attach an MDD object to an insert query:

```
r_Marray<r_Char> mddObject(...);                  // (1)
r_OQL_Query query("insert into mr1 values $1");  // (2)
query << mddObject;                              // (3)
```

Explanation:

(1) A transient MDD named mdd is created.

(2) The query object of type `r_OQL_Query` is initialized with an insert query statement including a placeholder `$1`.

(3) The MDD object is attached to the parameter `$1` of the query.

8.5.5 Schema Access Classes

The rasdaman Schema Access Classes (cf. Figure 8.4) enable the user to determine the type of a query result at runtime.

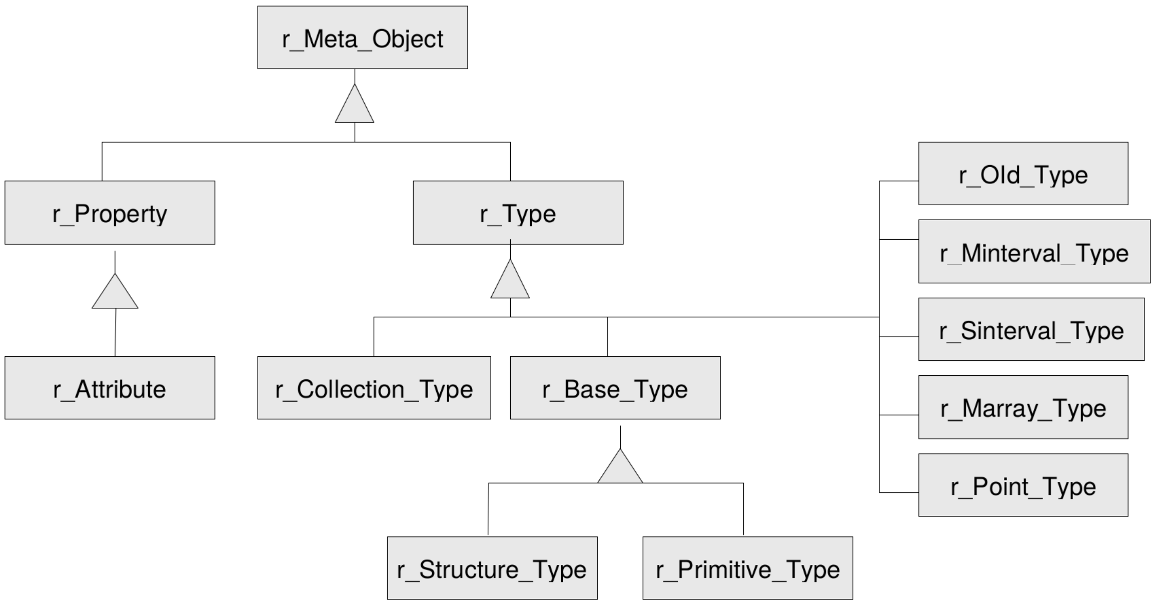


Figure 8.4: Schema Access Classes

Class `r_Meta_Object`

Instances of class `r_Meta_Object` are used to describe elements of type information. The class holds a name standing for the type name of its instances.

Class `r_Type`

`r_Type` is an abstract base class for all type descriptions. It provides runtime type information through the method `type_id()` which returns a value of type `r_Type_Id`. It is an identifier of the following list:

```
BOOL, OCTET, CHAR, SHORT, USHORT, LONG, ULONG, FLOAT, DOUBLE,
STRUCTURETYPE, MARRAYTYPE, COLLECTIONTYPE, SINTERVALTYPE,␣
↪MINTERVALTYPE,
POINTTYPE, OIDTYPE, COMPLEXTYPE1, COMPLEXTYPE2
```

Class `r_Collection_Type`

The class represents the type of a collection object. The type of the collection elements can be determined using method `element_type()`.

Class r_Base_Type

r_Base_Type is an abstract base class for all type descriptions allowed as MDD base types which can either be primitive or structured types. The method size() delivers the size of a type instance in bytes.

Class r_Primitive_Type

This class represents all primitive types in the ODMG-conformant representation of the rasdaman type system.

Class r_Structure_Type

This class represents all user defined structured types in the ODMG-conformant representation of the rasdaman type system. They are returned using the method print_status(). Members are described by r_Attribute instances and represent the state or the structure. They can be accessed using an iterator of type attribute_iterator. Structures do not have object identity.

Class r_Property

This class is an abstract base class for all elements describing the state of an application-defined type. Right now, the only subclass is r_Attribute.

Class r_Attribute

An instance of r_Attribute describes an object or a literal. An attribute has a name and a type. The name is returned by the inherited method r_Meta_Object::name(). The type description of an attribute can be obtained using the inherited method r_Property::type_of(). The method offset() gives back the byte offset of the corresponding data area within a structure. If the attribute is not defined within a structure, the offset is zero.

Example

The structure

```
struct
{
    char red;
    char green;
    char blue;
};
```

has three attributes. The name of the third one, for example, is blue, its type is char and its offset 2.

Class `r_Minterval_Type`

The class represents the type of an `r_Minterval` object.

Class `r_Sinterval_Type`

The class represents the type of an `r_Sinterval` object.

Class `r_Point_Type`

The class represents the type of an `r_Point` object.

Class `r_Marray_Type`

The class represents the type of an r_Marray object. The base type of the MDD object can be determined using the method `base_type()`.

Class `r_Oid_Type`

The class represents the type of an r_Oid object. The only meaningful comparison operations are equality and inequality of two OIDs.

Entry Points of the Type Schema

The type information can be accessed using one of the following methods:

```
const r_Type* r_Object::get_type_schema()

const r_Base_Type* r_GMarray::get_base_type_schema()

const r_Type* r_Collection::get_element_type_schema()
```

Example: Dynamic Type Information of a Query Result

In a query, new structures can be created which are not already defined in the database schema. For example, the following query forces the server to introduce an array type based on a 2-component cell structure:

```
select { img.red, img.green }
from rgb as img
```

Regardless of a result object's type being a database type or created on the fly, the type information can be accessed using the previously introduced type functions. The following - incomplete - code piece prints out the type information associated with the MDD objects of a query result.

```
r_Bag< r_Ref_Any > result_set;
// ...query preparation...
r_oql_execute( query_object, result_set );
r_Iterator< r_Ref_Any > iter = result_set.create_iterator();
for( iter.reset(); iter.not_done(); iter++, i++ )
{
    switch( result_set.get_element_type_schema()->type_id() )
    {
    case r_Type::MARRAYTYPE:
        r_Ref<r_GMarray>(*iter)->print_status( cout );
        break;
    case r_Type::POINTTYPE:
        r_Ref<r_Point>(*iter)->print_status( cout );
        break;
    // etc.
    }
}
```

Note: A result set may contain structures other than MDD, e.g., when a spatial domain or some aggregate scalar is specified in the select clause. E.g. the query

```
select sdom( a ) [0].lo
from mr as a
```

returns a set of integer values.

8.5.6 Storage Layout Classes

A specialized storage structure for MDD objects is used in secondary storage, which is designed to provide fast access to persistent MDD objects for the most typical operations on such objects. This storage structure is configurable so that it is possible to set the different parameters (*storage options*) that define it. The storage options for an MDD object should be set depending on the access characteristics expected for that object. The current version allows to configure *tiling* (i.e., the subdivision algorithm used for the MDD objects) and *storage format* (i.e., the way how MDD tiles are encoded and compressed in the database and how MDD objects are compressed for client/server transfer).

Tiling is the subdivision of the MDD object into multidimensional blocks (*tiles*) of the same dimensionality as the MDD object. A *tile* is a multidimensional subarray of an MDD object. Tiling enables fast access to parts of an MDD, since only the tiles intersected by an access are retrieved by rasdaman. Tiling may be done in different ways, resulting in tiles with different formats and sizes. For example, tiles in a two dimensional image may be squares or rectangles with different sizes (Figure 8.5).

In rasdaman, tiling is done according to a *tiling scheme*. Different tiling schemes allow the user to specify the subdivision of the domain in different ways. The choice of the tiling scheme and tiling parameters for an MDD object should be based on the most common type of access to the

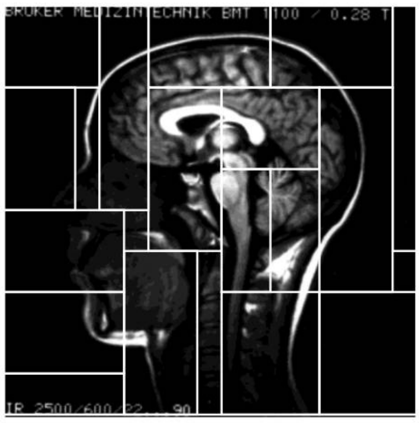


Figure 8.5: Tiling of a 2-D image.

MDD object. The following tiling schemes are provided: *aligned*, *default*, *directional*, *areas of interest* and *statistical* tiling. All tiling schemes take into account the tile size parameter, which defines the maximum size in characters for individual tiles of the MDD object.

Aligned tiling divides the object into blocks which are aligned and have the same specified format. Default tiling is the tiling scheme used in case no specific tiling scheme is specified for an MDD object. It is a multidimensional block with sizes of equal lengths along all the directions of the domain. In directional tiling, the MDD object is divided into blocks defined by a partition of the domain of the MDD along different directions of the domain. This subdivision is appropriate for objects which are accessed through selection of linear ranges along only part of the directions of the domain.

The storage format indicates how tiles of an object are stored in the database. This addresses both encoding and compression. Some encoding always has to be chosen; for compression, various alternatives are available, ranging from uncompressed storage over losslessly compressed to lossy compressed data.

An overview of the storage layout classes is given on Figure 8.6.

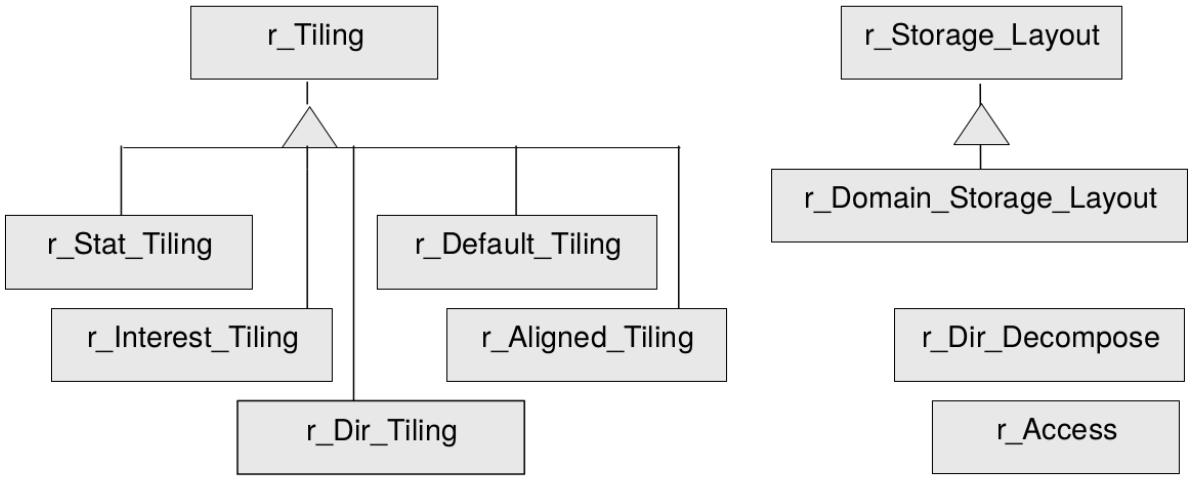


Figure 8.6: Storage Layout Classes

Class `r_Storage_Layout`

The classes of the `r_Storage_Layout` hierarchy are used to express the storage options for `r_Marray` objects. If an `r_Storage_Layout` object is passed to the `r_Marray` constructor, the options specified in it determine the structure of the object in persistent storage, otherwise, the default storage layout is used. It is important to note, however, that the notiling option of the client, activated by an environment variable, overrides the storage layout tiling options specified through `r_Storage_Layout`. If the rasdaman client is running with the option notiling, no tiling is done, independently of the storage layout chosen.

Class `r_Tiling`

Storage layout classes allow setting of the tiling option through instances of `r_Tiling` classes. When an `r_Marray` object is made persistent, in the rasdaman client the object is divided into blocks according to the tiling chosen for the object. These tiles are sent to the server and stored to constitute the MDD object. An index is built to access the tiles belonging to the MDD object.

Each derived class of `r_Tiling` implements a different decomposition method or tiling scheme. The following tiling classes are provided:

```
r_Aligned_Tiling

r_Dir_Tiling

r_Interest_Tiling

r_Stat_Tiling.
```

All these tiling schemes evaluate the tile size parameter `tile_size` which is the size of a tile in bytes. The default tile size is that specified for the rasdaman client.

Next, these tiling subclasses will be explained.

Class `r_Aligned_Tiling`

Aligned tiling is the regular tiling of an MDD object. Parameters provided are the tile format and tile size. The tile format specifies the sizes of a block along the different directions of the domain. These are interpreted as relative sizes. For example, if a `[0:0,0:1]` tile format is specified and a tile with exactly that format would have a size much smaller than the given tile size, that tile is stretched proportionally along all directions, so that the final tiles are twice as long in the second direction as in the first and have a size as close as possible to the tile size. An open interval (indicated by an asterisk “*”, see documentation for `r_Sinterval` and `r_Minterval`) along one of the directions specifies a direction of preferential access. Tiles will be made as long as possible in that direction.

Class `r_Dir_Tiling`

`r_Dir_Tiling` implements non-regular decomposition along specific directions of an MDD object. This tiling scheme allows a non-regular subdivision of the space. The user has to give the number of dimensions of the space and the decomposition wanted for each dimension.

Class `r_Dir_Decompose`

The `r_Dir_Decompose` class is used to specify a decomposition along one direction, i.e., dimension. The resulting tiling structure consists of a non-uniform grid where each grid line goes completely through the MDD and the distance between parallel gridlines is arbitrary.

An array of `r_Dir_Decompose` objects, with one element for each direction, must be provided.

Example

To specify tiling restrictions on the first two dimensions of a three-dimensional MDD object, the following code would apply:

```
r_Dir_Decompose decomp[3];
decomp[0] << 0 << 20 << 40 << 50;
decomp[1] << 0 << 15 << 20 << 50 << 60;
r_Dir_Tiling Tiling3DMDD( 3, decomp, ts );
```

ts in the last line specifies the tile size. The first and last elements put into the `r_Dir_Decompose` object must be the origin and limit of that dimension or a cross-section of the domain will occur (as if the elements outside the specification wouldn't mind). In this code example the first dimension is going from 0 to 50 and the second one from 0 to 60.

Class `r_Interest_Tiling`

The class `r_Interest_Tiling` implements the *areas of interest tiling* algorithm. The user specifies which areas are of interest (areas which are accessed very often) and tiling is performed accordingly, in order to optimize access to those areas.

Example:

If the areas `[, 50:60]` and `[, 65:70]` are of interest in the `[0:1000,0:1000]` domain, the following code does specification:

```
{
    // ...
    r_Minterval domain( "[0:1000,0:1000] ");
    r_Minterval interest1( "10:20,50:60] ");
    r_Minterval interest2( "[18:50,65:70] ");
    std::vector< r_Minterval > interest_areas;
    interest_areas.insert_element( interest1 );
    interest_areas.insert_element( interest2 );
    r_Interest_Tiling( interest_areas );
```

(continues on next page)

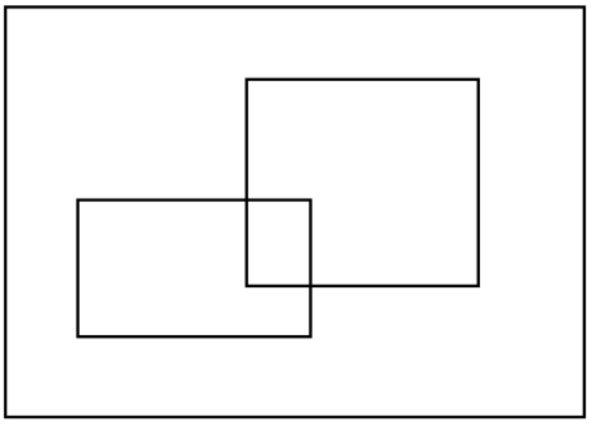


Figure 8.7: 2-D MDD object with two areas of interest

(continued from previous page)

```
    // ...
}
```

In addition to the list of areas of interest, two further parameters can be passed to the constructor, which are default arguments of the constructor :

```
r_Interest_Tiling( r_Dimension dim,
                   const std::vector<r_Minterval>& interest_areas,
                   r_Bytes ts = RMInit::clientTileSize,
                   Tilesize_Limit strat = SUB_TILING )
```

`ts` specifies the tile size to be used, whereas strat is the tile size limitation strategy. The *areas of interest* algorithm splits the multidimensional array into tiles aligned with the areas of interest so that future accesses to those areas result in no cells outside the area being loaded from disk. In order to perform this, the algorithm first calculates a maximum partition of the space using the *directional tiling* algorithm. Since this is suboptimal and the resulting tiles might have sizes greater than `clientTileSize` it then performs further merges or subtiling, depending on the tile size limitation strategy. The supported options for it are the following:

- `NO_LIMIT`: The blocks generated can have any size.
- `REGROUP`: Only when performing grouping/merging of tiles, the size of the resulting tile of two merges is checked against `clientTileSize`. If it is larger, they are not merged. Tiles larger than `clientTileSize` may exist (for instance, if the user specifies an interest area with a size larger than `clientTileSize`).
- `SUB_TILING`: In this strategy, regrouping is done regardless of the size of the generated tiles. After all the blocks are created, sub-tiling is performed on those whose size is larger than the tile size.
- `REGROUP_AND_SUBTILING`: This combines the last two strategies. When merging blocks, tiles larger than `clientTileSize` are never created and, when the final tiles are all created, sub-tiling is performed on those whose size is larger then `clientTileSize`.

Class `r_Stat_Tiling` and `r_Access`

These classes support *statistic tiling* and specification of access patterns, respectively. *Statistic tiling* splits MDD objects based on the access patterns passed to it as a parameter. It actually detects areas of interest out of a set of accesses and then performs tiling by using the *areas of interest* tiling algorithm. In order to determine the areas of interest, the algorithm performs a check of overlapping accesses to reduce accesses which correspond to the same area of interest to one single area of interest. In this step, the criteria used to reduce a set of accesses to a single area of interest is that if a group of accesses are near up to a given threshold, then they correspond to a single area of interest which is the minimum interval covering the accesses.

The *statistic tiling* algorithm then eliminates some of the areas of interest. It performs a check of the number of times each of the detected areas was accessed. Those which were accessed less than a given threshold are eliminated (they are accessed too few times to be considered areas of interest).

Five parameters are passed in the constructor of the `r_Stat_Tiling` class:

```
r_Stat_Tiling( r_Dimension dim,
               const std::vector<r_Access>& stat_info,
               r_Bytes ts = RMInit::clientTileSize,
               r_Area border_threshold = DEF_BORDER_THR,
               r_Double interesting_threshold = DEF_INTERESTING_THR␣
↪)
```

`border_th` is the border threshold for considering two access patterns to be the same, `interesting_th` is the interesting threshold, i.e., the percentage of accesses that must take place so that an area is considered being of interest when performing tiling and also `ts`, the tile size.

A call to `merge()` should be made prior to performing tiling, so that the statistic information about the accesses to the object can be updated and the tiling operation prepared.

```
r_Access merge(const std::vector<r_Access>& patterns)
```

This method inputs the statistic information into the class and calculates the new interest areas that will be used to perform tiling on the object. `r_Stat_Tiling` contains a list with the statistical information. This list is updated by the method. At the end, the list will contain the filtered and updated accesses count. This information can be used again as input to the method, or it can be stored for later usage.

The class r_Access represents an access pattern to a certain object. r_Stat_Tiling receives a list of these objects so that an appropriate tiling can be defined. The r_Access constructor

```
r_Access( const r_Minterval& region,
          unsigned long accesses = 1 )
```

takes as parameter the interval and the number of times the MDD subarray with domain region was accessed.

Class r_Convertor and Subclasses

The storage format indicator specifies the compression method used to compress / decompress tiles written to / retrieved from the database.

The transfer format indicator specifies the compression method used to compress / decompress tiles when transferred between client and server.

By default storage and transfer format is `r_Array` which means encoding in the server's main memory format, without any compression. The `r_Database` function `set_transfer_format()` allows to change transfer format and compression, for both directions uniformly:

```
void set_transfer_format( r_Data_Format format,
                          const char *formatParams=NULL )
```

The storage format in the server for MDD objects newly created by the client and its currently open transaction is set through `set_storage_format()`:

```
void set_storage_format( r_Data_Format format,
                         const char *formatParams=NULL)
```

Both functions understand these parameters, defined in the enumeration type enum `r_Data_Format` in module raslib, see Table 8.2.

Table 8.2: Storage and transfer formats and their parameters

Compression type	Constant	Description
“direct” storage	r_Array	no compression, row-major memory representation
Data exchange format	r_TIFF	TIFF format (2-D images, non-compressing)
	r_JPEG	JPEG format (2-D, lossy compression; *not recommended*!)
	r_HDF	HDF format (n-D, non-compressing)
	r_PNG	PNG format (2-D images, lossless compression)
	r_BMP	BMP format (2-D images, non-compressing)
	r_VFF	VFF format (3-D data, non-compressing)
	r_PPM	PPM format (2-D binary/gray/colour images, lossless)
	R_TOR	TOR format (used for 2-D geo laser scan images, non-compressing)
	R_DEM	ASCII format for 2-D digital elevation data (non-compressing)
Dedicated compressions (lossy if not indicated otherwise)	R_Auto	automatic compression (lossless)
	R_Zlib	ZLIB compression (lossless)
	R_RLE	RLE compression (lossless)
	R_Wavelet _Haar	Haar Wavelet compression
	r_Wavelet _Daubechies	Daubechies 4-tap Wavelet compression
	r_Sep_Zlib	ZLIB compression, compress base types separately (lossless)
	r_Sep_RLE	RLE compression, compress base types separately (lossless)
	r_Wavelet _Daub	Daubechies n-tap Wavelet compression, n=6, 8, . . . , 18, 20
	r_Wavelet _Least	Least asymmetric n-tap Wavelet comp., n=8, 10, . . . , 18, 20
	r_Wavelet _Coiflet	Coiflet n-tap Wavelet compression, n=6, 12, 18, 24, 30
	r_Wavelet _Qhaar	Lossy Haar Wavelet compression

Recommendations

- If space is not an issue, use `r_Array` storage for optimal performance.
- If compression is desired, use r_RLE for relatively homogeneous data, r_Zlib in general. R_Sep_Zlib and r_sep_RLE give an advantage in the compression rate whenever the cell type has a larger number (say, 3 and above) of cell components. All these compress

lossless, i.e. a compressed object inserted into the database will look the same after extraction.

- Use lossy compression only if you are sure that database users can live with information being filtered out of the original data.
- Almost all of the above formats have further parameters which allow fine tuning. They are passed in a string as comma-separated `"name=value"` pairs. See the `r_Convertor` class HTML documentation for the admissible names and values.
- Moreover, a white paper is available from rasdaman GmbH if you really want to go into the gory details.

Warning

From the "dedicated compression formats" listed above, only the RLE, SepRLE, Zlib, and SepZlib algorithms are fully released. The wavelet algorithms are provided as beta versions only, using them for non-experimental purposes is *not recommended* in the current version.

8.5.7 Error Classes

Figure 8.8 gives an overview on the rasdaman classes used to report on error situations:

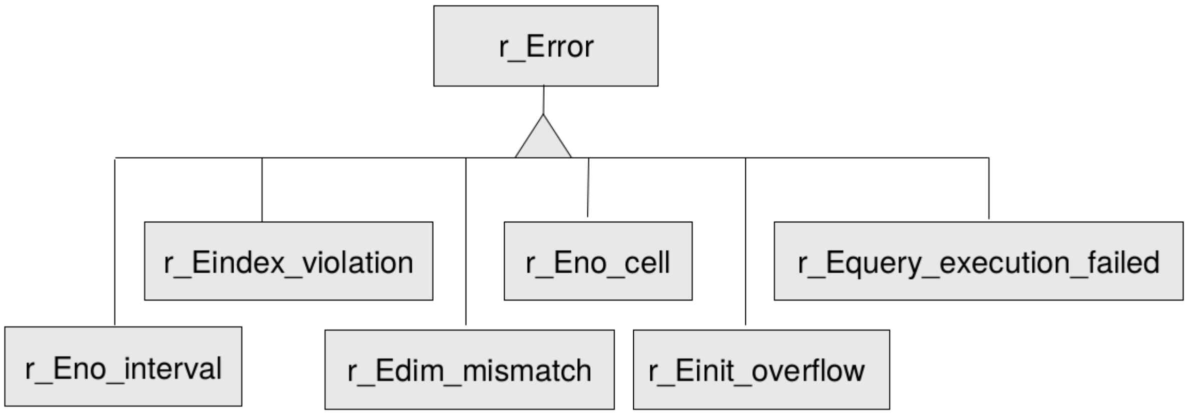


Figure 8.8: rasdaman Error Classes

Class `r_Error`

This class implements the relevant part of the ODMG C++ binding's `r_Error` class. It extends exception handling through deriving special classes for MDD specific errors. An error object consists of

- an error number which serves to uniquely identify the error,
- an error kind,
- an error text which verbally describes the error.

The error number, hence, serves as an index to a generic textual description of the error.

Error texts are loaded from the text file errtxts located in `$RMANHOME/bin` using the initialisation function `initTextTable()`. This mechanism allows the system administrator to translate error messages into target languages other than English.

If no error number is specified, the error kind will be used as error text.

The error description is received calling the member function `what()`.

Further information on error messages can be found in *Error Messages*.

Example

The following code fragment shows a typical try-catch block printing any potential error reported by rasdaman.

```
try
{
    // rasdaman access
}
catch( r_Error& errorObj )
{
    cerr << errorObj.what() << endl;
}
```

Class `r_Eno_interval`

This class represents an error object saying that the result is not an interval.

Class `r_Eindex_violation`

`r_Eindex_violation` represents an error object saying that the specified index is not within the bounds of the MDD object. In case the spatial domain of object a is [0:199] and the user asks for a[300] an error message of this class is raised.

Class `r_Edim_mismatch`

This class represents an error object saying that the dimensionalities of two objects do not match.

Class `r_Eno_cell`

`r_Eno_cell` represents an error object saying that the result is no cell. This happens f.e. if the cast operator for casting to the base type of class r_Marray is invoked on an object which is not 'zero-dimensional'.

Class `r_Einit_overflow`

This class represents an error object saying that an initialization overflow occured. This happens, e.g., if the stream operator is invoked more often than the object has dimensions.

Class `r_Equery_execution_failed`

The class is used for errors occuring through query execution. In most cases, the position which caused the error can be fixed. This position is specified by line number, column number, and the token which is involved. Additionally, the class is generic concerning the error type. Different error types can be specified by stating the error number.

Example

The following code segment shows possible error handling after query execution:

```
try
{
    // execute a rasdaman query
}
catch( r_Equery_execution_failed& errorObj )
{
    cerr << errorObj.what() << endl;
    cerr << "Line No " << errorObj.get_lineno();
    cerr << "Column No " << errorObj.get_columnno();
    cerr << "Token " << errorObj.get_token();
}
```

8.6 Linking MDD with Other Data

8.6.1 Sessions

Applications always maintain raster data and descriptive alphanumeric data. The latter often are called metadata - a term we adopt for the purpose of this discussion. Actually, all over the world a lot of effort already has been put into metadata modelling, and many database structures and metadata applications have been developed successfully. rasdaman does not reinvent the wheel: metadata remain unchanged in their (relational or object-oriented) database; they are not touched by rasdaman, but remain under the sole control of the underlying conventional DBMS (in the rasdaman documentation also referred to as "base DBMS").

Therefore, to work simultaneously with rasdaman and metadata, an application has to open both a rasdaman database and the database containing the metadata, and it must begin separate transactions in both databases.

Opening of database in rasdaman and the metadata DBMS are completely independent from each other, likewise are transactions in both systems. They can be nested or interleaved in any way.

In order to embed MDD objects and MDD collections in underlying databases, object identifiers and collection names may be used. These constitute references to rasdaman objects (which are stored in the base DBMS).

8.6.2 Collection Names

MDD collections in rasdaman must be named. This name can then be used by an application as a reference to the MDD collection. The most typical usage of these collection names is their storage in a base DBMS object or tuple in order to reference an MDD collection which is related to the object or tuple.

This is illustrated in the following example:

```
class Patient
{
    // ...
    private:
    d_String name;
    d_Date birthday;
    Address residence;
    SocialSecurityNumber ssn;
    //reference to rasdaman MDD collection:
    d_String XrayCollectionName;
    // ...
};
```

8.6.3 Object Identifiers

Each MDD object is uniquely identified in rasdaman by an object identifier. Object identifiers are implemented by the `r_OId` class. A globally unique object identifier has three components describing

- the system where it was created (system name),
- the database (base name) and
- the local object ID within the database.

The object identifier of a rasdaman object is returned by:

```
r_OId& r_Object::get_oid()
```

The object identifier may be used as a reference in an underlying database.

To be used as a reference in the underlying database the object identifier of a rasdaman object is stored as a member in an object of the underlying database. This is illustrated by the following example:

```
class SatelliteImage
{
    private:
    Date acquisitionDate;
    Location acquisitionLoc;
    // local reference to rasdaman MDD object:
    double imageRasOid;
    // ...
}
```

The member variable `imageRasOid` has to be translated into a rasdaman object identifier. This translation is done by the r_OId constructor:

```
r_OId::r_OId( const char* )
```

The string representation for a specific object identifier is returned by:

```
const char* r_OId::get_string_representation( )
```

Of course, alternatively the object identifier could be stored in its string representation.

8.7 Compilation and Linkage of Client Programs

8.7.1 Compilation

C++ applications using rasdaman have to include the header file `rasdaman.hh` which resides in `$RMANHOME/include`. Technically, `rasdaman.hh` includes further header files taken from the subdirectories of `$RMANHOME/include`.

The class library makes intensive use of templates. As templates are handled differently by the various compilers, individual measures have to be taken. To this end, the header files are instrumented to recognise the variable `OSTYPE` indicating the system platform. For example, setting OSTYPE to `linux-gnu` (case-sensitive!) indicates a Linux/Gnu environment, whereas the value `solaris` indicates a SUN/ Solaris platform. You should contact your dealer to find out which platforms are supported.

While in the deliverable sources (including the Makefiles provided) platform issues are dealt with, it nevertheless is important to understand the particularities. Therefore, some considerations follow next. If in doubt, you may want to contact the hotline.

Gnu

With the Gnu C++ compiler, the good way to handle templates is by early template instantiation using the compile flag `-DEARLY_TEMPLATE`. A template instantiation source file, template_inst.hh, is provided in the $RMANHOME/include/raslib directory; if the OSTYPE variable is set to linux-gnu, then this instantiation file will be included automatically.

Microsoft

With the Microsoft Visual C++ compiler, situation is similar as with Gnu above: it also needs early template instantiation.

Solaris

With the SUN-provided C++ compiler under Solaris, template instantiation at compile time is done by looking at the .cc files in the $RMANHOME/include subdirectories.

8.7.2 Linkage

For the linkage of an executable several libraries are needed. Those delivered with rasdaman are located in the $RMANHOME/lib directory.

One common problem are the dynamic libraries needed, such as libXmu.so. Usually there are different versions around. The version needed by a rasdaman application can be found out with the Unix ldd command which, for example, states:

```
libtiff.so.3 => /usr/lib/libtiff.so.3 (0x4001b000)
libstdc++-libc6.1-2.so.3 => not found
libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x4005e000)
libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x40071000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x400bc000)
libz.so.1 => /usr/lib/libz.so.1 (0x40160000)
libm.so.6 => /lib/libm.so.6 (0x4016f000)
libc.so.6 => /lib/libc.so.6 (0x4018c000)
libjpeg.so.62 => /usr/lib/libjpeg.so.62 (0x40281000)
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x402a0000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x402ab000)
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x402c2000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

As can be seen in the second line, libstdc++-libc6.1-2.so.3 cannot be found whereas all other references to dynamic libraries can be resolved. Sometimes a straightforward link to an older version helps, such as

```
ln -s libstdc++-libc6.1-2.so.2 $RMANHOME/lib/libstdc++-libc6.1-2.so.
↪3
```

Obviously very much care should be taken when fooling the system like this, and it is certainly not the recommended way.

Another common problem is to put the libraries into the right order in the link command, and which of them have to be linked twice to resolve all referenced symbols.

Some working examples can be found in the Makefiles of the delivered examples.

8.7.3 Client Environment Parameters

To allow for easier application steering, raslib evaluates the environment parameter `RMANCLIENTOPT` at program start-up. This variable can contain options similar to command line option syntax.

If contradicting options are set (e.g., `-tiling` and `-notiling`), then the last occurrence wins.

Options Known

Note: deprecated with the default rasnet server/client protocol.

-timeout	set server communication timeout seconds (default: 3600)
-notimeout	disable timeout, wait forever if necessary
-tilesize	set tile size bytes (default: 100000)
-notiling	disable client-side tiling
-l logfile	set log stream to *logfile* (default: ./client.log)

Example

The following shell dialog shows how an environment is set before invoking a rasdaman client. Settings done are: use timeout of 5 seconds, write log output to `/dev/null`.

```
$ export RMANCLIENTOPT="-timeout 5 -l /dev/null"
```

8.7.4 The Example Programs

An example program is delivered in `$RMANHOME/examples/c++`. This query program sends a rasql query to the rasdaman server and prints the result retrieved.

The code is documented and produces ample screen output, so it should be self explanatory. The programs are built by invoking `make` in the corresponding subdirectory.

Note: Before the test programs can be used, the rasdaman database has to be initialized.

8.7.5 Copyright Note

raslib contains code for password encoding based on MD5, located in the C++ library `$RMANHOME/lib/libcrypto.a`. This library must be linked to rasdaman applications in order to make them work.

Provision of this code is done in accordance with the GNU *Library General Public License* (see www.gnu.org).

8.7.6 Legal Note

Note that under some legislations usage and/or distribution of cryptography code may be prohibited by law. If you have obtained the abovementioned library in or from a region under such a legislation, whatever you do with it is fully under your own responsibility. Please inform rasdaman GmbH about the source where you have it obtained from so that we can take action against the violator.

8.8 HTML Documentation

All classes are described extensively in a set of HTML files shipped with the software. Starting point into the whole documentation is `$RMANHOME/doc/index.html`. Follow the "raslib" link to enter the description of the C++ interface.

The documentation can be viewed with any Web browser. Only graphical traversal between classes requires Java enabled; however, all links are available in textual form, too.

Top-level entry to the documentation shows the alphabetical listing of definitions, classes and functions; alternatively the class hierarchy display can be selected. Every class name is linked to the related class documentation. The subclass / superclass relations are indicated as indentation levels in the class list. Clicking a class name expands into the full class documentation consisting of three components.

First, there is the class inheritance hierarchy, including links to the direct subclasses and superclasses. The second part gives a short description of all class components, some of which have additional links to a more detailed documentation in the third part of the page. In this third part there is a detailed description of what the class does. Every time a class is used inside method declarations as either a parameter or return value, a link to the documentation of this class is provided.

CHAPTER

NINE

JAVA DEVELOPERS GUIDE

9.1 Preface

9.1.1 Overview

This guide provides information about how to use the rasdaman database management system. The booklet explains usage of rasj, the rasdaman Java API.

Follow the instructions in this guide as you develop your application which makes use of rasdaman services. Explanations detail how, from within a Java program, to create databases, collections, and instances; how to retrieve from databases; how to manipulate and delete instances within databases; how to influence physical storage parameters; how to do transaction handling and other administrative tasks.

9.1.2 Audience

The information in this manual is intended for application developers.

9.1.3 Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as *raswct*, release notes, and additional information on the rasdaman wiki.

The rasdaman Documentation Set consists of the following documents:

- Installation and Administration Guide
- Query Language Guide
- C++ Developer's Guide
- Java Developer's Guide
- raswct Developer's Guide

- the rasdaman wiki, accessible at www.rasdaman.org

9.2 Introduction

See the corresponding *Introduction* Section in the *rasdaman Query Language Guide*.

9.3 Terminology

See the corresponding *Terminology* Section in the *rasdaman Query Language Guide*.

9.4 Application Examples

9.4.1 Overview

This section contains an example of using the rasdaman Java API. The intention is, for the advanced programmer, to quickly get an overview on the programming style to be observed.

The source code can be found (slightly extended) in subdirectory `$RMANHOME/share/rasdaman/examples/java` of the rasdaman distribution directory.

For details on the operational semantics of the rasdaman data model the reader is strongly encouraged to study the *rasdaman Query Language Guide*.

9.4.2 Application Program Example Code

```
import rasj.*;
import rasj.odmg.*;
import org.odmg.*;
import java.util.*;

/** Example Java program for computing the avg cell value
 * for each n-D 8-bit grey image in a given collection.
 * set the server name with -server, the database name with -
↪database,
 * the collection name with -collection,
 * the port number with -port, the user login with -user,
 * the password with -passwd
 */
public class AvgCell {
    public static void main(String[] args) {
        String server = "localhost";
        String base = "RASBASE";
        String coll = "mr";
```

(continues on next page)

(continued from previous page)

```
        String port = "7001";
        String user = "rasguest";
        String passwd = "rasguest";

        double sum;

        for (int i = args.length - 1; i >= 0; i--) {
            //System.out.println(args[i]);
            if (args[i].equals("-server")) {
                server = args[i + 1];
            }
            if (args[i].equals("-database")) {
                base = args[i + 1];
            }
            if (args[i].equals("-collection")) {
                coll = args[i + 1];
            }
            if (args[i].equals("-port")) {
                port = args[i + 1];
            }
            if (args[i].equals("-user")) {
                user = args[i + 1];
            }
            if (args[i].equals("-passwd")) {
                passwd = args[i + 1];
            }
        }
        //System.out.println(server+base+coll+port+user+passwd);

        DBag resultBag = null;
        RasGMArray result = null;
        Transaction myTa = null;
        Database myDb = null;
        OQLQuery myQu = null;

        try {
            Implementation myApp = new RasImplementation(
                                            "http://" + server + ":" +␣
↪port);
            ((RasImplementation)myApp).setUserIdentification(user,␣
↪passwd);
            myDb = myApp.newDatabase();

            System.out.println("Opening database ...");
            myDb.open(base, Database.OPEN_READ_ONLY);

            System.out.println("Starting transaction ...");
            myTa = myApp.newTransaction();
            myTa.begin();
```

(continues on next page)

(continued from previous page)

```
            System.out.println("Retrieving MDDs ...");
            myQu = myApp.newOQLQuery();
            myQu.create("select img from " + coll + " as img");
            resultBag = (DBag)myQu.execute();
            if (resultBag != null) {
                Iterator iter = resultBag.iterator();
                while (iter.hasNext()) {
                    result = (RasGMArray)iter.next();
                    System.out.println(result);
                    if (result.getTypeLength() != 1) {
                        System.out.println("skipping image because"
                                + " of non-int cell type");
                    } else {
                        byte[] pixelfield = result.getArray();
                        sum = 0.0;
                        long size = result.getArraySize();
                        for (int i = 0; i < size; i++) {
                            sum += pixelfield[i];
                        }
                        System.out.println("Avarage over " + size +
↪" pixels is "
                                + ((sum / size) + 128));
                    }

                }
                System.out.println("All results");
            }

            System.out.println("Committing transaction ...");
            myTa.commit();

            System.out.println("Closing database ...");
            myDb.close();

        } catch (org.odmg.ODMGException e) {
            System.out.println("An exception has occurred: " + e.
↪getMessage());
            System.out.println("Try to abort the transaction ...");
            if (myTa != null) {
                myTa.abort();
            }

            try {
                System.out.println("Try to close the database ...");
                if (myDb != null) {
                    myDb.close();
                }
            } catch (org.odmg.ODMGException exp) {
```

(continues on next page)

(continued from previous page)

```
                System.err.println("Could not close the database: "
                        + exp.getMessage());
            }
        }
        System.out.println("Done.");
    }
}
```

Note: This sample program makes use of the `mr` collection provided with the rasdaman distribution package. See the rasdaman *Installation and Administration Guide* to learn on how to create this collection as part of the demonstration database.

9.5 rasj

9.5.1 Overview

The rasj package contains the API for Java-based access to the rasdaman database system. It relies on the ODMG standard which it implements to the extent that is necessary for raster data management.

The overall rasj package is subdivided into two packages, `rasj` and `org.odmg`. The `org.odmg` sub-package (see *ODMG*) implements the general ODMG specifications while the `rasj` sub-package implements rasdaman specific features.

9.5.2 Class Hierarchy

Note: All class hierarchies are generated from the rasj javadoc, which can be built in the `java/` directory with

```
mvn javadoc:javadoc
```

This generates a `javadoc` folder in the build directory (`build/java/target/site/apidocs/rasj`). Then, a commandline tool named `w3m` is used to dump the tree structure in HTML format of a package to text in console, example:

```
w3m -dump build/java/target/site/apidocs/rasj/package-tree.html
```

The `rasj` class hierarchy has the following structure.

```
* java.lang.Object
    * rasj.RasFastScale (implements rasj.global.RasGlobalDefs)
```

(continues on next page)

(continued from previous page)

```
    * rasj.RasImplementation (implements org.odmg.Implementation)
    * rasj.RasMInterval
    * rasj.odmg.RasObject (implements rasj.global.RasGlobalDefs)
        * rasj.RasGMArray (implements rasj.global.RasGlobalDefs)
            * rasj.RasMArrayByte
            * rasj.RasMArrayDouble
            * rasj.RasMArrayFloat
            * rasj.RasMArrayInteger
            * rasj.RasMArrayLong
            * rasj.RasMArrayShort
    * rasj.RasPoint
    * rasj.RasSInterval
    * rasj.RasStorageLayout
    * rasj.RasStructure
    * rasj.RasType
        * rasj.RasBaseType
            * rasj.RasPrimitiveType (implements rasj.global.
↪RasGlobalDefs)
            * rasj.RasStructureType
        * rasj.RasCollectionType
        * rasj.RasMArrayType
        * rasj.RasMIntervalType
        * rasj.RasOIDType
        * rasj.RasPointType
        * rasj.RasSIntervalType
    * java.lang.Throwable (implements java.io.Serializable)
        * java.lang.Exception
            * org.odmg.ODMGException
                * org.odmg.QueryException
                    * org.odmg.QueryInvalidException
                        * rasj.RasQueryExecutionFailedException
            * rasj.RasException
                * rasj.RasDimensionMismatchException
                * rasj.RasIndexOutOfBoundsException
                * rasj.RasResultIsNoCellException
                * rasj.RasResultIsNoIntervalException
                * rasj.RasStreamInputOverflowException
                * rasj.RasTypeInvalidException
            * java.lang.RuntimeException
                * org.odmg.ODMGRuntimeException
                    * rasj.RasConnectionFailedException
                * rasj.RasRuntimeException
                    * rasj.RasClientInternalException
                    * rasj.RasIllegalULongValueException
                    * rasj.RasIllegalUShortValueException
                    * rasj.RasInvalidNameException
                    * rasj.RasTypeNotSupportedException
                    * rasj.RasTypeUnknownException
```

9.5.3 Interface Hierarchy

The complete rasj interface hierarchy has the following structure.

```
* org.odmg.Implementation
    * rasj.RasImplementationInterface
```

9.6 ODMG

9.6.1 Overview

The ODMG classes implement classes defined in the ODMG standard providing functionality such as database open and close, transactions, querying, and unique identifiers, i.e., OIDs.

Don't Use `DArray`!

ODMG defines an interface `DArray` which also is part of the ODMG sub-package provided with the rasdaman distribution. These implement only 1-D arrays; most important, however, `DArray` is ***not compatible*** with rasdaman arrays. Therefore, ***do not use*** class `DArray` as a rasdaman array, but use class `RasGMArray` (and its subclasses) instead.

...But Do Use `Dbag`!

Queries return multi-sets as results. A *bag* or *multi-set* contains an arbitrary number of elements; like a set (and unlike a list), no particular sequence is defined, and like a list (and unlike a set), the same elements can occur multiply. The query result type, therefore, is `DBag`. See also *Storage Layout*.

9.6.2 Class Hierarchy

The complete `org.odmg` class hierarchy has the following structure.

```
* java.lang.Object
    * java.lang.Throwable (implements java.io.Serializable)
        * java.lang.Exception
            * org.odmg.ODMGException
                * org.odmg.DatabaseNotFoundException
                * org.odmg.DatabaseOpenException
                * org.odmg.ObjectNameNotFoundException
                * org.odmg.ObjectNameNotUniqueException
                * org.odmg.QueryException
                    * org.odmg.QueryInvalidException
                    * org.odmg.QueryParameterCountInvalidException
                    * org.odmg.QueryParameterTypeInvalidException
            * java.lang.RuntimeException
                * org.odmg.ODMGRuntimeException
                    * org.odmg.ClassNotPersistenceCapableException
```

(continues on next page)

(continued from previous page)

```
                    * org.odmg.DatabaseClosedException
                    * org.odmg.DatabaseIsReadOnlyException
                    * org.odmg.LockNotGrantedException
                    * org.odmg.NotImplementedException
                    * org.odmg.ObjectDeletedException
                    * org.odmg.ObjectNotPersistentException
                    * org.odmg.TransactionAbortedException
                    * org.odmg.TransactionInProgressException
                    * org.odmg.TransactionNotInProgressException
```

9.6.3 Interface Hierarchy

This is the `org.odmg` interface hierarchy:

```
* org.odmg.Database
* org.odmg.Implementation
* java.lang.Iterable<T>
    * java.util.Collection<E>
        * org.odmg.DCollection
            * org.odmg.DArray (also extends java.util.List<E>)
            * org.odmg.DBag
            * org.odmg.DList (also extends java.util.List<E>)
            * org.odmg.DSet (also extends java.util.Set<E>)
        * java.util.List<E>
            * org.odmg.DArray (also extends org.odmg.DCollection)
            * org.odmg.DList (also extends org.odmg.DCollection)
        * java.util.Set<E>
            * org.odmg.DSet (also extends org.odmg.DCollection)
* java.util.Map<K,V>
    * org.odmg.DMap
* org.odmg.OQLQuery
* org.odmg.Transaction
```

9.6.4 How To Use

The following code piece demonstrates a typical retrieval situation: a database is opened with username and password, a transaction is started, and then a query is executed against that database.

```
Transaction myTa = null;
Database myDb = null;
OQLQuery myQu = null;
DBag resultSet = null;
RasGMArray result = null;

Implementation myApp = new RasImplementation("http://" + server +␣
↪port );
```

(continues on next page)

(continued from previous page)

```
((RasImplementation)myApp).setUserIdentification(user, passwd);
myDb = myApp.newDatabase();
myDb.open( database, Database.OPEN_READ_ONLY );

myTa = myApp.newTransaction();
myTa.begin();

myQu = myApp.newOQLQuery();
myQu.create( "select mr from mr" );
resultSet = (DBag) myQu.execute();

// ...result set processing...

myTa.commit();
myDb.close();
```

Database Login

The database name and the address of a running server manager must be indicated. Further optional parameters and their defaults are:

- login (default: `"rasguest"`)
- password (default: `"rasguest"`)

Multiple ODMG Implementations

It is well possible to use several implementations - for example, from different vendors - of the ODMG classes simultaneously. Like rasj, other ODMG packages will provide an `Implementation` class in their `org.odmg` package. Instantiating one `Implementation` for each package is the only prerequisite to be done. The resulting code might look like the following (incomplete) example fragment where two different implementation classes are assumed, `RasImplementation` and `Implementation2`; note that transactions for different implementations are independent from each other.

```
Transaction myTa1 = null;
Database myDb1 = null;

Transaction myTa2 = null;
Database myDb2 = null;

Implementation rasApp1 = new RasImplementation( "http://" + server1␣
↪+ ":" + port1 );
((RasImplementation)rasApp1).setUserIdentification(user, passwd);
myDb1 = myApp1.newDatabase();
myDb1.open( rasbase, Database.OPEN_READ_ONLY );
MyTa1 = myApp1.newTransaction();
myTa1.begin();

Implementation2 myApp2 = new Implementation2( "http://" + server2 +
↪":" + port2 );
```

(continues on next page)

(continued from previous page)

```
((RasImplementation)myApp2).setUserIdentification(user, passwd);
myDb2 = myApp2.newDatabase();
myDb2.open( database2, Database.OPEN_READ_ONLY );
MyTa2 = myApp2.newTransaction();
myTa2.begin();

// ...now access both databases...

myTa1.commit();
myDb1.close();

myTa2.commit();
myDb2.close();
```

ODMG Functions Available

rasj does not implement ODMG fully (this would go beyond its purpose), rather it contains those functions necessary for rasdaman database access. When using the HTML hypertext documentation, clicking through the org.odmg package ultimately gets you to the rasdaman classes which implement the corresponding ODMG class. There, methods not available are marked as such.

Further Information

Details on how to process the query result can be found in *Storage Layout*. The example code makes use of the demonstration database whose set-up routines are part of the distribution package; find more on this topic in the rasdaman *Installation and Administration Guide*.

9.7 Points and Intervals

9.7.1 Overview

Point and interval handling is needed for indexing arrays, such as indication of array boundaries. To this end, classes `RasPoint`, `RasSInterval`, and `RasMInterval` for n-dimensional points, 1-D ("single-") intervals, and n-dimensional ("multi-") intervals resp. are provided.

Value Ranges and Consistency Constraints

All points, 1-D and n-D intervals can span negative values as well. Furthermore, intervals can have any integer value as lower bound. This is in contrast to most programming languages where usually the lower bound is fixed to 0.

However, intervals obviously need to match some consistency criteria to be valid. Foremostly, in a 1-D interval (class `RasSInterval`) as well as in an n-D interval (class `RasMInterval`) the lower bound must not be higher than the upper bound.

Further, operations between intervals of any type must yield a valid interval again. Consider the union of two 1-D intervals `s1` and `s2`,

```
s1.unionWith( s2 )
```

Intervals `s1` and `s2` must be overlap or at least be adjacent, otherwise the resulting interval would contain a hole (mathematically speaking, it would not be simply connected). As such situations are not allowed for intervals in rasdaman, corresponding exceptions will be thrown by rasj.

If nevertheless two intervals should be merged which are apart from each other, then operation `closureWith()` can be used. It will "fill" the gap between the intervals so that a valid result interval comes out.

The HTML manual lists each possible situation. It is recommended to study this for getting an understanding of all valid and invalid interval combinations.

9.7.2 Class Hierarchy

```
* java.lang.Object
    * rasj.RasPoint
    * rasj.RasSInterval
    * rasj.RasMInterval
```

Note: Class `java.lang.Object` obviously has further subclasses, not just the one shown here.

9.7.3 How To Use

Here are some sample code fragments showing usage of the point and interval classes:

RasPoint

```
// (1) point instantiation using string constructor:
RasPoint p1 = new RasPoint( "[ 3, 7 ]" );
// (2) point instantiation using numerical constructor:
RasPoint p2 = new RasPoint( 5, 0 );

// get point dimension:
int d = p2.dimension();

// test if points are equal:
boolean b = p1.equals( p2 );
```

RasSInterval

```
// create a 1-D intervals (100,200) and (-150,400), resp.:
RasSInterval s1 = new RasSInterval( 100, 200 );
RasSInterval s2 = new RasSInterval( "-150:400" );
```

(continues on next page)

(continued from previous page)

```
// no "[" and "]" !

// get upper bound of interval:
long hiBound = s2.high();
// get lower bound of interval:
long loBound = s2.low();

// test if interval intersects with another interval
// (the return value shows the kind of intersection)
int j = s1.intersectsWith( s2 );
```

RasMInterval

```
// create new 2-D interval, set bounds to (-1,1) and (3,7):
RasMInterval m1 = new RasMInterval( "[ -1:1, 3:7 ]" );
// create a 4-D interval, leaving open array bounds for now:
RasMInterval m2 = new RasMInterval( 4 );

// get number of cells:
long noOfCells = m1.cellCount();
```

9.8 Multidimensional Arrays

9.8.1 Overview

Instances of `RasGMArray` and its subclasses represent multidimensional arrays. To handle arrays with different base types and geometries, the “implements” relation of Java is used. With this approach, greyscale images, RGB images etc. can all be treated as subclasses of the general array class `RasGMArray`.

Currently supported are types for integer arrays (e.g., grayscale images) of various cell size, as well as types for floating-point arrays with single and double precision. All of them allow arrays of any dimension and extent per dimension.

Class Hierarchy

```
* rasj.odmg.RasObject (implements rasj.global.RasGlobalDefs)
    * rasj.RasGMArray (implements rasj.global.RasGlobalDefs)
        * rasj.RasMArrayByte
        * rasj.RasMArrayDouble
        * rasj.RasMArrayFloat
        * rasj.RasMArrayInteger
        * rasj.RasMArrayLong
        * rasj.RasMArrayShort
```

9.8.2 How To Use

A few code fragments will show appropriate usage of the array classes. To keep it brief and to the spot, we omit declarations and other standard steps; these can be looked up in the previous, complete coding examples.

Note: Current restriction

Queries can contain formal parameters, denoted by `$1`, `$2`, etc. (see *Query Language Guide* for details). In the current rasj implementation, only one MDD object can be bound per query (however, it is possible to bind several scalar values). This limitation will be overcome in future releases.

Example 1: compute summary data from array

The following code example retrieves all MDD objects from a sample collection and, fore each object, computes the average cell value. As a safeguard, averaging is carried out only in case of integer cells (i.e., greyscale pixels).

```
myQu = myApp.newOQLQuery();
myQu.create( "select mr from mr" );
DBag resultSet = (DBag) myQu.execute();
if (resultSet != null)
{
    Iterator iter = resultSet.iterator();
    while (iter.hasNext())
    {
        result = (RasGMArray) iter.next();
        if(result.getTypeLength() != 1)
            System.out.println("skipping image because of non-int␣
↪cell type" );
        else
        {
            byte[] pixelfield = result.getArray();
            double sum = 0.0;
            long size = result.getArraySize();
            for(int i=0; i<size; i++)
            sum += pixelfield[i];
            System.out.println( "Average over " + size +
            " pixels is " +
            ((sum/size)+128) );
        }
    }
}
```

Example 2: set up array object in main memory

The following code fragment instantiates a `RasGMArray` object as a 2-D greyscale image and fills it with values using the normal Java means:

```
// create 2-D MDD with cell length 1, i.e., type "byte":
RasGMArray myMDD = new RasGMArray(new RasMInterval( "[1:400,1:400]
↪"), 1 );
```

(continues on next page)

(continued from previous page)

```
// byte container for array data, matching in size:
byte[] mydata = new byte[160000];

// initialize array as all-black with two grey stripes:
for(int y=0; y<400; y++)
{
    for(int x=0; x<400; x++)
    {
        if((x>99 && x<151) || (x>299 && x<351))
            mydata[y*399+x]=100;
        else
            mydata[y*399+x]=0;
    }
}

// now insert byte array into MDD object
// (sets only the pointer, no copying takes place!):
myMDD.setArray(mydata);
```

As for the last line containing the import of array data into the MDD object, observe the following: There are specific get/set functions for the various supported array types, e.g., `getIntArray()`. While the `setArray()` and `getArray()` methods always will work, they will require data type conversion if the actual array cell type is not “byte”. Therefore, it is most efficient to always use that operation which respects the actual array data type.

The following code fragment instantiates a `RasGMArray` object as a 2-D greyscale image and fills it with values using the normal Java means:

Example 3: insert new array object into database

This example generates a new greyscale image collection named test in the database and inserts an image into this database collection.

Note that a new query object has to be generated for each query. It is not sufficient to just change the query string in the query object!

```
// set up query object for collection creation:
myQu.create( "create collection test GreySet" );
// set the object type name (used for server type checking):
myMDD.setObjectTypeName( "GreyImage" );
// finally, execute "create collection" statement:
myQu.execute();

// now create the insert statement:
myQu.create( "insert into test values $1" );
// let the server generate a new OID for the object to be
// inserted, and remember this OID locally:
myNewOID = myApp.getObjectId( myMDD );
// bind the MDD value which substitutes formal parameter $1:
myQu.bind( myMDD );
```

(continues on next page)

(continued from previous page)

```
// ...and ship the complete statement to the server:
myQu.execute();
```

9.8.3 rasdaman Cell Types

The set of cell base types known to rasdaman encompasses the usual numeric types. Below find the table of types known, and the necessary information to map them to Java types.

Null values, i.e., values of cells which have not been assigned a value yet, always are the numerical zero value of the corresponding type. This extends in the obvious way to composite cells.

Rasdaman	Length	Description
`octet`	8 bit	signed integer
`char`	8 bit	unsigned integer
`short`	16 bit	signed integer
`unsigned short`	16 bit	unsigned integer
`long`	32 bit	signed integer
`unsigned long`	32 bit	unsigned integer
`float`	32 bit	single precision floating point
`double`	64 bit	double precision floating point
`boolean`	1 bit[1]	true (nonzero value) false (zero value)

9.8.4 rasdaman Types vs. Java Types

Java types do not 1:1 correspond to rasdaman types. This is due to the fact that the Java type system in some aspects is different from what the ODMG Standard prescribes. Below find the most important caveats.

Long Integer

Long integer values in rasdaman always have 4 bytes, in accordance with the ODMG standard. The corresponding rasdaman types are `Ras_Long` and `Ras_ULong`.

In rasj, the array type to be used for 4-byte integers is `RasMArrayInteger` which matches with the Java `int` type occupying 4 bytes.

Mind that the Java type `long` represents 8 byte quantities. If an MDD object is passed to the database through rasj, a overflow test takes place on each integer value. An exception is thrown on overflow.

Unsigned Integers

Special care should be taken with unsigned integers, as Java does not support this. For example, for cells of type `Ras_UShort` (2 bytes) the array type `RasMArrayInteger` (4 bytes) must be used to collate values, according to the ODMG standard.

[1] memory usage is one byte per pixel

9.9 Storage Layout

9.9.1 Overview

At insertion time of an MDD object, several database-internal storage parameters can be set to affect the way the object is stored in the database. A `RasStorageLayout` object, attached to a `RasGMArray` MDD object, will guide storage of this MDD object when passed to the server through `RasOQLQuery.execute()`.

9.9.2 Class Hierarchy

```
* java.lang.Object
    * rasj.RasStorageLayout
```

9.9.3 How To Use

The following code fragment shows how to associate a storage layout object with an MDD object; the storage layout will be evaluated at insertion time of the MDD into the database.

```
// create 2-D MDD with cell length 1, i.e., type byte:
RasGMArray myMDD =
new RasGMArray(new RasMInterval( "[1:400,1:400]" ), 1 );

// assume that there is some byte array prepared, insert it:
myMDD.setArray( mydata );

// set image type name
myMDD.setObjectTypeName("GreyImage");

// add storage layout object:
RasStorageLayout myLayout = new RasStorageLayout();

// now you can set either TileSize or TileDomain; to this
// end, continue with Alternative 1 or 2, as described below
```

Alternative 1: set tile size

Having prepared the object as described above, now the tiling strategy can be set. Experience tells that a good size for tiles is 4 MB, but bear in mind that the optimal size for tiles depends on the actual user behaviour as well as various system parameters.

```
// define size of tiles as 128,000 bytes:
myLayout.setTileSize( 128000 );
myMDD.setStorageLayout( myLayout );
```

Alternative 2: set domain shape

As an alternative to setting the overall tile size, the domain can be prescribed. This is more exact, as it allows to define not only size, but also the extent per dimension. For example, if it is known from the user access patterns there are ten times as much vertical slices requested than are horizontal ones, then it may be a good strategy to define tiles with a vertical:horizontal ratio of 10 to 1.

```
// define tiles with spatial extent [1:1000,1:100]:
myLayout.setTileDomain("[1:1000,1:100]");
myMDD.setStorageLayout( myLayout );
```

Note: rasdaman also allows to set the storage and compression format, as well as client/server transfer format. However, currently the interface controlling these parameters is only available via the C++ interface, not yet via Java. In future versions format and compression control will be available via Java, too.

9.10 Collections and Queries

9.10.1 Overview

Bag versus Set

Queries return multi-sets as results. The corresponding query result type is `DBag`.

A *bag* or *multi-set* is a collection of elements similar to sets an lists; like a set (and unlike a list), no particular sequence is defined, and like a list (and unlike a set), the same elements can occur multiply. While `{1,2,3}` is an example for a set, `[1,2,2,3]` is a bag example; `[1,2,3]` denotes the same bag as `[3,2,1]`, because sequence is irrelevant in a bag.

Let us clarify the difference with an example. A query which returns the object identifiers (OIDs) of some database objects, such as

```
select oid(a)
from a
```

never will contain duplicates, as OIDs are unique by definition On the other hand, requesting summary information on MDD objects may well lead to duplicates; for example, in a query like this:

```
select avg_cells(a)
from a
```

several objects may share the same maximum or average cell value. In the latter case, it obviously is crucial to obtain duplicates also. Therefore, the query result always is `DBag`, which forms a particular subclass of the general class `DCollection`.

Nevertheless, we will use the term result set sometimes, as it is just common database speak.

Important Hint

Use `org.odmg.DBag`, do *not* use `rasj.odmg.RasBag`!

9.10.2 Class Hierarchy

```
* java.lang.Iterable<T>
    * java.util.Collection<E>
        * org.odmg.DCollection
            * org.odmg.DArray (also extends java.util.List<E>)
            * org.odmg.DBag
            * org.odmg.DList (also extends java.util.List<E>)
            * org.odmg.DSet (also extends java.util.Set<E>)
        * java.util.List<E>
            * org.odmg.DArray (also extends org.odmg.DCollection)
            * org.odmg.DList (also extends org.odmg.DCollection)
        * java.util.Set<E>
            * org.odmg.DSet (also extends org.odmg.DCollection)
```

9.10.3 How To Use

The following code piece demonstrates how to use object sets in the typical case of querying the database and piecewise processing the result set:

```
OQLQuery myQu = myApp.newOQLQuery();
myQu.create( "select mr from mr" );
DBag resultSet = (DBag) myQu.execute();
if (resultSet != null)
{
    Iterator iter = resultSet.iterator();
    while ( iter.hasNext() )
    {
        RasGMArray result = (RasGMArray) iter.next();
        // ...here now process result...
    }
}
```

Synchronous query execution

When a query is sent to the rasdaman server it will be executed in completeness - a running query cannot be aborted[2]. Care should be taken therefore not to start queries requiring resources beyond the capability of the server hardware and software environment, as the rasdaman service may be blocked for an indefinite time period.

[2] This has nothing to do with transactions - after each completion of a query, the embracing transaction can be aborted indeed.

9.10.4 Query Result Type

Database collections satisfy some criterion of homogeneity; this common property is expressed through the underlying type definition. Likewise, a collection returned as a query result has such an underlying common type definition. However, as queries dynamically describe and instantiate structures, this may not always adhere to some type existing in the database - sometimes the structure is new, so a type structure has to be generated "on the fly". While such a type does not have a name, its structure is well defined through the query itself.

This dynamic typing is predefined in the ODMG standard to which rasj adheres, so further information can be obtained there.

To access cells from arrays in query result bags, accessor functions are provided, such as `getObject()`, `getInteger()`. These functions are supervised by the type checking mechanism, hence using a function on an in appropriate type will cause an exception of type `ClassCastException`.

Generally speaking, it is up to the application to know the result type structure of the query it has sent to the server.

9.11 OIDs

9.11.1 Overview

The class `RasOID` manages object identifiers (OIDs) for persistent MDD and collections.

9.11.2 Class Hierarchy

```
* java.lang.Object
    * rasj.odmg.RasOID
```

Note: Class `java.lang.Object` obviously has further subclasses, not just the one shown here.

9.11.3 How To Use

The following code fragment prints the OID for each object in a query result set.

```
myQu = myApp.newOQLQuery();
myQu.create( "select mr from mr" );
DBag resultSet = (DBag) myQu.execute();
if (resultSet != null)
{
```

(continues on next page)

(continued from previous page)

```
    Iterator iter = resultSet.iterator();
    while ( iter.hasNext() )
    {
        RasGMArray result = (RasGMArray) iter.next();
        System.out.println( "<"
        + result.getOID().getSystemName() + "|"
        + result.getOID().getBaseName() + "|"
        + result.getOID().getLocalOID() + " >" );
        // last statement is equivalent to:
        // System.out.println( getObjectId( result ) );
    }
}
```

9.12 Type Management

9.12.1 Overview

rasdaman allows to define new types during runtime of the system. This is in contrast to programming languages where type structures are fixed at compilation time. rasdaman, therefore, offers separate mechanisms to maintain database types; these are provided through the `RasType` class and its subclasses. For each structure relevant in dealing with persistent (i.e., database stored) entities, a corresponding type class is provided.

Note: Right now, rasj does not allow to create and manipulate persistent types in the database; methods provided mainly serve to inquire the result type of a query for a maximum of code flexibility. Database type manipulation can be done through rasql queries, for more details see *Type Definition Using rasql*.

9.12.2 Class Hierarchy

```
* java.lang.Object
    * rasj.RasType
        * rasj.RasBaseType
            * rasj.RasPrimitiveType (implements rasj.global.
↪RasGlobalDefs)
            * rasj.RasStructureType
        * rasj.RasCollectionType
        * rasj.RasMArrayType
        * rasj.RasMIntervalType
        * rasj.RasOIDType
        * rasj.RasPointType
        * rasj.RasSIntervalType
```

9.12.3 How To Use

The following code piece demonstrates how the type structure given by some `RasType` object can be evaluated and printed in a user-friendly form.

```
// instantiate a sample MDD type object:
RasType rType = RasType.getAnyType( "marray <char, 1>" );

// Now let's forget again that we know rType, let's analyse.
// Check if the type object is some MDD type:
if (rType.getClass().getName().equals("rasj.RasMArrayType"))
{
    // yes, it is an MDD; is it structured or simple?
    if (rType.isStructType())
    {
        // yes, structured:
        System.out.println( "Structured base type is: " +
        rType.getBaseType() );
    }
    else
    {
        // no, atomic:
        System.out.println( "Atomic base type is: " +
        rType.getBaseType() );
    }
}
else
{
    // no, not an MDD at all.
    System.out.println(
    "type object doesn't describe an MArray." );
}
```

9.13 Exceptions

9.13.1 Overview

Exceptions serve to handle deviations from the desired flow of operation. Several exceptions can be thrown by rasj classes; as a general rule, all exceptions are subclassed from the general Java exception class `java.lang.Exception`. Exceptions are further grouped into four main classes

- `org.odmg.Exception`
- `java.lang.RuntimeException`
- `rasj.RasException`
- `rasj.RasRuntimeException`.

See the HTML documentation for details on the exception class hierarchy.

9.13.2 Class Hierarchy (pruned)

```
* java.lang.Object
    * java.lang.Throwable (implements java.io.Serializable)
        * java.lang.Exception
            * org.odmg.ODMGException
            * rasj.RasException
            * java.lang.RuntimeException
                * org.odmg.ODMGRuntimeException
                * rasj.RasRuntimeException
```

Note: All classes have further subclasses See *Class Hierarchy* and *Class Hierarchy* for more information.

9.13.3 Handling Exceptions in the Client

Catching an exception can be done, for example, as shown below. Obviously there are several ways doing this - however, a few rules should be obeyed:

- Granularity of exception catching depends on the overall program structure and purpose. For example, for data insertion one may want to build not just one large transaction, but several smaller units which, in case of failure, can be rerun with less time expenditure.
- Don't forget to clean up program state during exception recovery - think of closing (aborting? committing?) transactions, closing the database, etc.

Sample exception handling code

The following code piece demonstrates simple exception handling. The whole database access code is wrapped into a try statement. In case of an exception, the corresponding catch statement attempts to abort the transaction (if any is open) and to close the database. If in the course of these actions another exception occurs (for example, because the communication line has broken down), an error message is generated and the program terminates.

```
try
{
    Implementation myApp = new RasImplementation( "http://" +␣
↪server + port );
    ((RasImplementation)myApp).setUserIdentification(user, passwd);
    myDb = myApp.newDatabase();
    myDb.open(base, Database.OPEN_READ_ONLY);
    myTa = myApp.newTransaction();
    myTa.begin();
    // here do some work with the database
    myTa.commit();
```

(continues on next page)

(continued from previous page)

```
    myDb.close();
}
catch ( java.lang.Exception e ) // catch any error
{
    System.out.println( e.getMessage() );
    try
    {
        if(myTa != null)
            myTa.abort();
        if(myDb != null)
            myDb.close();
    }
    catch ( org.odmg.ODMGException exp ) // catch an abort
                                         // or close error
    {
        System.err.println( "Cannot commit/close: " + exp.
↪getMessage());
    }
}
```

9.13.4 Exceptions in the Class rasj.RasException

The following exceptions are rasj specific:

RasDimensionMismatchException

The dimensions of the two operand objects do not match.

RasIndexOutOfBoundsException

The specified index is not within the bounds of the array indexed.

RasResultIsNoCellException

The operation result is no cell, but an array cell is expected at this position. This happens, e.g., if the cast operator for casting to the base type of class `RasGMarray` is invoked on an object which is not 'zero-dimensional'.

RasResultIsNoIntervalException

The result is no interval, but an interval is expected at this position.

RasStreamInputOverflowException

An initialization overflow occured. This happens, e.g., if the stream input operator is invoked more often than the object has dimensions.

RasTypeInvalidException

Access method does not fit base type.

9.13.5 Exceptions in the Class `org.odmg.QueryInvalidException`

RasQueryExecutionFailedException

This exception extends `ODMGQueryInvalidException` by offering direct access to the rasdaman error number and the line, column and token in the query string that produced the error.

9.13.6 Exceptions in the Class `org.odmg.ODMGRuntimeException`

RasConnectionFailedException

This exception is raised when the connection to the server fails.

9.13.7 Exceptions in the Class `rasj.RasRuntimeException`

RasClientInternalException

This runtime exception indicates an internal error on client side which report to your dealer containing the complete error message and a precise description of the actions that lead to this exception.

RasTypeNotSupportedException

This exception is raised when the base type of a query result is not supported by the current version of the rasj package.

RasTypeUnknownException

This exception is raised when the base type of a query result is unknown on client-side.

RasInvalidNameException

This exception is thrown if an object name contains invalid characters.

RasIllegalULongValueException

Thrown if a RasMArrayLong is trying to be sent to the server where one or more cell values are out of the range of 32-bit unsigned integers.

RasIllegalUShortValueException

Thrown if a RasMArrayShort is trying to be sent to the server where one or more cell values are out of the range of 16-bit unsigned integers.

9.14 Compilation and Execution of Client Programs

9.14.1 Compiling Code Using rasj

Environment Variables

The `CLASSPATH` variable - which is used by the Java compiler to locate packages used - must be extended with the path for the rasj directory of the rasdaman distribution. This can be done, e.g., with the following command:

```
export CLASSPATH=$RMANHOME/lib/rasj.jar;$CLASSPATH
```

Alternatively, the `-classpath` or `-cp` option of `javac` can be used to explicitly make known the package locations to the Java compiler.

Further, the JDK class directory must be contained in `CLASSPATH`, and the JDK binaries directory must be contained in the `PATH` variable.

Java sources making use of the rasj package are compiled and run as usual. For example, a source file `Lookup.java` containing class `Lookup` would be compiled as

```
javac Lookup.java
```

Running it as an application would be done through this command line statement:

```
java Lookup
```

Sample Programs

Several sample Java programs are provided as part of the rasdaman distribution; they are located in the `$RMANHOME/share/rasdaman/examples/java` directory of the distribution.

Web Servlets and Applications

rasj allows to build applications written in Java which can be web servlets as well as applications. See petascope for example *Geo Services Guide*.

Notes

Remember the uppercase/lowercase distinction of Java!

For all classes with package definitions - such as `rasj.RasGMArray` - the package name must be prefixed.

9.14.2 Java Version Compatibility Statement

rasj has been successfully tested with JDK versions 1.7+.

9.14.3 HTTP communication

rasj internally uses HTTP to communicate with the rasdaman server. By selecting individual URLs and ports in the database open statement (see *ODMG*), safe database access across firewalls is possible.

9.14.4 Copyright Note

rasj contains code for password encoding based on MD5.

Provision of this code is done in accordance with the GNU *Library General Public License* (see www.gnu.org).

9.14.5 Legal Note

Note that under some legislations usage and/or distribution of cryptography code may be prohibited by law. If you have obtained the abovementioned library in or from a region under such a legislation, whatever you do with it is fully under your own responsibility. Inform rasdaman GmbH about the source where you have it obtained from so that we can take action against any violator.

9.15 HTML Documentation

The implementation is described in extensive documentation integrated with the source code from which a set of HTML files. This documentation can be used with any Web browser. The entry point for the complete documentation pages, including the rasj part, is `doc/index.html` in the rasdaman distribution directory (see *Server Architecture*).

ODMG Class Availability

Note that the `org.odmg` package is taken verbatim from the ODMG standard. rasdaman interface classes are derived as implementations of the standard classes. However, only those classes have been implemented which are necessary for rasdaman. If in doubt, the `Implementation` section should be consulted where unavailable items are marked (due to copyright restrictions, the ODMG text must remain unchanged).

CHAPTER

TEN

RASWCT WEB CLIENT TOOLKIT DEVELOPER GUIDE

10.1 Preface

10.1.1 Overview

Purpose of the raswct ("rasdaman Web Client Toolkit") toolkit is to allow developers creating Web user interfaces for displaying data from a raster database.

10.1.2 Implementation

The toolkit is developed in Javascript and uses popular libraries like jQuery. Its structure follows the principle of separating data transmission and processing from the presentation, the two main namespaces reflecting this philosophy:

- **Query namespace** - containing all the classes that can be used to retrieve data from a server, be it a simple HTTP server or a rasdaman server.
- **Widget namespace** - containing all the classes that can be used to display the data in meaningful ways

This document describes how to create widgets and modify them to suit particular purposes. At the end of each widget description an example of use is given. More examples can be found in the `applications/raswct/demo` folder.

10.1.3 Audience

Information in this manual is intended primarily for Web application developers.

10.1.4 Rasdaman Documentation Set

This manual should be read in conjunction with the complete rasdaman documentation set which this guide is part of. The documentation set in its completeness covers all important information needed to work with the rasdaman system, such as programming and query access to databases, guidance to utilities such as *raswct*, release notes, and additional information on the rasdaman wiki.

The rasdaman Documentation Set consists of the following documents:

- Installation and Administration Guide
- Query Language Guide
- C++ Developer's Guide
- Java Developer's Guide
- raswct Developer's Guide
- the rasdaman wiki, accessible at www.rasdaman.org

10.2 Introduction

10.2.1 Purpose and Use

This toolkit allows developers to easily create individualized Web interfaces for displaying multi-dimensional raster data. For example, diagrams serve to present 1-D query results, images and a geo Web Map interface serve to display 2-D query results. 3-D displays are under development. All such data can stem from multi-dimensional database contents, such as 1-D extracts from a 4-D climate data set.

Database queries can be hidden behind interactive parameter setting through sliders, gauges, etc., thereby hiding the complexity of the query language to casual users.

Crafting such Web interfaces often is as easy as writing HTML, without resorting to JavaScript, which is the raswct implementation language. That said, all JavaScript is available to advanced developers for designing high-end interactive data interfaces.

10.2.2 Implementation

The raswct toolkit is developed in Javascript and uses popular libraries, like jQuery. Its structure follows the principle of separating data transmission and processing from the presentation:

- The **Query namespace**, Rj.query, contains all the classes for data retrieval from a server, be it a simple HTTP server or a rasdaman server.
- The **Widget namespace**, Rj.widget, contains all the classes for displaying data in various ways.

- The **Utility namespace**, Rj.util, contains various utility functions that help creating interaction widgets.

10.2.3 References

The raswct toolkit is heavily used in the EarthLook geo service standards showcase.

10.3 Utility Namespace

10.3.1 Bindable

Rj.util.Bindable is a trait and it is used as an interface for binding objects.

10.3.2 Binder Manager

Rj.util._BinderManager class manages the binders between Rj objects.

10.3.3 Cache Engine

Rj.util._CacheEngine singleton class acts as a key value store for caching misc data directly in the browser.

10.3.4 Config Manager

Rj.util._ConfigManager class acts as a singleton to store the configuration data used across raswct modules.

10.3.5 Constants

Rj.util.Constants class contains all the constants needed across the toolkit.

10.3.6 CSV Parser

Rj.util.CSVParser class is designed to help with the parsing of CSV data produced by rasdaman server or petascope wcps services to native javascript objects.

10.3.7 Data Series

Rj.util.DataSeries class is an discrete indexed array object, similar to how a 2D diagram is represented e.g. [[1, 2], [3, 6]] where the domain is {1,3} and the codomain is {2,6}.

10.3.8 Error Manager

Rj.util._ErrorManager singleton class manages the error messages, displaying them to the user or just reporting them in the dev console.

10.3.9 Global State

Rj.util.GlobalState class provides a common area for defining shared static information across modules.

10.3.10 Map layer

Rj.util.MapLayer class defines a layer used as an abstraction for map layers that can be added to any Rj.widget.Map

10.3.11 Multidimensional Array

Rj.util.MultiDimArray class is a representation of a multidimensional array that has easy to use accessor methods.

10.3.12 Notification Manager

Rj.util._NotificationManager class defines a notification diaglogue to show success/failure to users.

10.3.13 Observable

Rj.util.Observable is a trait and it is used as an interface for event communication.

10.3.14 Util

This file extends the functionality of the underscore library to utilities that are needed across the project.

10.3.15 XMLDoc

Rj.util.XMLDoc class provides a series of utility functions for easier parsing of XML docs using XPath.

10.4 Query Namespace

10.4.1 Executable

Description

Rj.query.Executable is a trait and it should be used in the composition of any query classes that can be evaluated by a services.

Atrributes

Name	Type	Description
- cached	Object	Configuration for the cached property, default it contains { value: false }

Methods

Name	Parameters	Return Type	Description
+ evaluate()	callback: Function, persistent: Bool		Evaluates the query and calls a corresponding callback function
- fireCallbacks()	response: Object, httpStatus: Int		Iterate array of callback functions and trigger these functions by input response and httpStatus values
- evaluateCached()	transport: Rj.query.Transport, callback: Function		Check if Rj.util.CacheEngine contains a response before applying a callback
- evaluateRaw()	transport: Rj.query.Transport, callback: Function		Send request from transport to server to get response before applying a call back

10.4.2 Transport

Description

Rj.query.Transport is a trait and it is used by the Executable trait to send the queries to the server to be evaluated.

Atrributes

Name	Type	Description
- serviceUrl	String	Petascope endpoint to send request
- serviceHttpMethod	String	HTTP method to send request (default: POST)
- params	Object	An object to contain param (keys, values) for the request
- parseResponse	Object	Parse the response from the request
- binary	Bool	If reponse is not in text format (default: true)

Methods

Name	Parameters	Return Type	Description
+ toHashCode()			Create a String request from all properties and calculate a hashcode from it

10.4.3 LiteralQuery

Description

Rj.query.LiteralQuery is class to create a literal query trait provides functionality for defining string queries containing parameters that can be changed. This is an interface class for subclasses to implements. A literal query example: e.g "SELECT @col FROM @col WHERE @cond"

Atrributes

Name	Type	Description
- query	String	The litteral query, e.g. "SELECT @col FROM @col"
- variables	Object	An object of form {variable: value}, {"@col" : "mr"}

Methods

Name	Parameters	Return Type	Description
+ setVariable()	varName: String, value: String		Set a variable to a certain value.
+ getVariable()	varName: String	String	Get a value for a a certain variable.
+ toString()		String	Returns the string representation after the query is expanded by replacing the vars.
+ isReady()		Bool	Indicates if all the variables in the query are set.
- expand()		String	Iterate the variables array and replace the place holders with values from the array

10.4.4 WCPS Query

Description

Rj.query.WCPSQuery is class to create a WCPSQuery object which can send WCPS queries to a service that can process them and parse the result to obtain meaningul data for Widgets.

Atrributes

Name	Type	Description
- binaryFormat	bool	Should be set to true if the query returns a binary format (e.g. image) instead of text format.
- WCPSService	String	Petascope endpoint for WCPS query.

Methods

Name	Parameters	Return Type	Description
+ transport()			Returns a Rj.query.Transport object that can be used internally by Executable trait

10.4.5 Rasql Query

Description

Rj.query.RasQuery is class to create a RasQuery object which is a Rasql query. All queries defined in rasql can be used with this class.

Methods

Name	Parameters	Return Type	Description
+ transport()			Returns a Rj.query.Transport object that can be used internally by Executable trait
- parseResponse()			Parse the text result from rasdaman as JSON object

10.5 Widget Namespace

10.5.1 Base Widget

Description

Rj.widget._BaseWidget is base class for widgets, exposing methods for easy communication between the current widget and other widgets on the page. All widgets also contain a descendant of Rj.query.LiteralQuery which it can use to receive information from ther server.

Atrributes

Name	Type	Description
- selector	CSS3 / XPath	A CSS3/XPath selector used as indentifier for the position of the widget.

Methods

Name	Parameters	Return Type	Description
+ show()			Make the widget visible. By default widgets are rendered invisible
+ hide()			Make the widget invisible
+ destroy()			Destroy the widget
- render()			laceholder function that should be extended by any showing widget
- clear()			Registers a new handler for a specific event
- refresh()			Removes the widget from the container and re-renders it

Warning: This is a base class for graphs so it shouldn't be initialized or used except for extending the current graph system.

10.5.2 Input Widget

Description

Rj.widget._InputWidget is a simple grouper class that helps better define the relationships between widgets.

Attributes

Name	Type	Description
- value	string	The value displayed in widget.

Warning: This is a base class for graphs so it shouldn't be initialized or used except for extending the current graph system.

10.5.3 Knob Widget

Description

Rj.Widget.Knob class defines a knob widget.

Attributes

Name	Type	Description
- min	Int	The lower bound of the knob.
- max	Int	The higher bound of the knob.
- value	Int	The initial value of the knob.
- reverse	Bool	If true, the values are distributed backwards (from 360 degrees to 0 degrees).
- snap	Int	The number of degrees from which the knob is snapped to 0.

Methods

Name	Parameters	Return Type	Description
- setValue()	value: Int		Set value for knob's value.
- render()			Render this knob widget

Examples

The following code creates a Knob object within a `<div id="knob"></div>` element:

```
var knob = new Rj.Widget.Knob(0, 10, 5, false, 20);
knob.renderTo("knob");
```

10.5.4 Slider Widget

Description

Rj.widget._Slider class defines a slider widget. This class is private and should be instatiated on its own. See Rj.widget.HorizontalSlider and Rj.widget.VerticalSlider if you need to create a slider.

Attributes

Name	Type	Description
- min	Int	The min value the slider can take
- max	Int	The max value the slider can take
- orientation	String	The orientation of the slider, either vertical or horizontal
- step	Float	The step size to which the slider should be increased on slide action
- tooltip	Bool	True if the slider should have a tooltip, false otherwise
- label	String	The label shown in the tooltip
- height	Int	The height of the slider
- width	Int	The width of the slider
- instantChange	Int	If true the slider will change the value to the slide movement of the incrementor, otherwise only to the mouse up movement

Methods

Name	Parameters	Return Type	Description
- createTooltip()			Creates a tooltip attached to the slider
- prepareRendering()			Prepares the rendering process for the dojoRenderer
- finishRendering()			Finishing touches to the slider
- renderDojoSlider()			Renders the slider using the dojo library widget
- render()			Renders the slider with all its components
- getDojoClass()			Return Rj.widget._Slider.DojoSliderClasses
- clear()			Destroy the slider widget recursively
- refresh()			Refresh the widget by using softRefresh() with timeout
- softRefresh()			Refresh the widget by using clear() and render()

10.5.5 Horizontal Slider Widget

Description

Rj.widget.HorizontalSlider class defines a horizontal slider that can be used to slide through an interval of numerical values. It extends Rj.widget._Slider class.

10.5.6 Vertical Slider Widget

Description

Rj.widget.VerticalSlider class defines a vertical slider that can be used to slide through an interval of numerical values. It extends Rj.widget._Slider class.

10.5.7 Output Widget

Description

Rj.widget._OutputWidget is a simple grouper class that helps better define the relationships between widgets.

Warning: This is a base class for graphs so it shouldn't be initialized or used except for extending the current graph system.

10.5.8 Binary Image Widget

Description

Rj.widget.BinaryImage class defines a binary image widget that can consume uint8 data and transform it into a image that can be displayed in the browser.

Attributes

Name	Type	Description
- width	Int	The width of the image
- height	Int	The height of the image
- binaryData	String	Image in base64 String

Methods

Name	Parameters	Return Type	Description
- generateBase64Data()			Convert an array of buffer data to base64 String
- render()			Render the base64 String to an image and display

10.5.9 Map Widget

Description

Rj.widget.Map class defines a widget used for displaying maps composed of several layers.

Attributes

Name	Type	Description
- map	OpenLayers.Map	OpenLayers map object
- width	Int	The width of the map
- height	Int	The height of the map

Methods

Name	Parameters	Return Type	Description
- clear()			Destroy the OpenLayers map object and clear the div container of it
- render()			Render the base64 String to an image and display

10.5.10 Gauge Widget

Description

Rj.Widget.Gauge class defines a circular gauge widget.

Attributes

Name	Type	Description
- value	Int	The initial value displayed.
- min	Int	The lower bound of the displayed values.
- max	Int	The upper bound of the displayed value.
- title	String	The title of the widget.
- label	String	The label of the widget.
- widthScale	Float	The scale at which the widget is displayed. 1 is the reference point.
- showMinMax	Bool	Shows or hides the bounding values.

Methods

Name	Parameters	Return Type	Description
- render()			Display the Gauge widtget

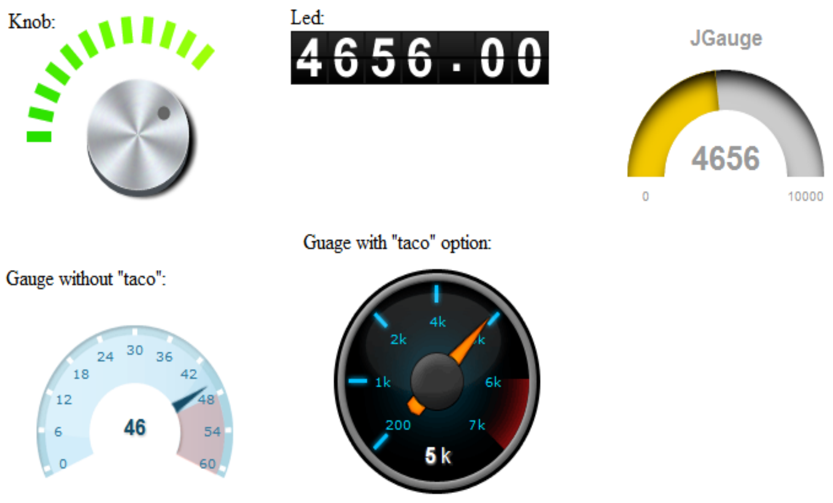
Examples

The following example will display a gauge within a `<div id ="gauge"></div>` element.

```
var gauge = new Rj.Widget.Gauge(null, 24);
gauge.renderTo("gauge");
```

Gauge overview

Below the gauges are listed which are available currently; they are described in the subsequence subsections.



10.5.11 Led Widget

Description

Rj.Widget.Led class defines a led counter widget.

Attributes

Name	Type	Description
- value	Float	The initial value displayed.
- intDigits	int	The number of digits of the display.
- fracDigits	Bool	The number of fractional digits to display.

Methods

Name	Parameters	Return Type	Description
- render()			Display the Led widtget

Examples

The following example will display a LED within a <div id = “led”></div> element.

```
var led = new Rj.Widget.Led(100.54, 3, 2);
led.renderTo("led");
```

10.5.12 SpeedoMeter Widget

Description

Rj.widget.SpeedoMeter class defines a speedo meter widget.

Attributes

Name	Type	Description
- value	Float	The initial value displayed
- labelSuffix	String	The suffix of the label

Methods

Name	Parameters	Return Type	Description
- render()			Display the SpeedoMeter widtget

10.5.13 Tooltip Widget

Description

Rj.widget.ToolTip class defines a a tooltip widget.

Attributes

Name	Type	Description
- value	String	Text to be shown
- pretext	String	Pre text to be shown
- postext	String	Post text to be shown
- adjust	Object	Some keys, values to adjust tooltip
- place	String	The place to shown tooltip (default: bottom)
- mouse	Bool	Mouse event on tooltip (default: false)
- delay	Int	Time to show tooltip (default: 1000)

Methods

Name	Parameters	Return Type	Description
- render()			Display the Tooltip widtget

10.5.14 Diagram Widget

Description

Rj.widget._Diagram class defines a widget used as a base for all diagrams.

Attributes

Name	Type	Description
- title	String	The title of this diagram.
- xLabel	String	The title of the X axis.
- yLabel	String	The title of the Y axis.
- tooltip	String	Indicates whether a Tooltip with tips about how the diagram works should be shown.
- dataSeries	Rj.util.DataSeries	The series to be plotted.
- width	Int	The width of the diagram.
- height	Int	The height of the diagram.

Methods

Name	Parameters	Return Type	Description
+ getData		Array	Returns the data series in the format that is sent to the plot.
+ addDataSeries	series: Rj.util.DataSeries		Adds a data series to the diagram as an array of form [[x,y] , [x1, y1]].
+ removeDataSeries	seriesName: String		Removes a series from the diagram.
- configure	cfg: Object	Object	Configures the chart object before rendering. All subclasses should override this method in order to add their specific configurations.
- getSeriesColors		Array	Returns an array of series' colors.
- getSeriesNames		Array	Returns an array of series' names.
- render			Renders the widget by its id DOM element.
- bindSeries	series: Rj.util.DataSeries		Add event listeners for series and then call self._refresh() method.

Warning: This is a base class for graphs so it shouldn't be initialized or used except for extending the current graph system.

10.5.15 Area Diagram

Description

Rj.Widget.AreaDiagram class defines a widget used for displaying area graphs.

Methods

Name	Parameters	Return Type	Description
- configure	cfg: Object	Object	Configures the chart object before rendering .

Examples

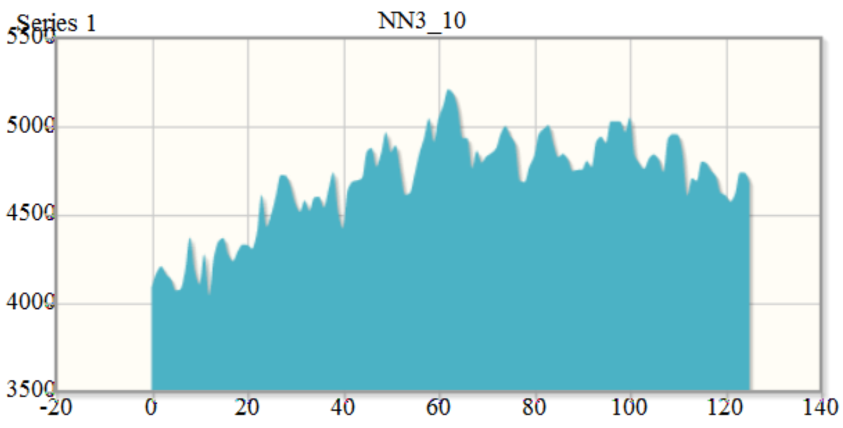
JS Code:

```
var source = "NN3\_10"
//Initialize query
var query = new Rj.query.WCPSQuery('for t1 in (mean_summer_airtemp)␣
↪return encode (t1[ x(126), y(-10) ], "csv")');
//Create widget
var diagram = new Rj.Widget.AreaDiagram( query, "#chartPlace",␣
↪source);
// Get diagram axis and labels before data is rendered
diagram.addListener( 'wcps','datapreload',
    function(response){
        var values = [];
        for(var i = 0; i < response.data.length; i++){
            values.push( [i, parseInt(response.data[i], 10)]);
        }
        //Configure the widget labels
        this.configure({
            axes: {
                xaxis: { title: response.domainInfo.axisLabel },
                yaxis: { title : "Values" }
            }
        });
        return { data : values };
    }
);
//Load the data and render the widget
diagram.loadData(true);
```

HTML Code:

```
<div id='chartPlace' style='width:600px; height:500px;'>
<!-- The chart will go here -->
</div>
```

Visual appearance:



10.5.16 Bar Diagram

Description

Rj.Widget.BarDiagram class defines a widget used for displaying bar graphs.

Methods

Name	Parameters	Return Type	Description
- configure	cfg: Object	Object	Configures the chart object before rendering .

10.5.17 Linear Diagram

Description

Rj.Widget.LinearDiagram class defines a widget used for displaying linear graphs.

Methods

Name	Parameters	Return Type	Description
- configure	cfg: Object	Object	Configures the chart object before rendering..

Examples

JS Code:

```
//Initialize query
var query = new Rj.query.WCPSQuery('for t1 in (mean_summer_airtemp)␣
↪return encode (t1[ x(126), y(-10) ], "csv")');
//Create widget
var diagram = new Rj.Widget.LinearDiagram( query, "#chartPlace",␣
↪source );
// Get diagram axis and labels before data is rendered
// by listening to the datapreload event
diagram.addListener( 'wcps','datapreload',
```

(continues on next page)

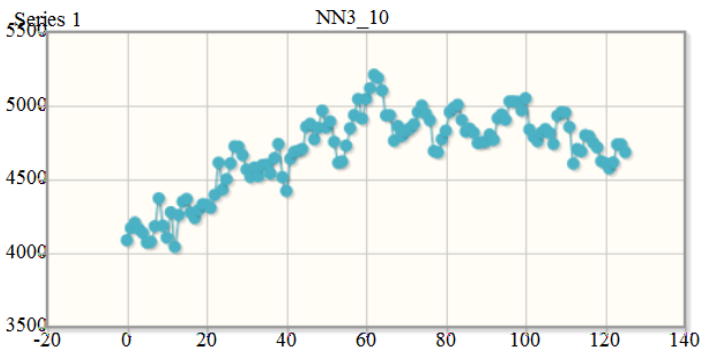
(continued from previous page)

```
    function(response){
        // Check if any errors occurred,
        // and if so display a nice error message
        if(response.error){
            $("body").append( "<div id='dialog'>" + response.error␣
↪+ '</div>');
            $( "#dialog" ).dialog({
                modal : true,
                title : 'Parse Error'
            }).show();
            throw "Error while processing the data";
        }
        var values = [];
        for(var i = 0; i < response.data.length; i++){
            values.push( [i, parseInt(response.data[i], 10)]);
        }
        //Configure the widget axes
        this.configure({
            axes : {
                xaxis:{ title : response.domainInfo.axisLabel },
                yaxis: { title : "Values" }
            }
        });
        return { data : [values] };
    }
);
// load data and render widget
diagram.loadData(true);
```

HTML Code:

```
<div id='chartPlace' style='width:600px; height:500px;'>
<!-- The chart will go here -->
</div>
```

Visual appearance:



10.5.18 Scatter Diagram

Description

Rj.widget.ScatterDiagram class defines a widget used for displaying scattered graphs.

Methods

Name	Parameters	Return Type	Description
- configure	cfg: Object	Object	Configures the chart object before rendering.

Examples

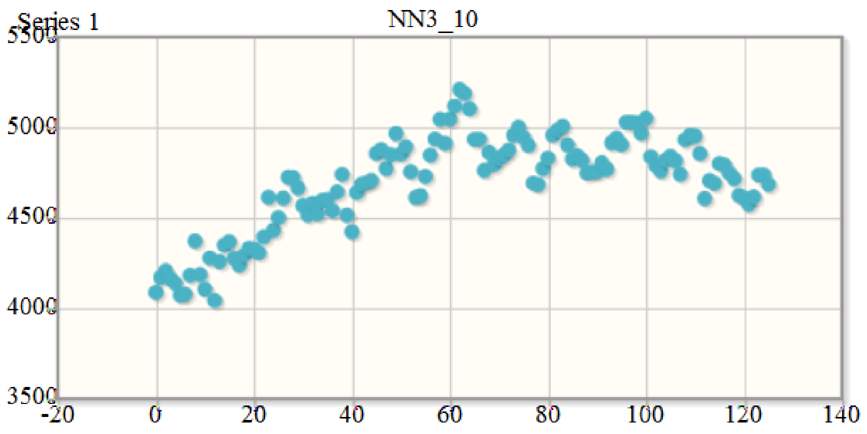
JS Code:

```
// Initialize query
var query = new Rj.query.WCPSQuery('for t1 in (mean_summer_airtemp)␣
↪return encode (t1[ x(126), y(-10) ], "csv")');
// Create widget
var diagram = new Rj.Widget.ScatterDiagram(query, "#chartPlace",␣
↪source);
// Get diagram axis and labels after data is loaded
// by listening to datapreload event
diagram.addListener( 'wcps','datapreload',
    function(response){
        var values = [];
        for(var i = 0; i < response.data.length; i++){
            values.push( [i, parseInt(response.data[i], 10)]);
        }
        // Configure widget labels
        this.configure({
            axes : {
                xaxis: { title : response.domainInfo.axisLabel },
                yaxis : { title : "Values" }
            }
        });
        return { data : values };
    }
);
diagram.loadData(true);
```

HTML Code:

```
<div id='chartPlace' style='width:600px; height:500px;'>
<!-- The chart will go here -->
</div>'
```

Visual appearance:


Series 1
NN3_10
5500
5000
4500
4000
3500
-20
0
20
40
60
80
100
120
140

CHAPTER

ELEVEN

CHEATSHEETS

11.1 WCS

The OGC Web Coverage Service (WCS) standard defines support for modeling and retrieval of geospatial data as *coverages* (e.g. sensor, image, or statistics data).

WCS consists of a *Core* specification for basic operation support with regards to coverage discovery and retreival, and various *Extension* specifications for optional capabilities that a service could provide on offered coverage objects.

11.1.1 Core

The Core specification is agnostic of implementation details, hence, access syntax and mechanics are defined by *protocol extensions*: KVP/GET, XML/POST, and XML/SOAP. Rasdaman supports all three, but further on the examples are in *KVP/GET* exclusively, as it is the most straightforward way for constructing requests by appending a standard query string to the service endpoint URL. Commonly, for all operations the KVP/GET request will look as follows:

```
http(s)://<endpoint url>?service=WCS
                        &version=2.0.1
                        &request=<operation>
                        &...
```

Three fundamental operations are defined by the Core:

- **GetCapabilities** - returns overal service information and a list of available coverages; the request looks generally as above, with the <*operation*> being GetCapabilities:

  ```
  http(s)://<endpoint url>?service=WCS&version=2.0.1&
  ↪request=GetCapabilities
  ```

  Example:

  > http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCapabilities

- **DescribeCoverage** - detailed description of a specific coverage:

```
http(s)://<endpoint url>?service=WCS&version=2.0.1&
↪request=DescribeCoverage
                             &coverageId=<coverage id>
```

Example:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&
> request=DescribeCoverage&coverageId=AvgLandTemp

- **GetCoverage** - retreive a whole coverage, or arbitrarily restricted on any of its axes whether by new lower/upper bounds (*trimming*) or at a single index (*slicing*):

```
http(s)://<endpoint url>?service=WCS&version=2.0.1&
↪request=GetCoverage
                             &coverageId=<coverage id>
          [optional]         &subset=<axis>(<lower>,<upper>)
          [optional]         &subset=<axis>(<index>)
          [optional]         &format=<mime type>
```

Example:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCoverage&co
> 90.0,85.3)&subset=ansi("2014-10-01")&format=image/jpeg

11.1.2 Updating

The Transaction extension (WCS-T) specifies the following operations for constructing, maintenance, and removal of coverages on a server: *InsertCoverage*, *UpdateCoverage*, and *DeleteCoverage*.

Rasdaman provides the wcst_import tool to simplify the import of data into analysis-ready coverages (aka datacubes) by generating WCS-T requests as instructed by a simple configuration file.

11.1.3 Processing

The Processing extension enables advanced analytics on coverages through WCPS queries. The request format is as follows:

```
http(s)://<endpoint url>?service=WCS&version=2.0.1&
↪request=ProcessCoverages
                             &query=<wcps query>
```

E.g, calculate the average on the subset from the previous GetCoverage example:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=ProcessCoverages&que
> $c in (AvgLandTemp) return avg($c[Lon(-90.0:85.3), ansi("2014-10-01")])

11.1.4 Range subsetting

The cell values of some coverages consist of multiple components (also known as ranges, bands, channels, fields, attributes). The Range subsetting extension specifies the extraction and/or recombination in possibly different order of one or more bands. This is done by listing the wanted bands or band intervals; e.g *AverageChlorophyllScaled* has Blue, Green, and Red bands and the following recombines them into a Red, Green, Blue order:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCoverage&coverag
> 01-01")&rangesubset=Red,Green,Blue

11.1.5 Scaling

Scaling up or down is a common operation supported by the Scaling extension. An additional GetCoverage parameter indicates the scale factor in several possible ways: as a single number applying to all axes, multiple numbers applying to individual axes, full target scale domain, or per-axis target scale domains. E.g. a single factor to downscale all axes by 4x:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCoverage&coverag
> 10-01")&format=image/jpeg&scaleFactor=0.25

Currently only nearest neighbour interpolation is supported for scaling.

11.1.6 Reprojection

The CRS extension allows to reproject a coverage before retreiving it. For example `AverageChlorophyllScaled` has native CRS EPSG:4326, and the following request will return the result in EPSG:3857:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCoverage&coverag
> 01-01")&outputCrs=http://ows.rasdaman.org/def/crs/EPSG/0/3857

or change the CRS in which subset or scale coordinates are specified:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&
> request=GetCoverage&coverageId=AverageChlorophyllScaled&format=
> image/png&subset=Lon(0,10000000)&subset=Lat(0,20000000)&subset=
> unix(%222015-01-01%22)&subsettingCrs=http://ows.rasdaman.org/def/crs/
> EPSG/0/3857

11.1.7 Interpolation

Reprojection (optionally with subsequent scaling) can be performed with various interpolation methods as enabled by the Interpolation extension:

> http://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCoverage&coverageId=mean_summer_airtemp&outputCrs=http://ows.rasdaman.org/def/crs/EPSG/0/3857&interpolation=http://www.opengis.net/def/interpolation/OGC/1/cubic

Rasdaman supports several interpolation methods as documented *here*.

11.2 WCPS

The OGC Web Coverage Processing Service (WCPS) standard defines a protocol-independent declarative query language for the extraction, processing, and analysis of multi-dimensional coverages representing sensor, image, or statistics data.

The overall execution model of WCPS queries is similar to XQuery FLOWR:

```
for $covIter1 in (covName, ...),
    $covIter2 in (covName, ...),
    ...
let $aliasVar1 := covExpr,
    $aliasVar2 := covExpr,
    ...
where booleanExpr
return processingExpr
```

Any coverage listed in the WCS *GetCapabilities* response can be used in place of `covName`. Multiple `$covIter` essentially translate to nested loops. For each iteration, the `return` clause is evaluated if the result of the `where` clause is `true`. Coverage iterators and alias variables can be freely used in where / return expressions.

Conforming WCPS queries can be submitted to rasdaman as WCS ProcessCoverages requests, e.g:

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1
    &request=ProcessCoverages
    &query=for $covIter in (covName) ...
```

The *WSClient* deployed with every rasdaman installation provides a convenient console for interactively writing and executing WCPS queries: open http://localhost:8080/rasdaman/ows in your Web browser and proceed to the *ProcessCoverages* tab.

Operations can be categorized by the type of data they result in: scalar, coverage, or metadata.

11.2.1 Scalar operations

- **Standard operations** applied on scalar operands return scalar results:

Operation category	Operations
Arithmetic	`+ - * / abs round`
Exponential	`exp log ln pow sqrt`

Trigonometric	`sin cos tan sinh cosh tanh`
	`arcsin arccos arctan`

Comparison	`> < >= <= = !=`
Logical	`and or xor not bit overlay`
Select field from multiband value	`.`
Create multiband value	`{ bandName: value; ..., bandName: value }`
Type casting	`(baseType) value` where baseType is one of: boolean, [unsigned] char / short / int / long, float, double, complex, complex2

- **Aggregation operations** summarize coverages into a scalar value.

Aggregation type	Function / Expression
Of numeric coverages	`avg`, `add` (or alias `sum`), `min`, `max`
Of boolean coverages	`count` number of true values; `some`/`all` = true if some/all values are true
General condenser	`condense` *op* `over` $iterVar axis(lo:hi), . . . [`where` boolScalarExpr] `using` scalarExpr

The *general condenser* aggregates values across an iteration domain with a condenser operation *op* (one of `+`, `*`, `max`, `min`, `and`, or `or`). For each coordinate in the iteration domain defined by the `over` clause, the scalar expression in the `using` clause is evaluated and added to the final aggregated result; the optional `where` clause allows to filter values from the aggregation.

11.2.2 Coverage operations

- **Standard operations** applied on coverage (or mixed coverage and scalar) operands return coverage results. The operation is applied pair-wise on each cell from the coverage operands, or on the scalars and each cell from the coverage in case some of the operands are scalars. All coverage operands must have matching domains and CRS.
- **Subsetting** allows to select a part of a coverage (or crop it to a smaller domain):

  ```
  covExpr[ axis1(lo:hi), axis2(slice), axis3:crs(...), ... ]
  ```

  1. `axis1` in the result is reduced to span from coordinate `lo` to `hi`. Either or both `lo` and `hi` can be indicated as `*`, corresponding to the minimum or maximum bound of that axis.
  2. `axis2` is restricted to the exact slice coordinate and removed from the result.
  3. `axis3` is subsetted in coordinates specified in the given `crs`. By default coordinates must be given in the native CRS of `C`.
- **Extend** is similar to subsetting but can be used to enlarge a coverage with null values as well, i.e. lo and hi can extend beyond the min/max bounds of a particular axis; only trimming is possible:

  ```
  extend( covExpr, { axis1(lo:hi), axis2:crs(lo:hi), ... } )
  ```
- **Scale** is like extend but it resamples the current coverage values to fit the new domain:

  ```
  scale( covExpr, { axis1(lo:hi), axis2:crs(lo:hi), ... } )
  ```

  Currently only nearest neighbour interpolation is supported for scaling.
- **Reproject** allows to project a 2D coverage with geo X/Y axes by a CRS:

  ```
  crsTransform( covExpr, { axisX:outputCRS, axisY:outputCRS }, {␣
  ↪interpolation } )
  ```

  or shorthand version

  crsTransform(covExpr, “outputCRS”, { interpolation })

  For supported interpolation methods see the options for *resampleAlg parameter*.
- **Conditional evaluation** is possible with the `switch` statement:

  ```
  switch
    case boolCovExpr return covExpr
    case boolCovExpr return covExpr
    ...
    default return covExpr
  ```
- **General coverage constructor** allows to create a coverage given a domain, where for each coordinate in the domain the value is dynamically calculated from a value expression which potentially references the iterator variables:

```
coverage covName
over $iterVar axis(lo:hi), ...
values scalarExpr
```

- **General condenser on coverages** is same as the scalar general condenser, except that in the `using` clause we have a coverage expression. The coverage values produced in each iteration are cell-wise aggregated into a single result coverage.

```
condense op
over $iterVar axis(lo:hi), ...
[ where boolScalarExpr ]
values covExpr
```

- **Encode** allows to export coverages in a specified data format, e.g:

```
 encode(covExpr, "image/jpeg")

WCPS supports ``application/gml+xml`` format corresponding to␣
↪OGC WCS ``GetCoverage`` request.
Many further formats are supported, see :ref:`here <rasql-
↪encode-function-data-format>` for details.
```

11.2.3 Atomic types

The set of atomic types for Coverage range field data types according to OGC WCPS standard. See *rasdaman atomic types* for comparison.

Table 11.1: Coverage atomic range field data types

type name	size	description
`boolean`	1 bit	true (nonzero value), false (zero value)
`char`	8 bit	signed integer
`unsigned char`	8 bit	unsigned integer
`short`	16 bit	signed integer
`unsigned short`	16 bit	unsigned integer
`int`	32 bit	signed integer
`unsigned int`	32 bit	unsigned integer
`float`	32 bit	single precision floating point
`double`	64 bit	double precision floating point
`cint16`	32 bit	complex of 16 bit signed integers
`cint32`	64 bit	complex of 32 bit signed integers
`complex`	64 bit	single precision floating point complex
`complex2`	128 bit	double precision floating point complex

11.2.4 Metadata operations

Several functions allow to extract metadata information about a coverage `C`:

Metadata function	Result
imageCrsDomain(C, a)	Grid (lo, hi) bounds for axis a
imageCrsDomain(C, a).x	Where x is one of `lo` or `hi` returning the lower or upper bounds respectively
domain(C, a, c)	Geo (lo, hi) bounds for axis a in CRS c returning the lower and upper bounds respectively
domain(C, a, c).x	Where x is one of `lo` or `hi` returning the lower or upper bounds respectively
domain(C, a)	Geo (lo, hi) bounds for axis a returning the lower and upper bounds respectively
domain(C, a).x	Where x is one of `lo` or `hi` returning the lower or upper bounds respectively
domain(C)	List of comma-separated axes and their bounds according to coverage's CRS orders respectively. Each list element contains an axis a with the lower and upper bounds in the axis CRS
crsSet(C)	Set of CRS identifiers
imageCrs(C)	Return the grid CRS (CRS:1)
nullSet(C)	Set of null values
cellCount(C)	Total number of grid pixels

11.2.5 Comments

WCPS supports SQL-like commenting styles:

- Single line comments start with `--`. Any text following `--` to the end of the line will be ignored. Example:

```
return encode($c, "image/png") -- Output encoded as 2D image
```

- Multi-line comments start with `/*` and end with `*/`. Any text between `/*` and `*/` are ignored. Example:

```
/*
    Output encoded as 2D image; result can be viewed in
    Web browsers or image viewer tools.
```

(continues on next page)

(continued from previous page)

```
*/
return encode($c, "image/png")
```

11.3 WMS

The OGC Web Map Service (WMS) standard defines map portrayal on geo-spatial data. In rasdaman, a WMS service can be enabled on any coverage, including 3-D or higher dimensional; the latest 1.3.0 version is supported.

rasdaman supports two operations: *GetCapabilities*, *GetMap* from the standard. We will not go into the details, as users do not normally hand-write WMS requests, but let a client tool or library generate them instead. Check the *Clients* section for some examples.

11.4 Clients

11.4.1 Rasdaman WSClient

WSClient is a web-client application to interact with WCS (version 2.0.1) and WMS (version 1.3.0) compliant servers. Once rasdaman is installed it is usually accessible at `http://localhost:8080/rasdaman/ows`; a publicly accessible example is available at http://ows.rasdaman.org/rasdaman/ows. The client has three main tabs: `OGC Web Coverage Service (WCS)`, `OGC Web Map Service (WMS)` and `Admin`. Further on, the functionality in each tab is described in details.

WCS

There are sub-tabs for each of OGC WCS standard requests: GetCapabilities, DescribeCoverage, GetCoverage, ProcessCoverages.

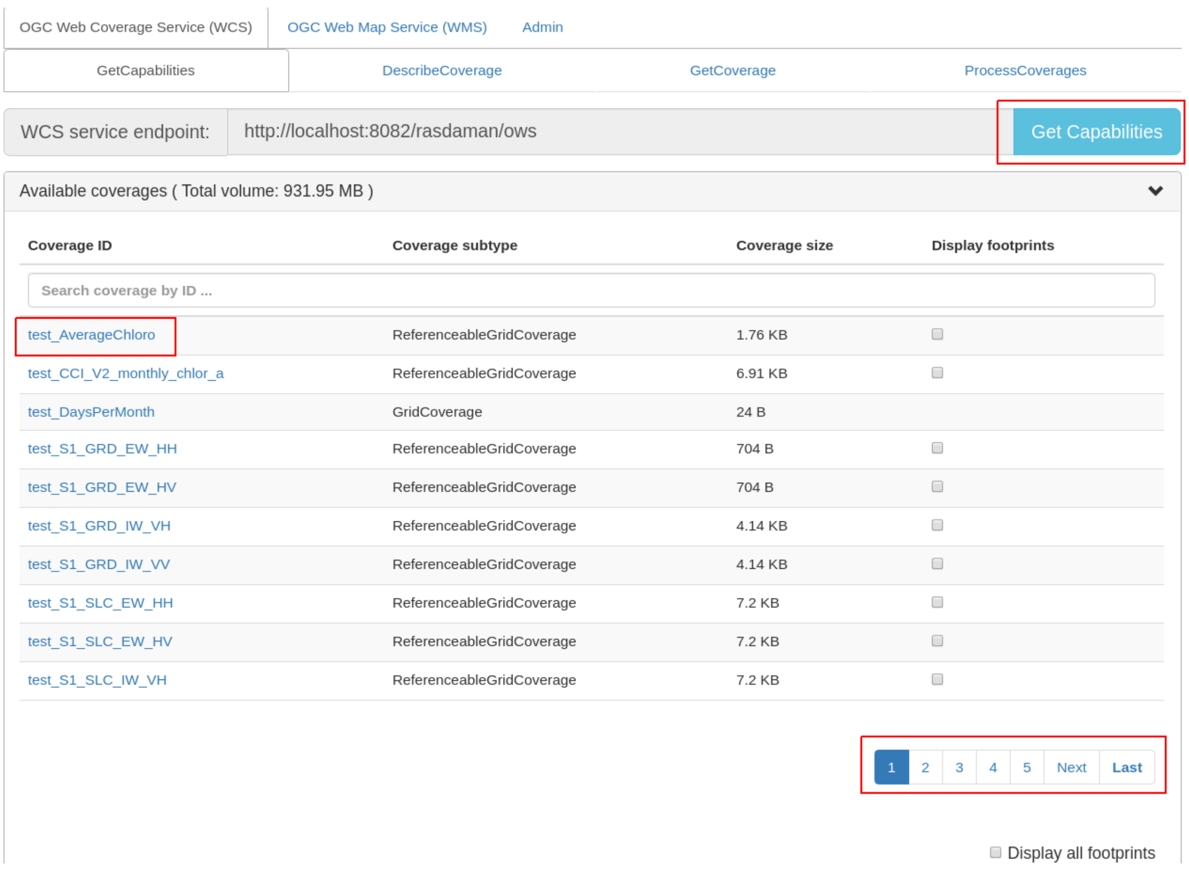
GetCapabilities

This is the default tab when accessing the WSClient. It lists all coverages available at the specified WCS endpoint. Clicking on the `Get Capabilities` button will reload the coverages list. One can also search a coverage by typing the first characters of its name in the text box. Clicking on a coverage name will move to `DescribeCoverage` tab to view its metadata.

If a coverage is geo-referenced, a checkbox will be visible in the `Display footprints` column, allowing to view the coverage's geo bounding box (in EPSG:4326) on the globe below.

At the bottom the metadata of the OGC WCS service endpoint are shown. These metadata can be changed in the `Admin -> OWS Metadata Management` tab. Once updated in the admin tab, click on `Get Capabilities` button to see the new metadata.

DescribeCoverage



WCS service endpoint: http://localhost:8082/rasdaman/ows

Available coverages (Total volume: 931.95 MB)

Coverage ID	Coverage subtype	Coverage size	Display footprints
test_AverageChloro	ReferenceableGridCoverage	1.76 KB	☐
test_CCI_V2_monthly_chlor_a	ReferenceableGridCoverage	6.91 KB	☐
test_DaysPerMonth	GridCoverage	24 B	
test_S1_GRD_EW_HH	ReferenceableGridCoverage	704 B	☐
test_S1_GRD_EW_HV	ReferenceableGridCoverage	704 B	☐
test_S1_GRD_IW_VH	ReferenceableGridCoverage	4.14 KB	☐
test_S1_GRD_IW_VV	ReferenceableGridCoverage	4.14 KB	☐
test_S1_SLC_EW_HH	ReferenceableGridCoverage	7.2 KB	☐
test_S1_SLC_EW_HV	ReferenceableGridCoverage	7.2 KB	☐
test_S1_SLC_IW_VH	ReferenceableGridCoverage	7.2 KB	☐

Figure 11.1: List of coverages shown on the GetCapabilities tab.

Here the full description of a selected coverage can be seen. One can type the first few characters to search for a coverage id and click on `Describe Coverage` button to view its OGC WCS metadata.

Once logged in as admin, it's possible to replace the metadata with one from a valid XML or JSON file.

GetCoverage

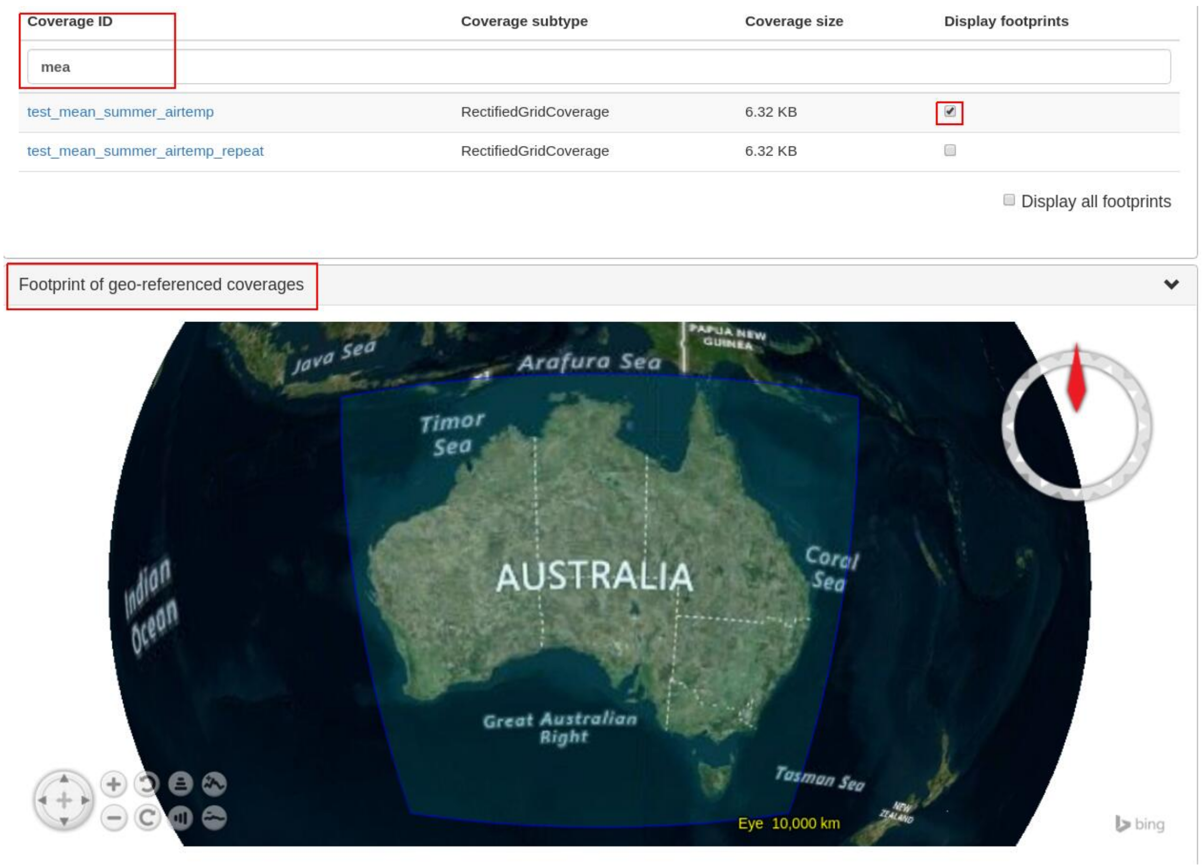
Downloading coverage *data* can be done on this tab (or the next one, ProcessCoverages). It's similiarly possible search for a coverage id in the text box and click on `Select Coverage` button to view its boundaries. Depending on the coverage dimension, one can do trim or slice subsets on the corresponding axes to select an area of interest. The output format can be selected (provided it supports the output dimension). Finally, clicking on `Get Coverage` button will download the coverage.

In addition, further parameters can be specified as supported by the WCS extensions, e.g. scaling factor, output CRS, subset of ranges (bands), etc.

ProcessCoverages

WCPS queries can be typed in a text box. Once `Excute` is clicked, the result will be

- displayed on the output console if it's a scalar or the query was prefixed with `image>>` (for 2D png/jpeg) or `diagram>>` for (1D csv/json);
- otherwise it will be downloaded.

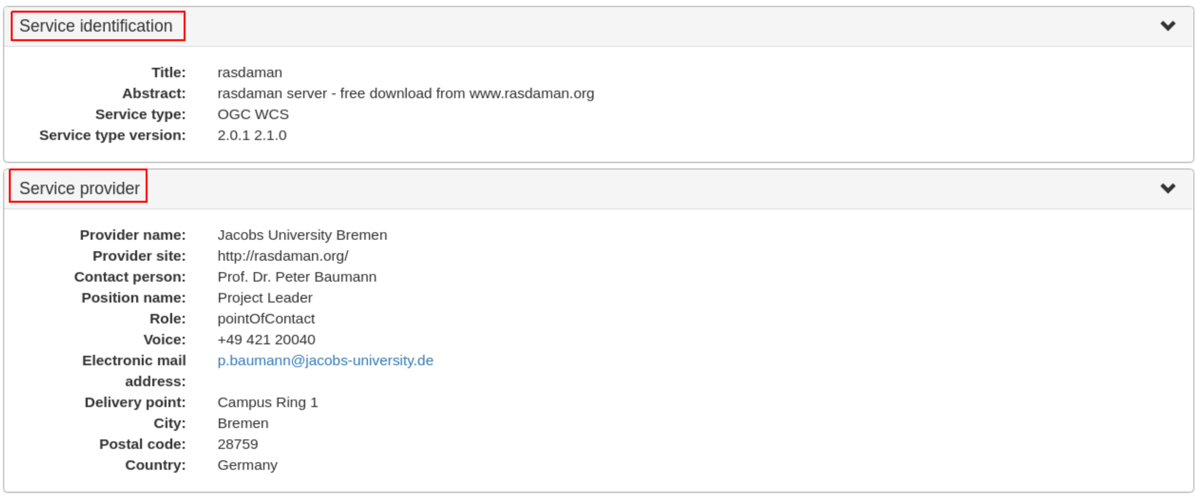


Coverage ID	Coverage subtype	Coverage size	Display footprints
mea			
test_mean_summer_airtemp	RectifiedGridCoverage	6.32 KB	✓
test_mean_summer_airtemp_repeat	RectifiedGridCoverage	6.32 KB	☐

☐ Display all footprints

Footprint of geo-referenced coverages

Figure 11.2: Selected coverage footprints shown on a globe.



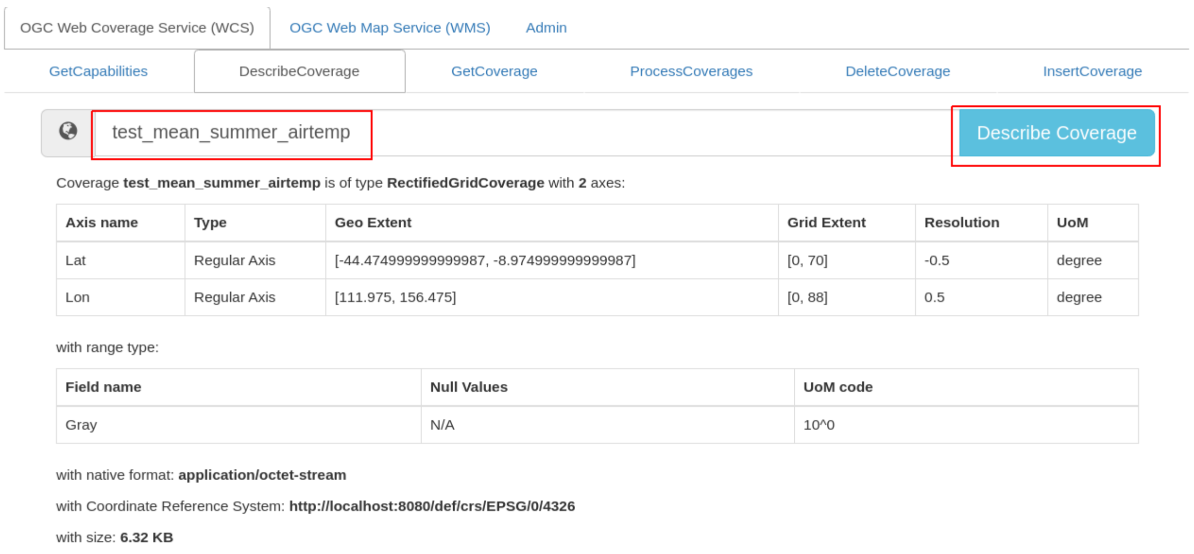
Service identification

Title:	rasdaman
Abstract:	rasdaman server - free download from www.rasdaman.org
Service type:	OGC WCS
Service type version:	2.0.1 2.1.0

Service provider

Provider name:	Jacobs University Bremen
Provider site:	http://rasdaman.org/
Contact person:	Prof. Dr. Peter Baumann
Position name:	Project Leader
Role:	pointOfContact
Voice:	+49 421 20040
Electronic mail address:	p.baumann@jacobs-university.de
Delivery point:	Campus Ring 1
City:	Bremen
Postal code:	28759
Country:	Germany

Figure 11.3: WCS service metadata.



Coverage **test_mean_summer_airtemp** is of type **RectifiedGridCoverage** with **2** axes:

Axis name	Type	Geo Extent	Grid Extent	Resolution	UoM
Lat	Regular Axis	[-44.474999999999987, -8.974999999999987]	[0, 70]	-0.5	degree
Lon	Regular Axis	[111.975, 156.475]	[0, 88]	0.5	degree

with range type:

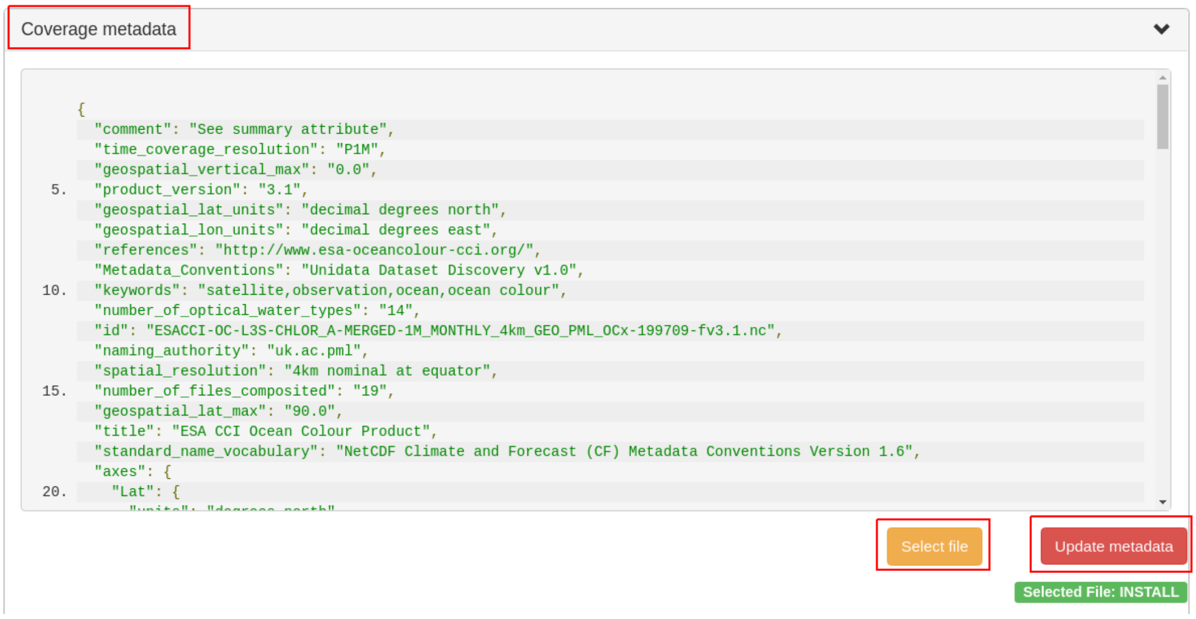
Field name	Null Values	UoM code
Gray	N/A	10^0

with native format: **application/octet-stream**

with Coordinate Reference System: **http://localhost:8080/def/crs/EPSG/0/4326**

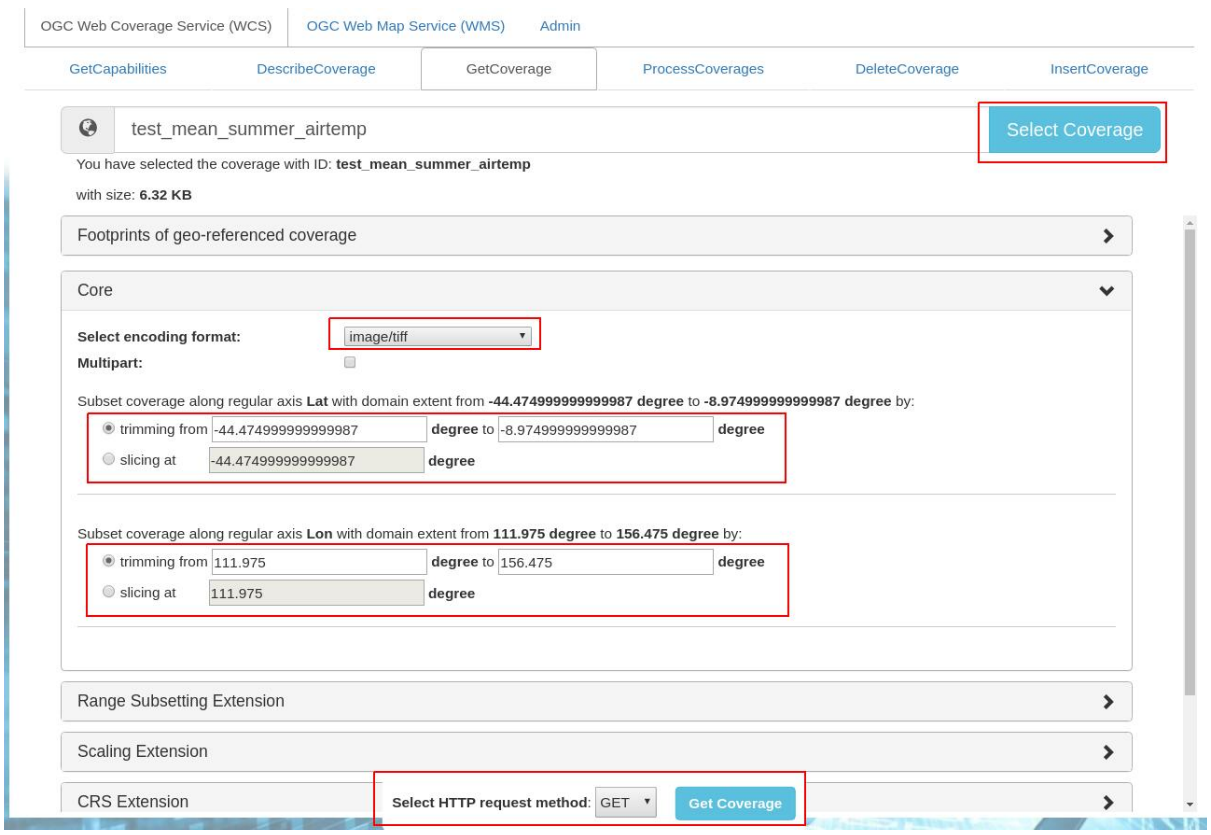
with size: **6.32 KB**

Figure 11.4: Showing full description of a coverage.



```
{
  "comment": "See summary attribute",
  "time_coverage_resolution": "P1M",
  "geospatial_vertical_max": "0.0",
  "product_version": "3.1",
  "geospatial_lat_units": "decimal degrees north",
  "geospatial_lon_units": "decimal degrees east",
  "references": "http://www.esa-oceancolour-cci.org/",
  "Metadata_Conventions": "Unidata Dataset Discovery v1.0",
  "keywords": "satellite,observation,ocean,ocean colour",
  "number_of_optical_water_types": "14",
  "id": "ESACCI-OC-L3S-CHLOR_A-MERGED-1M_MONTHLY_4km_GEO_PML_OCx-199709-fv3.1.nc",
  "naming_authority": "uk.ac.pml",
  "spatial_resolution": "4km nominal at equator",
  "number_of_files_composited": "19",
  "geospatial_lat_max": "90.0",
  "title": "ESA CCI Ocean Colour Product",
  "standard_name_vocabulary": "NetCDF Climate and Forecast (CF) Metadata Conventions Version 1.6",
  "axes": {
    "Lat": {
```

Figure 11.5: Updating the metadata of a coverage.



test_mean_summer_airtemp

You have selected the coverage with ID: **test_mean_summer_airtemp**

with size: **6.32 KB**

Select encoding format: image/tiff

Multipart:

Subset coverage along regular axis **Lat** with domain extent from **-44.474999999999987 degree** to **-8.974999999999987 degree** by:

trimming from -44.474999999999987 **degree** to -8.974999999999987 **degree**

slicing at -44.474999999999987 **degree**

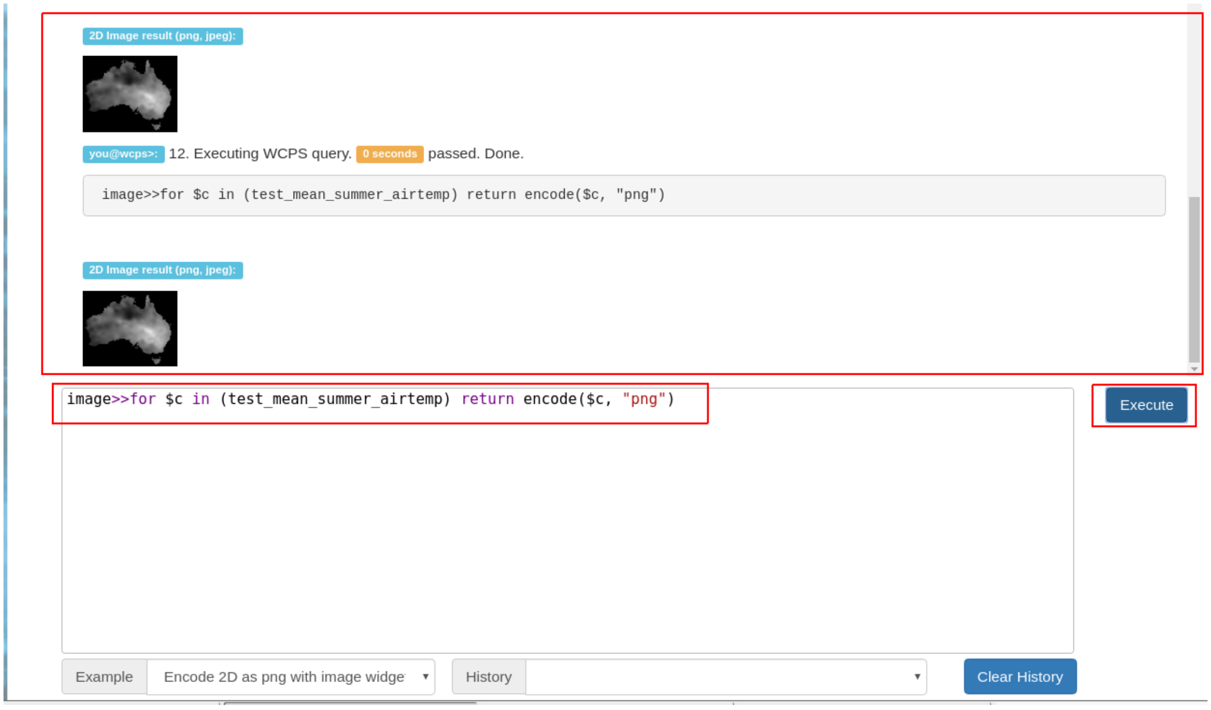
Subset coverage along regular axis **Lon** with domain extent from **111.975 degree** to **156.475 degree** by:

trimming from 111.975 **degree** to 156.475 **degree**

slicing at 111.975 **degree**

Select HTTP request method: GET

Figure 11.6: Downloading a subset of a coverage, encoded in image/tiff.


2D Image result (png, jpeg):
2D Image result (png, jpeg):
Execute
Example
Encode 2D as png with image widge
History
Clear History

you@wcps>: 12. Executing WCPS query. 0 seconds passed. Done.

```
image>>for $c in (test_mean_summer_airtemp) return encode($c, "png")
```

```
image>>for $c in (test_mean_summer_airtemp) return encode($c, "png")
```

Figure 11.7: Query and output areas on the ProcessCoverages tab.

DeleteCoverage

This tab allows to *delete* a specific coverage from the server. It is only visible when logged in the `Admin` tab.

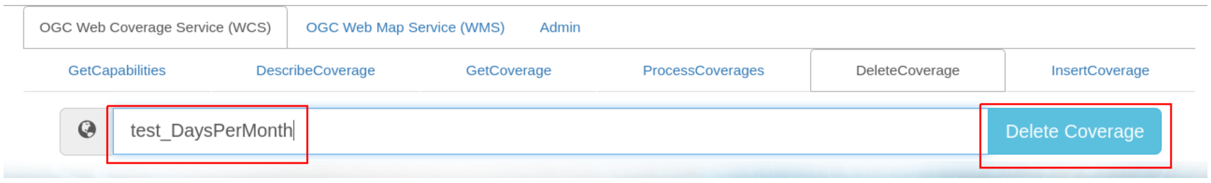


Figure 11.8: Deleting coverage test_DaysPerMonth.

InsertCoverage

Similarly, this tab is only visible when logged in the `Admin` tab. To insert a coverage, a URL pointing to a valid coverage definition according to the WCS-T standard needs to be provided. Clicking on `Insert Coverage` button will invoke the correct WCS-T request on the server.

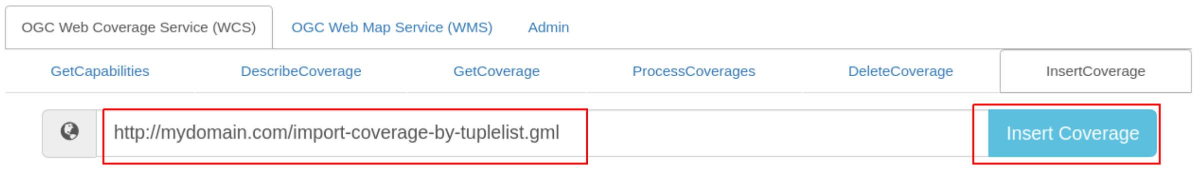

OGC Web Coverage Service (WCS)
OGC Web Map Service (WMS)
Admin
GetCapabilities
DescribeCoverage
GetCoverage
ProcessCoverages
DeleteCoverage
InsertCoverage
http://mydomain.com/import-coverage-by-tuplelist.gml
Insert Coverage

Figure 11.9: Inserting a coverage given a URL pointing to a valid GML document.

WMS

This tab contain sub-tabs which are related to the supported OGC WMS requests.

GetCapabilities

This tab lists the available layers on the specified server. To reload the list, click on the `Get Capabilities` button. Clicking on a layer name will move to `DescribeLayer` tab to view its description.

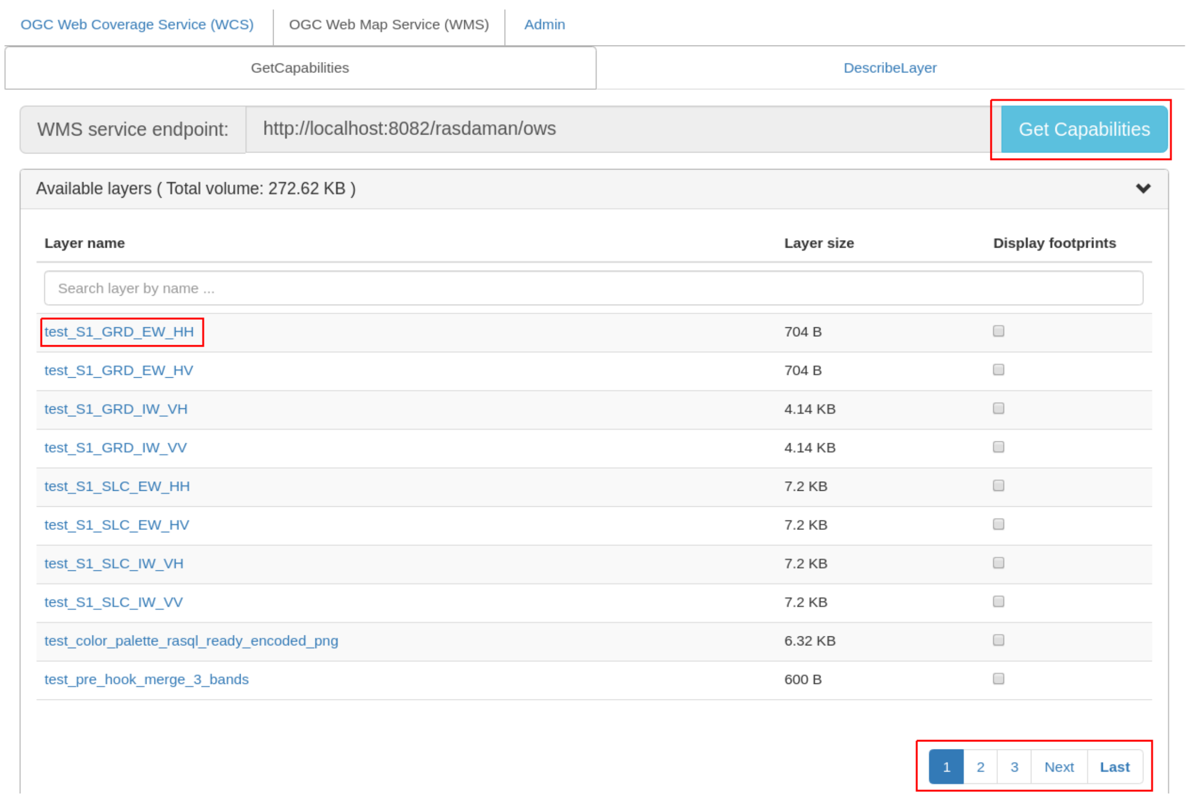
Similar to the WCS GetCapabilities tab, it's possible to search for layer names, or show their footprints.

DescribeLayer

Here the full description of a selected layer is shown. One can type the first few characters to search for a layer name and click on `Describe Layer` button to view its OGC WMS metadata.

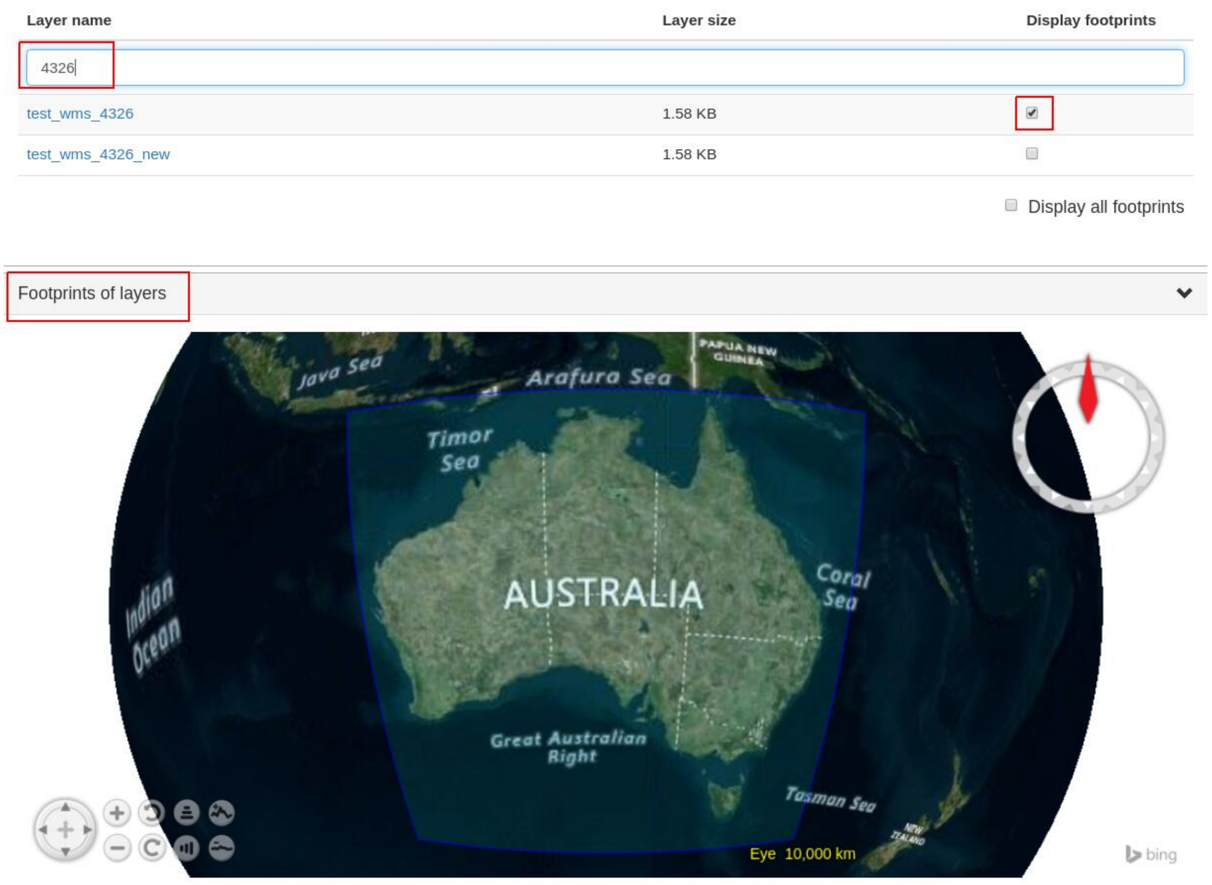
Depending on layer's dimension, one can click on `show layer` button and interact with axes' sliders to view a layer's slice on the globe below. Click on the `hide layer` button to hide the displayed layer on the globe.

Once logged in as admin, managing WMS styles is possible on this tab. To create a style, it is required to input various parameters along with a rasql or WCPS query fragment, which are applied on every GetMap request if the style is active. Afterwards, click on `Insert Style`


OGC Web Coverage Service (WCS)
OGC Web Map Service (WMS)
Admin
GetCapabilities
DescribeLayer
WMS service endpoint:
http://localhost:8082/rasdaman/ows
Get Capabilities
Available layers (Total volume: 272.62 KB)
Search layer by name ...
1
2
3
Next
Last

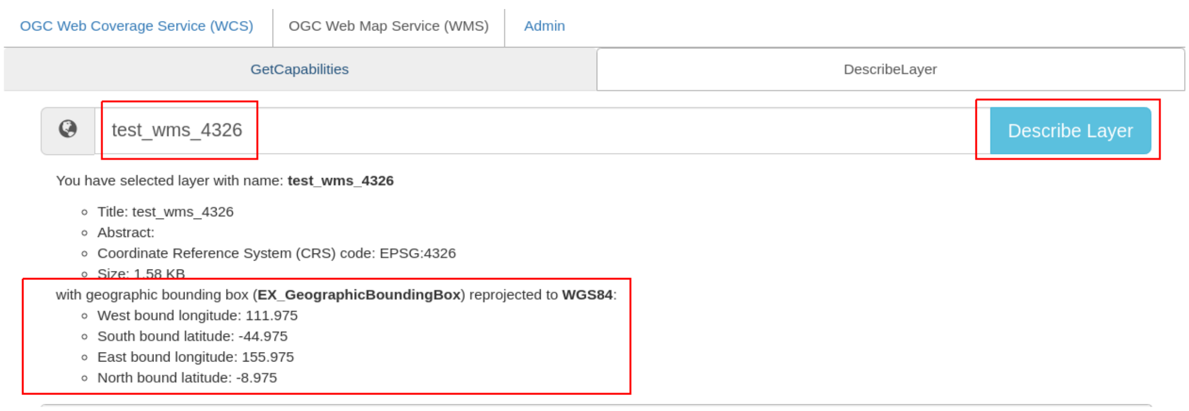
Layer name	Layer size	Display footprints
test_S1_GRD_EW_HH	704 B	☐
test_S1_GRD_EW_HV	704 B	☐
test_S1_GRD_IW_VH	4.14 KB	☐
test_S1_GRD_IW_VV	4.14 KB	☐
test_S1_SLC_EW_HH	7.2 KB	☐
test_S1_SLC_EW_HV	7.2 KB	☐
test_S1_SLC_IW_VH	7.2 KB	☐
test_S1_SLC_IW_VV	7.2 KB	☐
test_color_palette_rasql_ready_encoded_png	6.32 KB	☐
test_pre_hook_merge_3_bands	600 B	☐

Figure 11.10: List of layers shown on the GetCapabilities tab.



Layer name	Layer size	Display footprints
test_wms_4326	1.58 KB	✓
test_wms_4326_new	1.58 KB	☐

Figure 11.11: Selected layer footprints shown on a globe.



You have selected layer with name: **test_wms_4326**

- Title: test_wms_4326
- Abstract:
- Coordinate Reference System (CRS) code: EPSG:4326
- Size: 1.58 KB

with geographic bounding box (**EX_GeographicBoundingBox**) reprojected to **WGS84**:

- West bound longitude: 111.975
- South bound latitude: -44.975
- East bound longitude: 155.975
- North bound latitude: -8.975

Figure 11.12: Showing full description of a layer.

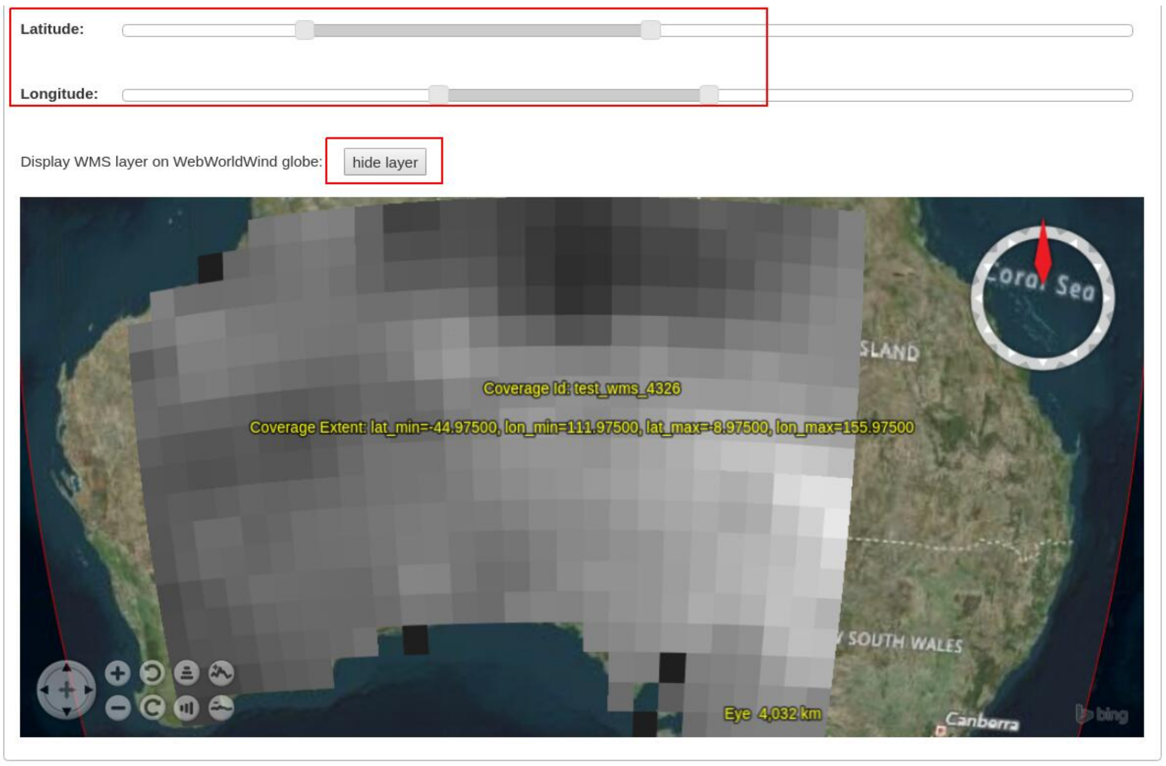
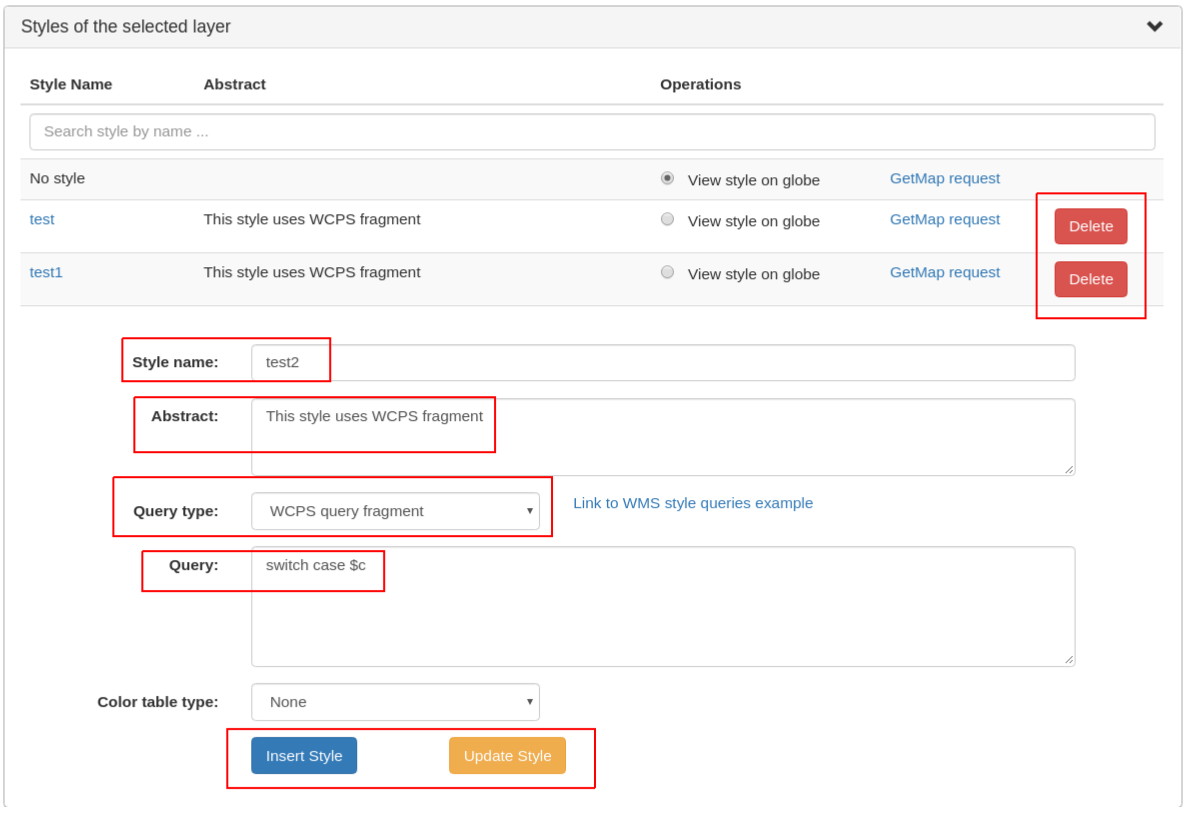

Latitude:
Longitude:
Display WMS layer on WebWorldWind globe:
hide layer
Coverage Id: test_wms_4326
Coverage Extent: lat_min=-44.97500, lon_min=111.97500, lat_max=-8.97500, lon_max=155.97500
Coral Sea
SLAND
SOUTH WALES
Canberra
Eye 4,032 km
bing

Figure 11.13: Showing/hiding a layer on the map.

to insert a new style or `Update Style` to update an existing style of the current selected layer. One can also delete an existing style by clicking on the `Delete` button corresponding to a style name.



Styles of the selected layer

Style Name	Abstract	Operations		
No style		● View style on globe	GetMap request	
test	This style uses WCPS fragment	○ View style on globe	GetMap request	Delete
test1	This style uses WCPS fragment	○ View style on globe	GetMap request	Delete

Style name: test2

Abstract: This style uses WCPS fragment

Query type: WCPS query fragment

Query: switch case $c

Color table type: None

Figure 11.14: Style management on the DescribeLayer tab.

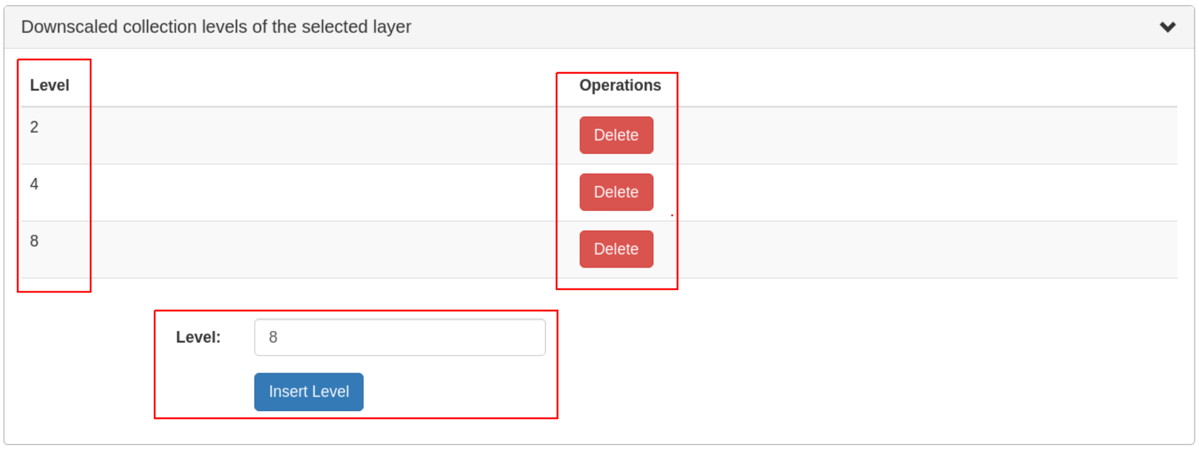
Finally, once logged in as admin, managing downscaled collection levels of a WMS layer is also possible on this tab. To create a new level, it is required to input level parameter (positive number). Afterwards, click on `Insert Level` to insert a new downscaled collection level of the current selected layer. One can also delete an existing level by clicking on the `Delete` button corresponding to a downscaled collection level.

11.4.2 NASA WebWorldWind

- Simple example to setup a web page with a map from a WMS server using WebWorld-Wind:

```
<html>
  <head>
    <script src="https://files.worldwind.arc.nasa.gov/
↪artifactory/web/0.9.0/worldwind.min.js"></script>
    <script>
      document.addEventListener("DOMContentLoaded",␣
↪function(event) {
        WorldWind.Logger.setLoggingLevel(WorldWind.Logger.LEVEL_
↪WARNING);
```

(continues on next page)



Level	Operations
2	Delete
4	Delete
8	Delete

Level: 8

Figure 11.15: Downscaled collection level management on the DescribeLayer tab.

(continued from previous page)

```
        var wwd = new WorldWind.WorldWindow("canvasOne");
        var layers = [{
          layer: new WorldWind.BingRoadsLayer(null),
          enabled: true
        }, {
          layer: new WorldWind.CoordinatesDisplayLayer(wwd),
          enabled: true
        }, {
          layer: new WorldWind.ViewControlsLayer(wwd),
          enabled: true
        }];

        for (var l = 0; l < layers.length; l++) {
          wwd.addLayer(layers[l].layer);
        }

        var layerNamesToRequest = ["AvgTemperatureColorScaled"];
        var config = {
          title: "AvgTemperatureColorScaled", version: "1.3.0",
          service: "http://ows.rasdaman.org/rasdaman/ows",
          layerNames: layerNamesToRequest,
          // min Lat, max Lat, min Long, max Long of the␣
↪requesting layer
          sector: new WorldWind.Sector(-90, 90, -180, 180),
          levelZeroDelta: new WorldWind.Location(36, 36),
          numLevels: 15, format: "image/png", styleNames: "",␣
↪size: 256
        };

        var wmsLayer = new WorldWind.WmsLayer(config);
        wmsLayer.enabled = true;
        wwd.addLayer(wmsLayer);
      });
```

(continues on next page)

(continued from previous page)

```
    </script>
  </head>
  <body>
      <canvas id="canvasOne" style="width: 100%; height: 100%;">
↪ </canvas>
  </body>
</html>
```

11.4.3 Python / Jupter Notebook

OWSLib

OWSLib is a Python package that helps with programming clients for OGC services such as WCS, WCPS, or WMS. To install it follow the official installation instructions. Example usage for WCS follows below.

```
>>> # Import OWSLib in Python once installed
... from owslib.wcs import WebCoverageService

>>> # Create coverage object
... my_wcs = WebCoverageService('http://ows.rasdaman.org/rasdaman/
↪ows',
...                               version='2.0.1')

>>> # Get list of coverages
... print my_wcs.contents.keys()
['RadianceColor', 'test_irr_cube_2', 'test_mean_summer_airtemp',
 'test_double_1d', 'INSPIRE_EL', 'AverageChlorophyllScaled',
↪'INSPIRE_OI_RGB',
 'Temperature4D', 'INSPIRE_OI_IR', 'visible_human', 'INSPIRE_WS_LC',
 'meris_lai', 'climate_earth', 'mean_summer_airtemp', 'multiband',
 'ls8_coastal_aerosol', 'NN3_3', 'NN3_2', 'NN3_1', 'NN3_4',
 'AvgTemperatureColorScaled', 'AverageChloroColorScaled', 'lena',
 'Germany_DTM', 'climate_cloud', 'FiLCCoverageBit',
↪'AverageChloroColor',
 'LandsatMultiBand', 'RadianceColorScaled', 'AvgLandTemp', 'NIR',
↪'BlueMarbleCov']

>>> # Get geo-bounding boxes and native CRS
... my_wcs.contents['AverageChlorophyllScaled'].boundingboxes
[{'nativeSrs': 'http://ows.rasdaman.org/def/crs-compound?
  1=http://ows.rasdaman.org/def/crs/EPSG/0/4326&
  2=http://ows.rasdaman.org/def/crs/OGC/0/UnixTime',
  'bbox': (-90.0, -180.0, 90.0, 180.0)}]

>>> # Get axis labels
... my_wcs.contents['AverageChlorophyllScaled'].grid.axislabels
```

(continues on next page)

(continued from previous page)

```
['Lat', 'Long', 'unix']

>>> # Get dimension
... my_wcs.contents['AverageChlorophyllScaled'].grid.dimension
3

>>> # Get grid lower and upper bounds
... my_wcs.contents['AverageChlorophyllScaled'].grid.lowlimits
['0', '0', '0']

>>> my_wcs.contents['AverageChlorophyllScaled'].grid.highlimits
['119', '239', '5']

>>> # Get offset vectors for geo axes
... my_wcs.contents['AverageChlorophyllScaled'].grid.offsetvectors
[['-1.5', '0', '0'], ['0', '1.5', '0'], ['0', '0', '1']]

>>> # For coverage with time axis get the date time values
... my_wcs.contents['AverageChlorophyllScaled'].timepositions
[datetime.datetime(2015, 1, 1, 0, 0), datetime.datetime(2015, 2, 1,␣
↪0, 0),
 datetime.datetime(2015, 3, 1, 0, 0), datetime.datetime(2015, 4, 1,␣
↪0, 0),
 datetime.datetime(2015, 5, 1, 0, 0), datetime.datetime(2015, 7, 1,␣
↪0, 0)]
```

rasdapy3

rasdapy3 is a client API for rasdaman that enables building and executing rasql queries within python. Best practice code snippets are also provided.

wcps_rasdaman.py

wcps_rasdaman.py is a python client which sends a WCPS query to a rasdaman server and wraps the response for further use depending on the response format chosen in the query.

11.4.4 Access from R

Accessing rasdaman from R is possible in three ways right now:

- *RRasdaman* enables connecting to rasdaman, executing rasql queries, and retreiving results. Note that it is *only* for rasql queries, so it is not suitable for querying geo-referenced coverages.
- CubeR allows convenient executiong of WCPS queries directly from R. Check also this accompanying presentation.

- ows4R provides an interface to OGC Web services, including Web Coverage Service (WCS) which is supported by rasdaman. Steps to install `ows4R` package and its dependencies on Ubuntu 20.04:

  ```
  sudo apt-get install libsodium-dev libudunits2-dev

  sudo R

  install.packages("sodium")
  install.packages("keyring")
  install.packages("geometa")
  install.packages("units")
  install.packages("sf")
  install.packages("ows4R")
  ```

  For more details check the ows4R WCS tutorial.

11.4.5 OpenLayers

Simple example to setup a web page with a map from a WMS server using OpenLayers:

```
<html>
  <head>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.
↪com/ajax/libs/openlayers/3.8.2/ol.css"></link>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
↪openlayers/3.8.2/ol.js"></script>
    <script>
     document.addEventListener("DOMContentLoaded",␣
↪function(event) {
        var layers = [
          new ol.layer.Tile({
            source: new ol.source.TileWMS({
              url: "https://ahocevar.com/geoserver/wms",
              params: {'LAYERS': 'ne:NE1_HR_LC_SR_W_DR'}
            })
          }),
          new ol.layer.Tile({
            source: new ol.source.TileWMS({
              url: "http://ows.rasdaman.org/rasdaman/ows",
              params: {'LAYERS': 'AvgTemperatureColorScaled
↪'}
            })
          })
        ];
        var map = new ol.Map({
          layers: layers,
          target: 'map',
          view: new ol.View({
```

(continues on next page)

(continued from previous page)

```
            center: [7.5, 53.15178], projection : "EPSG:4326
↪", zoom: 6
          })
        });
      });
    </script>
  </head>
  <body>
    <div id="map" style="width: 100%; height: 95vh"> </div>
  </body>
</html>
```

11.4.6 Leaflet

Simple example to setup a web page with a map from a WMS server using Leaflet:

```
<html>
  <head>
    <link rel="stylesheet" href="https://unpkg.com/
↪leaflet@1.6.0/dist/leaflet.css"/>
    <script src="https://unpkg.com/leaflet@1.6.0/dist/
↪leaflet.js"></script>
    <script>
      document.addEventListener("DOMContentLoaded",␣
↪function(event) {
        var map = new L.Map('map', {
          center: new L.LatLng(40, 52),
          zoom: 3, attributionControl: true, zoomControl:␣
↪true, minZoom: 2
        });
        var wmsLayer = L.tileLayer.wms("http://ows.rasdaman.
↪org/rasdaman/ows", {
          version: '1.3.0', layers:
↪'AvgTemperatureColorScaled', format: 'image/png'
        });
        map.addLayer(wmsLayer);
      });
    </script>
  </head>
  <body>
    <div id="map" style="width: 100%; height: 100%;"> </div>
  </body>
</html>
```

11.4.7 QGIS

11.4.8 Command-line tools

It's straightforward to make individual OGC WCS / WCPS / WMS requests from the terminal. Examples with `curl` follow.

- Make a GetCapabilities request:

```
curl "http://ows.rasdaman.org/rasdaman/ows\
?service=WCS&version=2.0.1&request=GetCapabilities"
```

- Execute a WCPS query with a ProcessCoverages request:

```
curl "http://ows.rasdaman.org/rasdaman/ows" --out test.png --
↪data-urlencode \
'service=WCS&version=2.0.1&request=ProcessCoverages&query=\
for c in (mean_summer_airtemp) return encode(c, "png")'
```

- Upload files to be processed with `decode()` operator, see *here*.

When the server requires basic authentication for a request, the rasdaman user credentials can be specified with the `--user` option, e.g.

```
curl --user "rasadmin:rasadmin" \
     "http://localhost:8080/rasdaman/ows?
      service=WCS&version=2.0.1&request=DeleteCoverage&
↪coverageId=test_coverage"
```

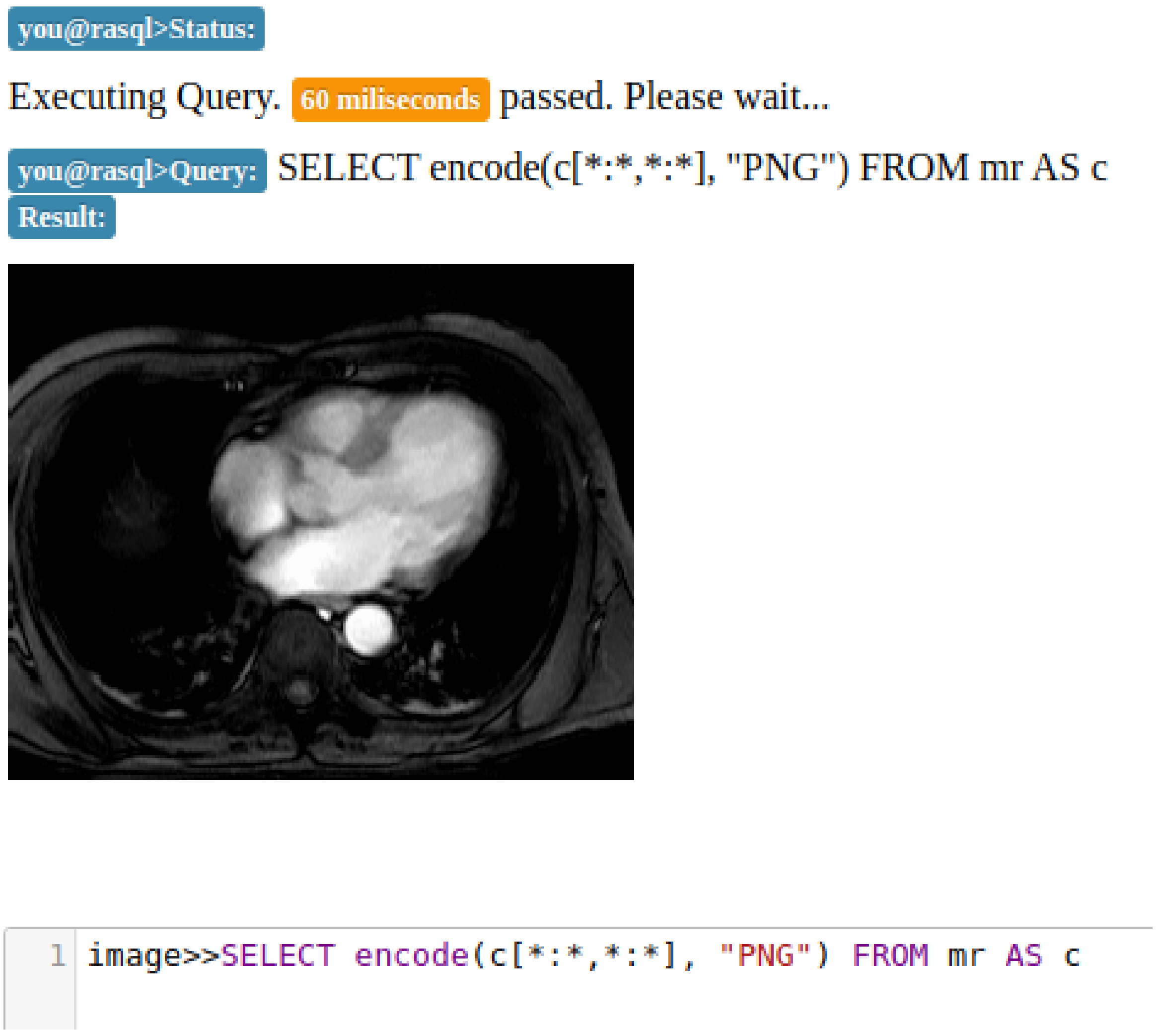
11.4.9 Rasql Web Console

The rasql web console is installed by rasdaman in `$RMANHOME/share/rasdaman/www/rasql-web-console`. It requires petascope to be running in the background in order to execute queries.

Various widgets are available, with the most commonly-used being:

- `image` to visualize a 2D image result, e.g. `image>>select encode(..., "jpeg") from ...`
- `diagram` on csv encoded data, e.g. `diagram(type=area, width=300)>>select encode(..., "csv") from ...`
- `text` to visualize a text result, e.g. `text>>select dbinfo(...) from ...`

Without using a widget the result is downloaded.



you@rasql>Status:

Executing Query. 60 miliseconds passed. Please wait...

you@rasql>Query: SELECT encode(c[*:*,*:*], "PNG") FROM mr AS c

Result:

```
image>>SELECT encode(c[*:*,*:*], "PNG") FROM mr AS c
```

Figure 11.16: Example of a 2D image result.