



## Distributed File Management

### Distributed File Management 2.0 *and* Adaptation of Use Cases and Testing<sup>1</sup>

Deliverables	D3.4 / D3.5
Authors	Ralf Wahner and Thomas Brüsemeister
Editors	Thomas Röblitz
Date	2008-08-04 11:40:32 +0200 (Mon, 04 Aug 2008)
Document Version	1.0.0
Current Version	1.0.0
Previous Versions	0.4.1, 0.4.0, 0.3.8-0, 0.2.0, 0.1.0

#### **A: Status of this Document**

Deliverables D4 and D5 of working group 3.

#### **B: Reference to project plan**

Fourth and fifth deliverable of working group *Distributed File Management*.

#### **C: Abstract**

---

<sup>1</sup>This work is part of the AstroGrid-D project and D-Grid. The project is funded by the German Federal Ministry of Education and Research (BMBF).

This deliverable documents the implementation of the AstroGrid-D file management system (D3.4) and the adaptation of the use cases to it as well as basic functionality tests (D3.5).

## D: Change History

Version	Date	Name	Brief summary
0.1.0	November 13 <sup>th</sup> , 2007	Mikael Höggvist, Thomas Brü- semeister, Ralf Wahner	Initial draft
0.2.0	May 13 <sup>th</sup> , 2008	Thomas Röblitz	Structure
0.3.0	May 14 <sup>th</sup> , 2008	Ralf Wahner	LaTeX-rootfile ( <i>wg3-d4.tex</i> ) revised. A separate <i>.tex</i> file is assigned to each <code>\section{}</code> . Several custom <code>\newcommand{}</code> s in <i>input/latex_commands.tex</i> .
0.3.1	May 14 <sup>th</sup> , 2008	Ralf Wahner	Section 3.1 <i>Command-line Interface</i> provided (needs revision).
0.3.2	May 20 <sup>th</sup> , 2008	Ralf Wahner	Subsections <i>Editing Files</i> and <i>Retrieve Files</i> added. Several typos fixed.
0.3.3	June 2 <sup>nd</sup> , 2008	Thomas Röblitz	Added Chapter 5 <i>Experiences in Using ADM</i> covering the content of Deliverable 3.5.
0.3.4	June 12 <sup>th</sup> , 2008	Ralf Wahner, Thomas Brüse- meister	Chapter 4 <i>Installation, Configuration and Administration of the ADM</i> written. Section 3.1 <i>Application Programming Interface</i> written.
0.3.5	June 15 <sup>th</sup> , 2008	Ralf Wahner, Thomas Brüse- meister	Chapter 2 <i>System Design</i> completely rewritten. Section 5.2 <i>Performance and Scalability Test Environment Setup</i> written.
0.3.6	June 16 <sup>th</sup> , 2008	Ralf Wahner	Subsection 4.2 <i>Server Installation</i> extended (registering file-spaces with ADM).
0.3.7	June 17 <sup>th</sup> , 2008	Thomas Röblitz	Section 5.1 on basic tests and recommendations for further improvements
0.3.8	June 18 <sup>th</sup> , 2008	Ralf Wahner	Subsection 2.4 <i>Virtual Filesystem and Replica Operations</i> completely rewritten. Introductory paragraph for the sourcecode example.
0.4.0	June 29 <sup>th</sup> , 2008	Ralf Wahner	Chapter 4 <i>Installation, Configuration and Administration of ADM</i> revised. Section 5.2 <i>Using the ADM for Managing NBody Data</i> still missing.
0.4.1	July 11 <sup>th</sup> , 2008	Ralf Wahner	Directory <i>3_4/misc</i> now includes <i>tree-gen.pl</i> , a Perl program for building a systematically configured testing environment for ADM; see Section 5.2 <i>Performance and Scalability Test Environment Setup</i> .
1.0.0	August 4 <sup>th</sup> , 2008	Ralf Wahner	Final Version.

---

## Contents

<b>A. Status of this Document</b>	<b>1</b>
<b>B. Reference to project plan</b>	<b>1</b>
<b>C. Abstract</b>	<b>2</b>
<b>D. Change History</b>	<b>2</b>
<b>1 Introduction</b>	<b>5</b>
<b>2 System Design</b>	<b>5</b>
2.1 Introduction and Architecture Outline . . . . .	5
2.2 Format of the HTTP Response Message Body . . . . .	6
2.3 LFID Reverse Lookup . . . . .	7
2.4 Virtual Filesystem and Replica Operations . . . . .	8
<b>3 ADM Interfaces</b>	<b>10</b>
3.1 Command-line Interface . . . . .	10
3.1.1 Introduction and Basic Usage . . . . .	10
3.1.2 Commit, Retrieve and Edit Files . . . . .	13
3.1.3 File-space Concept . . . . .	17
3.1.4 File Replication and Cleanup . . . . .	19
3.1.5 Outlook . . . . .	20
3.2 Application Programming Interface . . . . .	20
3.2.1 Sourcecode Example . . . . .	23
3.2.2 Compiling the Sourcecode . . . . .	24
<b>4 Installation, Configuration and Administration of ADM</b>	<b>24</b>
4.1 Client Installation and Configuration . . . . .	25
4.2 Service Installation and Configuration . . . . .	25
<b>5 Experiences in Using ADM</b>	<b>28</b>
5.1 Basic Functionality Tests . . . . .	28

---

5.2 Performance and Scalability Test Environment Setup . . . . .	29
<b>References</b>	<b>31</b>

## 1 Introduction

The first version of the *Distributed File Management* has been specified in Deliverable 3.2, [2], and tested in Deliverable 3.3, [3]. Testing the initial approach revealed certain disadvantages concerning working with as well as inherent properties of the Globus Replica Location Service (RLS). The second version of the *Distributed File Management*, namely the AstroGrid-D Data Management (ADM), presented in this deliverable, focuses on *user-friendliness* and *robustness* of the file management and storage capacity. The ADM, designed and written by Thomas Brüsemeister<sup>2</sup> in the late summer of 2007, provides a virtual filesystem and simultaneously cares for the placement of the corresponding physical files on one or more storage facilities. ADM is a tool for distributed data-management [1], providing the following features:

- An intuitive command-line interface (simpler and more consistent than RLS).
- Monitoring of the physical availability of a file linked to a symbolic name (dead link problem) and automatically accessing a file replica if the requested one is not available.
- Management of different storage servers (at the level of physical files).
- Automatic maintenance of a core set of logical file metadata (creation time, owner, etc.).
- Managing logical files in some hierarchical directory system (similar to a typical file system hierarchy).

The term "AstroGrid-D Data Management" is equivalent to the notion of the "virtual filesystem" and both are used interchangeably in this text. The subsequent second chapter describes the design underlying ADM. The third chapter presents the ADM command-line interface in a tutorial-like manner and a reference documentation for the application programming interface. The fourth chapter concerns the installation procedure, configuration and administration of the AstroGrid-D Data Management. The last chapter covers the content of Deliverable 3.5 *Adaptation of the Use Cases and Testing of Deliverable 3.4*.

## 2 System Design

The AstroGrid-D Data Management (ADM) implements a client-server approach. The client program `adm` is based on the ADM library, both written in C, while the server (or "service"; see below) is written in Java providing a straightforward access to PostgreSQL data bases. At the time of this writing the ADM service occupies a single host, only. Spreading the service across two or more hosts, i.e. migrating to a distributed service, is an issue for a future release. This chapter describes ADM along general lines with selected items presented in more detail.

### 2.1 Introduction and Architecture Outline

The protocol implemented by the ADM service is an application-level protocol for distributed filesystem and replica management. It is based on HTTP/HTTPS, a widespread and well known protocol

---

<sup>2</sup>E-mail: [tbruese@ari.uni-heidelberg.de](mailto:tbruese@ari.uni-heidelberg.de)

for distributed, collaborative, hypermedia information systems. Instead of an additional messaging layer like Simple Object Access Protocol (SOAP), the AstroGrid-D Data Management implements a kind of Representational State Transfer (REST) interface in order provide filesystem and replica management.

The ADM uses a relational database, namely PostgreSQL, to store a unique descriptor, i.e. a Logical File Identifier (LFID) for each file, as well as meaningful or typical meta data for each file or directory, e.g. the owner and a timestamp in order to log when the entry has been registered with the filesystem. Figure 1 on page 7 shows the database table layout. Whereas file ownership and creation timestamp are compulsory meta data and ADM transparently cares for their maintenance, individual files can be endowed with custom (user-defined) properties. ADM provides the command-line client `adm`, including a C-library, which offers an easy-to-use access to the stored files. Furthermore, ADM ships with a web interface which permits to browse the virtual filesystem graphically; see Figure 3 on page 12.

Figure 2 on page 8 shows a bird's eye view of the ADM architecture outline. Each host has an ADM client installed, which communicates in two different ways with its environment. ADM implements a *virtual* filesystem, i.e. something *pretending* to be a *real* filesystem like Ext-3, JFS or NTFS. Physical files are renamed with a 32 bit hash value generated by means of the Message Digest 5 (MD5) algorithm on the file content and stored on so-called file-spaces in a flat hierarchy. While this approach is advantageous from a technical point of view, because naming conflicts between files are virtually impossible and searching the stock of files includes only a few subdirectories, the cryptic filenames are unreadable for humans. This is, where the virtual filesystem comes into play. The virtual filesystem maps each cryptic filename onto a human readable filename assigned by a human user and stored in the aforementioned PostgreSQL data base. The two ways of communication are between client and virtual filesystem on the one hand and between client and the file-space on the other. ADM clients can talk directly to the file-spaces in order to read or write physical files, whereas they need to contact the ADM service first, in order to access the data base tables that constitute the virtual filesystem.

## 2.2 Format of the HTTP Response Message Body

Currently, three response formats are supported, namely `none`, `csv` and `html`. Common output formats like JavaScript Object Notation (JSON) and the omnipresent Extensible Markup Language (XML) are scheduled. The content type of an HTTP response message body can be controlled by declaring the preferred Multipurpose Internet Mail Extensions (MIME) type in the header of the request transmitted to the ADM service. To get the response message body in comma-separated value (CSV) or HTML format the request header field `Accept` should be set to `text/csv` or `text/html`, respectively. Setting the response format to `none` causes the ADM service to sent back the HTTP status code, only, whereas the response message body is empty, which obviously represents an economical way to try if a resource is valid and accessible prior to downloading it. The header field `Accept` can be overridden by adding the query suffix `?format=<format>` to the URL. This is useful when trying to download the data in non-HTML format using a webbrowser.

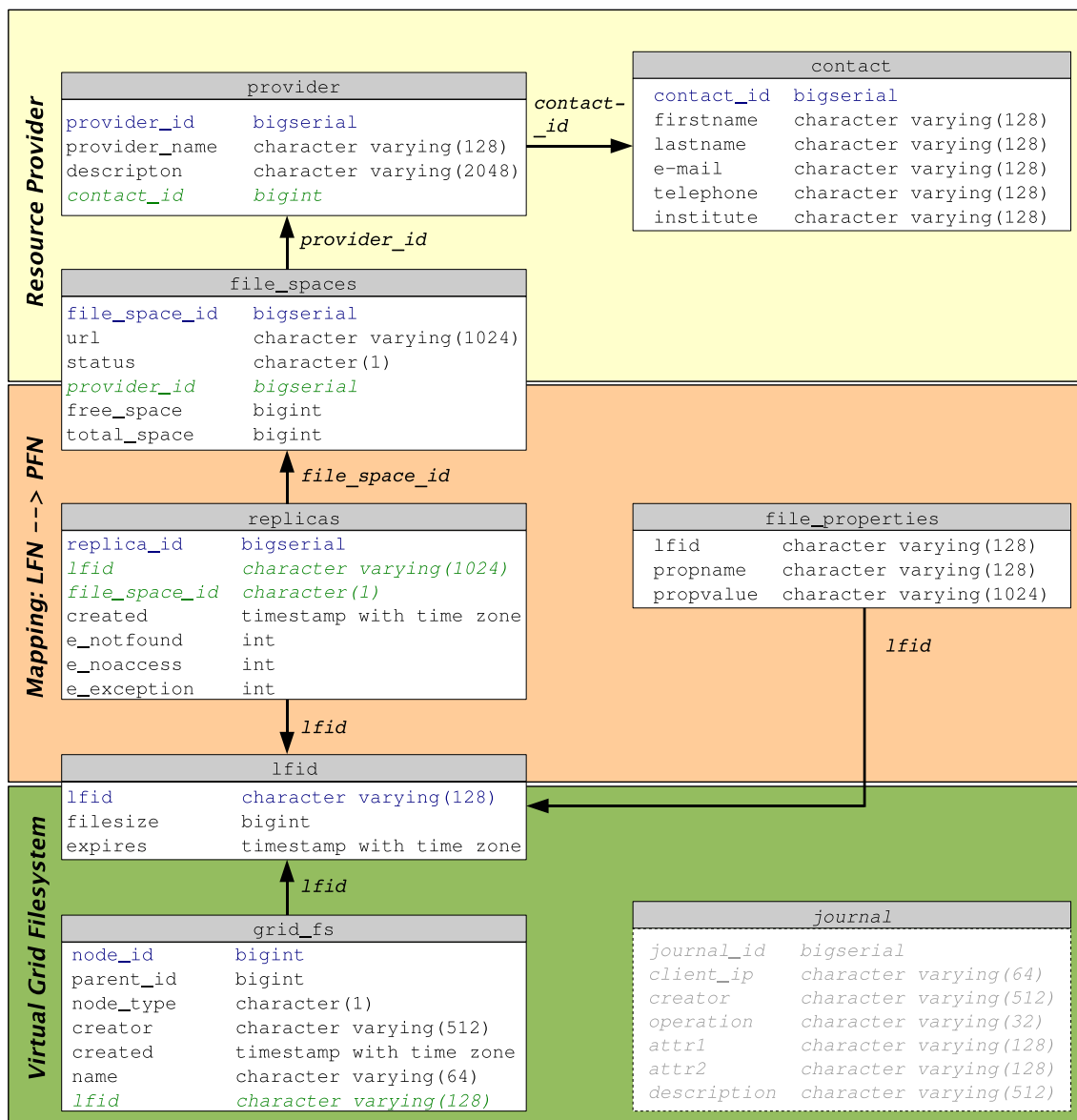


Figure 1: The current ADM database table layout (PostgreSQL) consists of seven tables. Primary key fields are set in blue and foreign key fields in green letters. The journal table in the lower right corner will allow to log the operations processed on the file system (the table is not yet used in the current prototype).

### 2.3 LFID Reverse Lookup

The fully-qualified physical filenames (PFNs) behind a given Logical File Identifier (LFID) can be resolved by means of /adm/lfid/<lfid>, which internally relies on the HTTP-method GET. Here, "fully-qualified" refers not only to the path with respect to the root of the filesystem where the physical file resides, but also to the employed host name, port number and transmission protocol, i.e. HTTP, FTP or GSIFTP to name just a few. If replica exist, an LFID usually points to an "array" of PFNs; see Section *Understanding logical and physical filenames and mappings* in Deliverable 3.3.

The PFNs are delivered to the requester wrapped up in the response message body. Otherwise, the ADM service returns an HTML page containing the HTTP status code 404 (Not found).

## 2.4 Virtual Filesystem and Replica Operations

Virtual Filesystem and Replica Operations are common HTTP requests against the ADM service by means of the HTTP-method POST. The request message body has the following format

```
<OPERATION> ADM/<PROTOCOL-VERSION>
<ATTRIBUTE-1> <ATTRIBUTE-VALUE-1>
<ATTRIBUTE-2> <ATTRIBUTE-VALUE-2>
      :
      :
<ATTRIBUTE-n> <ATTRIBUTE-VALUE-n>
```

<OPERATION> represents an intervention in the virtual filesystem and is one of: ADDFILE, which registers a new file, RMFILE, which removes the file, all replicas and file properties, LINK, which creates a reference to an existing file, ADDREP, which creates a replica of one file, RMREP, which removes a replica, MKDIR, which creates a directory, RMDIR, which removes a directory, MOVE, which moves a file or directory to a different location or renames the filesystem entry, PROPSET, which endows a property to a file or PROPDEL, which removes a property from a file. Currently, the

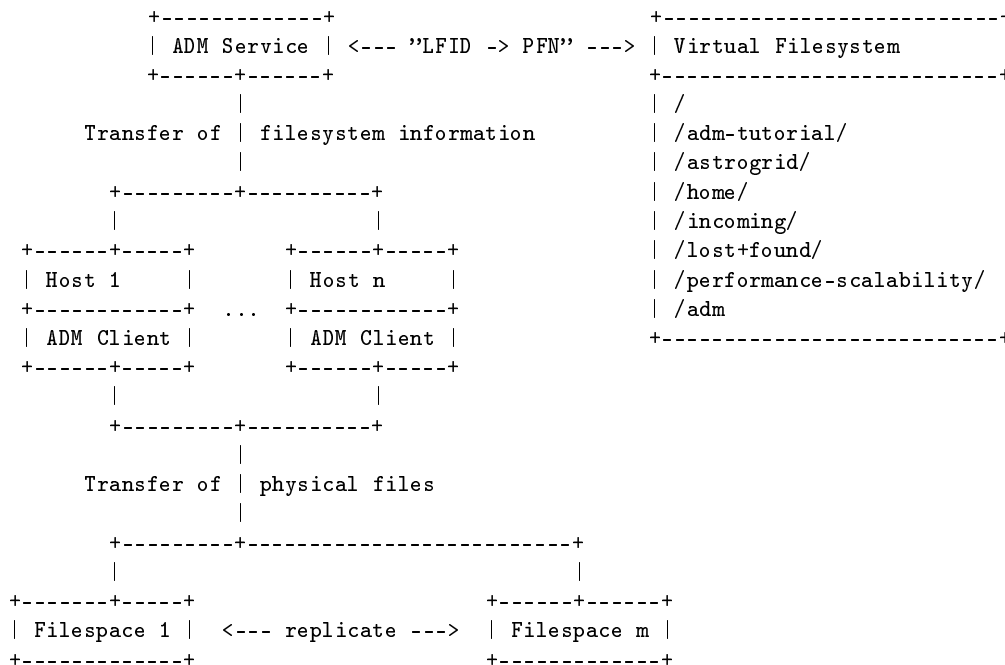


Figure 2: Architecture outline of the AstroGrid-D Data Management. One instance of the ADM service handles requests sent by many clients and talks to the data base representing the virtual filesystem. Transmission of physical files occurs directly between client and file-space. Note the distinction between physical files and meta data in the lower/upper part of the figure, respectively.



ADM service accepts one <PROTOCOL VERSION>, only, namely version 0.9. Each operation has an individual number of Parameters represented as key-value-pairs, where key and value are separated by whitespace and two adjacent pairs are separated by single line breaks. For example, the following request instructs the ADM service to create the directory */new-directory* directly below the root of the virtual filesystem; note the leading slash (/):

```
MKDIR ADM/0.9
PATH /new-directory
USERDN /O=GermanGrid/OU=ZAH/CN=Ralf Wahner
```

ADM performs common filesystem operations like file and directory creation, renaming and moving as well as deletion, to name just a few. The functional range of the filesystem operations provided by ADM is declared in the special file *PROTOCOL*, which belongs to each copy of the ADM service distribution below its "root directory" *admservice*. Beyond the mere "keywords" insertable for <OPERATION> *PROTOCOL* specifies valid parameter names and values as well as the messages, the ADM service returns depending on the outcome of the individual operation. The following sourcecode-like formatted paragraph shows the specification for the command behind `adm add` and `adm mkdir`, respectively:


```
Operation: ADDFILE
Registers a new file in the vfs
Required Attributes:
  LFID    MD5 checksum, 32 hex characters
  PATH    /path/in/vfs
  FSPACE  The file-space ID
  URL     The URL to the file-space
  SIZE    Filesize in bytes
  USERDN  The distinguished name of the user.

HTTP status codes:
200 OK
201 LFID already exists, created link, no transfer required
400 ERROR, message is delivered in HTTP-body
```

```
Operation: MKDIR
Creates a directory in the VFS
Required Attributes:
  PATH /path/in/vfs
  USERDN The distinguished name of the user.

HTTP status codes:
200 OK
400 ERROR, message is delivered in HTTP-body
404 Directory already exists
```

Apart from the ADM client program `adm`, requests for filesystem operations can be issued by any program or tool capable of talking to a common HTTP server. Assume, that the above request to create */new-directory* has been written to the file *request.mkdir*. Then the request can be sent to the ADM service by means of the plain old `wget` command-line tool:

```
agrid064@alnitak:~$ wget --post-file=request.mkdir http://alnitak:12000/adm/ 
--13:46:09-- http://alnitak:12000/adm/
```

```
=> 'index.html.3'  
Resolving alnitak... 129.206.110.246  
Connecting to alnitak|129.206.110.246|:12000... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 58  
100%[=====>] 58          --.--K/s  
13:46:09 (3.07 MB/s) - 'index.html.3' saved [58/58]
```

## 3 ADM Interfaces


The AstroGrid-D Data Management (ADM) has a command-line interface for interactive manual handling of a few files and directories as well as an application programming interface (API) in order to communicate with the virtual filesystem directly from inside scientific code. Currently, the API is restricted to C sourcecode, but bindings for Perl and Java are scheduled for the near future.

### 3.1 Command-line Interface


The command-line interface is meant for manual or scripting access to the virtual filesystem and is equivalent to the application programming interface with respect to its functional range. First off, this text concentrates on the command-line interface `adm` in a tutorial-like manner and defers the discussion including the library to the second subsection.

#### 3.1.1 Introduction and Basic Usage

ADM provides a set of commands that allow to add, delete or move/rename files and directories. In the style of the Subversion syntax, an ADM *command* is composed of two "words" and one or more arguments:

```
# Basic adm command structure  
adm <subcommand> argument(s) 
```

The first word is always `adm`, the ADM client program, or tersely speaking, the *client*. The second word is the actual instruction, alias *subcommand*, according to the client messages, that indicates the operation to be carried out on the virtual filesystem, e.g. `list` in order to display the contents of a directory. Usually, an `adm` subcommand has one or two arguments, except for `adm help` which has either none or a single argument as well as the always no-argument command `adm info`. First off, if invoked without arguments `adm help` displays a list of all `adm` commands available: `adm help` (no arguments)

```
# Display available adm commands  
agrid064@alnitak:~$ adm help   
  
adm command-line client, version 0.2.0  
  compiled on Mar 10 2008, 08:55:58  
  
adm <subcommand> [options] [args]
```

Type 'adm help <subcommand>' for help on a specific subcommand.

Available subcommands:

```
add (put)
edit (ed)
find
get
info
link (copy, cp, ln)
list (ls)
mkdir
move (mv, ren, rename)
propdel (pdel, pd)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
remove (rm, del, delete)
replicate (rep)
resolve (res)
rmdir
```


ADM is a tool for distributed data-management.

Copyright (c) 2007-2008 Thomas Bruesemeister, ZAH.

*Remark:* adm help can have an optional argument, namely a subcommand, in which case it shows a more detailed documentation for subcommand; see below. When the requested subcommand is misspelled or does not exist, adm help falls back to the above no-argument output.

At the time of this writing (06/2008), there is just one subcommand left, that has not yet been included in the current version of the client, namely adm locate, to *quickly* look up files and directories in the virtual filesystem. For the time being, the present client provides the slower but more reliable find subcommand. The behavior of adm find and adm locate generally reflects the corresponding Unix or Linux commands, where locate relies on a regularly refreshed database, while find examines the given subtree of the filesystem in its current state each time it is invoked. Since locate can "see" only what the filesystem contained when its database was recently refreshed, newly created files and directories are invisible to locate until the next database update happens. On the other hand, locate is quite fast, because it merely queries its database instead of browsing the whole filesystem. Since find analyzes a complete subtree of the filesystem entry-by-entry, i.e. find is recursive by default, it unearths all files matching the search criterion right at the moment of its invocation and therefore usually consumes more time.

According to the above output most ADM subcommands provide an abbreviated two-character-version, e.g. adm ls vfs-path abbreviates adm list vfs-path, where "vfs" is short hand for "virtual filesystem". The list (ls) subcommand displays the content of the directory specified by vfs-path:

```
# Show root directory
agrid064@alnitak:~$ adm ls / 
adm-tutorial/
astrogrid/
home/
incoming/
```

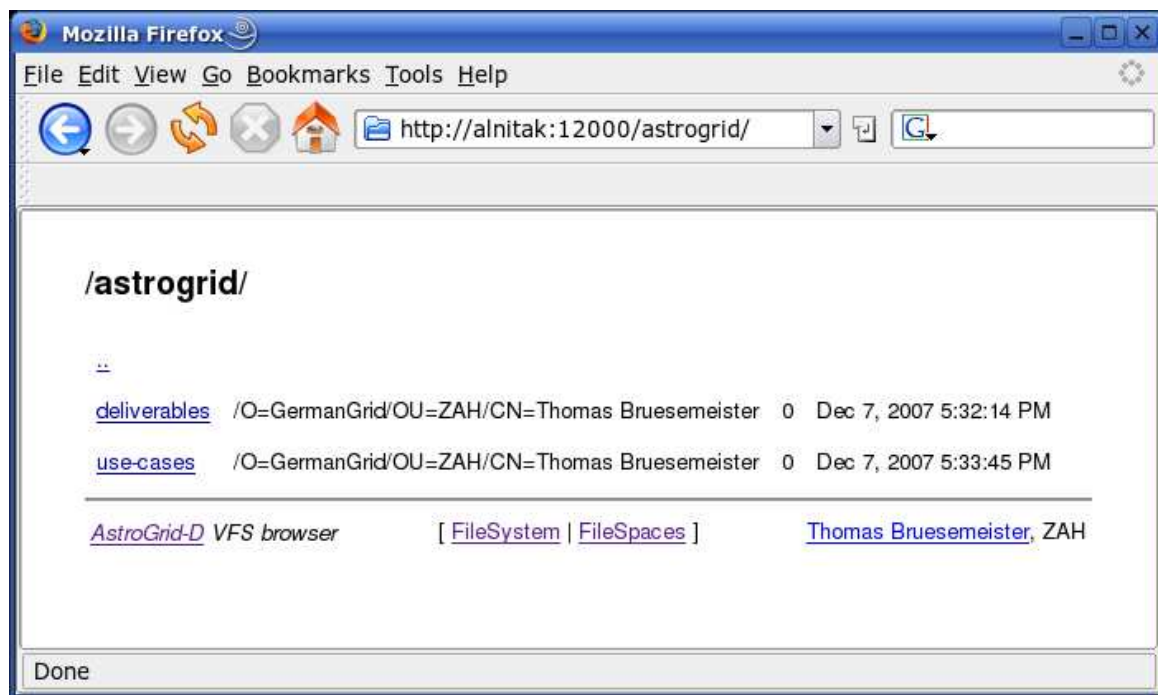


Figure 3: ADM-Webinterface on `http://alnitak.ari.uni-heidelberg.de:12000`.

```
lost+found/
adm
```

Even though it means more typing, the code examples in this text use the more distinct non-abbreviated notation. Table 1 on page 15 shows all subcommands at hand with the current client together with the short hand syntax, if available. Note, that following Unix and Linux habits, the root directory of the virtual filesystem is `/`. Moreover, all ADM commands need absolute paths, except for `adm info` which is the only no-argument command. Like its shell counterpart the `ls` command has an `-l` command-line flag in order to show more elaborate information compared to the sole file- or directory name:

```

# Show root directory (verbose output)
agrid064@alnitak:~$ adm list -l / ↵
d nGrid/OU=ZAH/CN=Ralf Wahner      0 2008-01-08 10:25 adm-tutorial/
d ZAH/CN=Thomas Bruesemeister     0 2007-12-07 17:32 astrogrid/
d ZAH/CN=Thomas Bruesemeister     0 2007-12-07 17:31 home/
d ZAH/CN=Thomas Bruesemeister     0 2007-12-07 17:32 incoming/
d ZAH/CN=Thomas Bruesemeister     0 2008-03-05 09:51 lost+found/
d nGrid/OU=ZAH/CN=Ralf Wahner     0 2008-01-08 13:14 performance-scalability/
s ADM                             0 2007-12-07 17:18 adm
7 entries

```

The leftmost column indicates the file type of the entries, where lowercase `d` is assigned to directories whereas lowercase `f` denotes a file. `ADM` is a particular directory internally used by ADM for administrative purposes and therefore has type `s` in order to be distinguishable from normal directories. The second column displays the file owner, compiled from the attribute mnemonics found

in the users proxy certificate and truncated for the sake of readability. The two-character keywords are defined in the Lightweight Directory Access Protocol (LDAP) specification and mean: common name (CN), organization (O), organizational unit (OU) and country (C). The third column shows the file size in bytes and intentionally vanishes for directories. Finally, the fourth column indicates date and time when the entries were created.

Due to its presetting, `adm list` displays all entries in the specified directory, regardless of the ownership. The `-u` flag suppresses all files and directories other than those owned by the user invoking the `adm ls` command:

```
# Show root directory (verbose output)
agrid064@alnitak:~$ adm list -l / ↩
d nGrid/OU=ZAH/CN=Ralf Wahner 0 2008-01-08 10:25 adm-tutorial/
d nGrid/OU=ZAH/CN=Ralf Wahner 0 2008-01-08 13:14 performance-scalability/
7 entries (5 filtered)
```

The output of `adm help list` demonstrates how to access the built-in documentation for an ADM command and summarizes the previous two examples:

```
# Show build-in help for "list"
agrid064@alnitak:~$ adm help list ↩
Lists files and directories in the virtual filesystem.
usage: list vfs-path
Valid Options:
  -l [--long]           : use a long listing format
  -u [--userdn-matches] : shows only entries matching your userdn
Example: adm ls -l /home
```

By the way, as the above output shows, each `adm help <subcommand>` contains an example how to use this subcommand.

### 3.1.2 Commit, Retrieve and Edit Files

At the beginning of the brief round tour about file management with ADM, a new directory `/adm-tutorial/vfs_tour` is created by means of `adm mkdir`:

```
# How to create a new directory
agrid064@alnitak:~$ adm mkdir /adm-tutorial/vfs_tour ↩
```

According to Unix or Linux habits, a successful ADM command normally does not display a message. Again, `adm list` verifies, that the new directory now exists:

```
# Show directory /adm-tutorial (verbose output) ↩
agrid064@alnitak:~$ adm list -l /adm-tutorial
d nGrid/OU=ZAH/CN=Ralf Wahner 0 2008-01-08 10:16:42 vfs_tour/
1 entries
```

All operations in this text are supposed to take place in the above */adm-tutorial* directory; see above output of `adm list -l /` on page 11. A valid file or directory name, with respect to the virtual filesystem implemented by the AstroGrid-D Data Management, may contain uppercase and lowercase latin letters (a, ..., z, A, ..., Z), underscores (`_`), plus (`+`) and minus (`-`) signs as well as dots (`.`), in other words any string matching the regular expression `^[\\w\\-\\+\\.]+$/`.

**Commit Files.** Files are committed to ADM by means of `adm add`, which is equivalent to `adm put`. Assume that the current working directory contains the file *my\_jobdescription.jsdl*. This file is then delegated to ADM by

```
# How to register a file with ADM
```

```
agrid064@alnitak:~$ adm add -v my_jobdescription.rsl /adm-tutorial ↩
```

```
Source: file:///home/Aguid/agrid064/
```

```
Dest: gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/fs01/  
my_jobdescription.rsl -> aab3c89633c6af44407ecede98f4fb5  
1743 bytes 0.01 MB/sec avg 0.01 MB/sec inst
```

Usually, `adm add` operates quiet. The above detailed output is due to providing the `-v` flag with the command invocation. The target location with respect to the virtual filesystem can be a directory path without trailing file name or a fully qualified file name. Note, that the client accepts absolute paths, only, without exception, i.e. for all subcommands. If invoked without trailing file name, `adm add` implicitly appends the basename of the physical file to the path with respect to the virtual filesystem, whereas in the latter case, the file can in one step be put under ADM control and renamed. The cryptic string `aab3c89633c6af44407ecede98f4fb5` is generated by applying the "Message-Digest Algorithm 5" on the file content, i.e. `md5sum my_jobdescription.rsl`, and represents the name ADM uses internally to unequivocally identify *my\_jobdescription.jsdl* among the other files registered with ADM.

By default, `add` behaves "non-recursive", i.e. it handles just one file and no directories at a time. In order to enable handling of whole filesystem subtrees, `add` and several other subcommands (see Table 2 on 16) own the `-r` flag. The following command was used to register the L<sup>A</sup>T<sub>E</sub>X source of this tutorial text with ADM:

```
# How to register a filesystem subtree with ADM (ignore invisible entries)
```

```
agrid064@alnitak:~$ adm add -r adm-tutorial /adm-tutorial/latex-source ↩
```

Note, that the */adm-tutorial/latex-source* directory is supposed to exist before invoking the above command; `adm` will complain otherwise. If invoked with the `-r` flag, according to its presetting, `add` ignores files and directories with leading dot (`.`) in their name, i.e. "invisible" files and directories, e.g. *.bashrc* or the *.svn* directories when the sourcecode underlies version control via Subversion. This default behavior can be overridden by means of additionally providing the `-a` flag, which simply tells the client to consider the dotted filesystem entries as well:

```
# How to register a filesystem subtree with ADM (include invisible entries)
```

```
agrid064@alnitak:~$ adm add -r -a adm-tutorial /adm-tutorial/latex-source ↩
```

The `-p` flag for `add` allows to specify the number of so-called "parallel streams" to use for the file transfer and is directly handed over to the corresponding flag of the underlying `globus-url-copy`

adm-Subcommand	Options	Description
add	-a, -b, -p, -r, -s, -v	Register a file with ADM. See mkdir for directories.
copy (cp)	no options	Create a new link to an already registered file. (Files only.)
delete (del)	-r, -v	Unregister a file from ADM. See rmdir for directories.
edit (ed)	no options	Allows in-situ editing on a registered file without download.
find	no options	Search for files and directories matching a pattern.
get	-b, -p, -r, -s, -v	Download a file or directory registered with ADM.
info	no options	Print status and properties of the client (adm).
link (ln)	no options	See copy (cp)
list (ls)	-l, -u	Print the contents of a directory. (Directories only.)
<del>locate</del>	<del>////////////////////</del>	<del>Not yet implemented, see page 11</del>
mkdir	no options	Register a new directory with ADM.
move (mv)	no options	Change the location or name of a file or directory.
propdel (pdel, pd)	no options	Unregister a property from a file registered with ADM.
propget (pget, pg)	no options	Retrieve a property value.
proplist (plist, pl)	no options	Show the properties registered for a file.
propset (pset, ps)	no options	Register a property, i.e. a name-value pair, for a file registered with ADM. (Files only)
put	-a, -b, -p, -r, -s, -v	See add
remove (rm)	-r, -v	See delete (del)
rename (ren)	no options	See move (mv)
replicate (rep)	-b, -s	Creates a replica for a file registered with ADM. (Files only.)
resolve (res)	-a, -f, -l	Prints all replica available for a given file registered with ADM.
rmdir	no options	Unregister an <i>empty</i> directory from ADM.

Table 1: Overview of the ADM-subcommands. The left column shows all subcommands available in the current release. The column in the middle summarizes the flags available for each subcommand. Since many flags are valid for two or more subcommands the description of the flags has been separated from this table; see Table 2 on page 16. The right column briefly tells about the purpose of the individual subcommands.

command. It is recommended to keep the presetting of four streams unchanged, i.e. to not use -p, because experience has proven that four streams care for the best transfer capacity across the internet.<sup>3</sup> The add subcommand owns two more flags, namely -s and -b. Understanding these flags which also appear with several other subcommands (again, see Table 2 on page Table 16), requires understanding the notion of the file-space. Therefore, introducing -s and -b is deferred to the next subsection *File-space Concept*.

<sup>3</sup>See section *Performance Options, "How do I pick a value?"* under [5] for a short discussion on data transmission with parallel streams and how to choose the number of connections.

Option	Occurrence	Description
-a [--all]	add, put	Include files and directories when their names has a leading dot (invisible files/directories).
-a [--all]	resolve	Show also replicas on inactive file-spaces.
-b [--fallback]	add, get, put, replicate	Try to access an alternative file-space if available and give up otherwise.
-f [--file]	resolve	Name of the file where all occurrences of adm:// are supposed to be substituted by physical file names.
-p [--parallel-streams]	add, get, put	Specify how many parallel streams to use for the data transfer. Default is 4 streams.
-r [--recursive]	add, delete, get, put, remove	Apply command to the subtree of the filesystem given by the command argument.
-s [--file-space]	add, get, put	Access the specified file-space only and give up immediately if the file-space is unavailable.
-v [--verbose]	add, delete, get, put, remove	Show verbose output for the command at hand.
-l [--long]	list	Show verbose information about files and directories, e.g. file owner and file size.
-u [--userdn-matches]	list	Show only files and directories owned by the user who invoked the list command.

Table 2: Options of the ADM-subcommands. Each one-character option has a corresponding long version. Except for -a, which has different meanings for add (put) and resolve, the meaning of the options is consistent for all subcommands; see Table 1 on page 15.

**Move and Rename Files.** Files and directories in the virtual filesystem can be moved from one place to another by means of `adm move`, abbreviated by `adm mv`. This command has always two arguments, namely the entry to be moved and the target file or directory. Given, that `/adm-tutorial/vfs_tour` has a subdirectory `jsdl` the following command will change the location of `my_jobdescription.jsdl` from `/adm-tutorial/vfs_tour` to the new subdirectory:

```
# Move a file to a different directory
agrid064@alnitak:~$ adm move /adm-tutorial/vfs_tour/my_jobdescription.rsl \ (↔)
                        /adm-tutorial/vfs_tour/jsdl

# Verify that the file has successfully been relocated
agrid064@alnitak:~$ adm ls -l /adm-tutorial/vfs_tour/jsdl (↔)
f nGrid/OU=ZAH/CN=Ralf Wahner 1743 2008-01-08 23:49:08 my_jobdescription.rsl
1 entries
```

The content of the new directory is listed to immediately confirm, that the move operation has occurred. The same command is used to change file and directory names within the virtual filesystem. Again, `adm help` denotes, that `adm move`, abbreviated by `adm mv`, is the same as `adm rename`, abbreviated by `adm ren`.

**Retrieve Files.** Files and directories registered with ADM can be downloaded from the virtual filesystem by means of `adm get`:



```
# How to retrieve a single file registered with ADM
agrid064@alnitak:~$ adm get -v /adm-tutorial/adm-tutorial.pdf ↩
Source: gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/fs01/
Dest:   file:///home/Tux/rwagner/
        606423e0e5c49b093de51e677dca0b44 -> adm-tutorial.pdf
```

Also `adm get` is "non-recursive" by default but the `-r` switch enables the command to retrieve even whole directories:

```
# How to recursively retrieve a directory registered with ADM
agrid064@alnitak:~$ adm get -r /adm-tutorial/latex-source/ ↩
```

This command downloads the aforementioned  $\text{\LaTeX}$ -sourcecode directory containing the ADM-Tutorial.

**Edit Files.** The current subsection finishes with a quite smart feature of ADM, namely in-situ editing a file under ADM control without prior downloading and subsequent uploading the file again from/to the virtual filesystem. The following command opens the file `sec_03.tex` and displays its contents readable and writable in the editor specified by means of the users `$EDITOR` environment variable:

```
# How to edit (in-situ) a file registered with ADM
agrid064@alnitak:~$ adm edit /adm-tutorial/latex-src/input/sec_03.tex ↩
```

Figure 4 on page 18 shows the effect of the above command-line. Internally, ADM uses a file copy with a special name in order to keep the former version save until the editor is appropriately closed and the file is written back to the virtual filesystem. Editing on-site the storage location is handy for minor quick modifications where retrieving the file locally, editing and later shoveling the file back to ADM would be disproportionate.

### 3.1.3 File-space Concept

Where does a *physical* file reside, after it has been registered with AstroGrid-D Data Management? ADM subcommands that actually transfer files in either direction between grid accounts and storage facilities, i.e. the aforementioned `adm (add|put)` and the below described `adm replicate` and `adm get` have two additional flags, namely `-s` and `-b`, which allow to specify a so-called *file-space*. From the users perspective, a file-space is a large amount of disk space with a unique identifier provided by a member of the AstroGrid-D community, that can be accessed to store scientific data. Each client can individually select a *default file-space*. The client talks to a file-space by means of its Uniform Resource Locator (URL).

Currently ADM owns three file-spaces, one at the "Center for Astronomy of Heidelberg" (3.64 Terabytes) and two at the "Astrophysical Institute Potsdam" (2×1.73 Terabytes), as the always no-argument command `adm info` certifies:

```
# Show status information about ADM (including all file-spaces available)
agrid064@alnitak:~$ adm info ↩
```

```

emacs@alnitak
\sourcecomment{# Show root directory (verbose output)} \
\srcindent{00}\shellprompt{} \sourcecodehighlight{adm ls -l /} [6pt]
\srcindent{00}d nGrid/OU=ZAH/CN=Ralf Wahner \srcindent{3}0 2008-01-08 10:25 adm-tutorial/\
\srcindent{00}d ZAH/CN=Thomas Bruesemeister \srcindent{3}0 2007-12-07 17:32 astrogrid/\
\srcindent{00}d ZAH/CN=Thomas Bruesemeister \srcindent{3}0 2007-12-07 17:31 home/\
\srcindent{00}d ZAH/CN=Thomas Bruesemeister \srcindent{3}0 2007-12-07 17:32 incoming/\
\srcindent{00}d ZAH/CN=Thomas Bruesemeister \srcindent{3}0 2008-03-05 09:51 lost+found/\
\srcindent{00}d nGrid/OU=ZAH/CN=Ralf Wahner \srcindent{3}0 2008-01-08 13:14 performance-scalability/\
\srcindent{00}s \srcindent{24}ADM \srcindent{3}0 2007-12-07 17:18 adm\
\srcindent{00}7 entries
\end{sourcecodeENV}
***
The leftmost column indicates the file type of the entries, where lowercase \sourcecode{d} is assigned to directories
whereas lowercase \sourcecode{f} denotes a file. \filename{ADM} is a particular *** ausgezeichnet (besonders)
directory internally used by \ADM{} for administrative purposes and therefore has type \sourcecode{s} in order to
be distinguishable from normal directories. The second column displays the file owner, compiled from the attribute
mnemonics found in the users proxy certificate and truncated for the sake of readability. The two-character keywords
are defined in the Lightweight Directory Access Protocol (LDAP) specification and mean: common name (\sourcecode{CN}),
organization (\sourcecode{O}), organizational unit (\sourcecode{OU}) and country (\sourcecode{C}). The third column
shows the file size in bytes and intentionally vanishes for directories. Finally, the fourth column indicates date and
time when the entries were created. The output of \sourcecode{adm help list} demonstrates how to access the built-in
documentation for an \ADM{} command and summarizes the previous two examples:
*** adm help list
\begin{sourcecodeENV}
\sourcecomment{# Show build-in help for 'list'} \
\srcindent{00}\shellprompt{} \sourcecodehighlight{adm help list} [6pt]
\srcindent{00}Lists files and directories in the virtual filesystem. [6pt]
\srcindent{00}usage: list vfs-path [6pt]
\srcindent{00}Valid Options: \
\srcindent{02}-l [--long] \srcindent{11}: use a long listing format \
\srcindent{02}-u [--userdn-matches] \srcindent{1}: shows only entries matching your userdn [6pt]
\srcindent{00}Example: adm ls -l /home
\end{sourcecodeENV}
***
By default, \sourcecode{adm list vfs-path} displays all entries in the \filename{vfs-path} directory, where
'vfs' ist short hand for 'virtual filesystem', regardless of their individual ownership.
The optional \sourcecode{-u} (\sourcecode{-{}-user\dn-mat\ches}) switch allows to filter the
output in order to show only those files and directories that belong to the grid user who invoked the command.
*The \sourcecode{-b} switch is boolean, i.e. it is either present or absent and never has an argument.
By the way, as the above output shows, each \sourcecode{adm help <subcommand>} contains an example how
to use this \sourcecode{subcommand}.
-1:-- adm eu/prk5 (LaTeX)--L135--C21--32%-----

```

Figure 4: In-situ editing a file registered with the virtual filesystem. adm edit allows quick modifications on files without prior manual download and upload afterwards.

```

ADM service information, URL: http://alnitak.ari.uni-heidelberg.de:12000
      Version: 0.2.0-dev, $Revision: 278 $
      Protocol: ADM/0.9
      Service uptime: 26 days 21:18:52
      File-spaces: 3 [3 up 0 down]
      LFIDs: 2122
      Directories: 82
      Replicas: 2141
      MRU cache (size/hits/misses): 256/6834/2224
      Path lookback (hits/misses): 2178/45

User-DN:
  /O=GermanGrid/OU=ZAH/CN=Ralf Wahner

File-spaces:
  ID S URL                                     FREE          TOTAL
  1 a gsiftp://alnitak.ari.uni-heidelberg.de/... 3990339803400 4000000000000
  2 a gsiftp://astrodata10.gac-grid.org/...    1899768040842 1900000000000
  3 a gsiftp://astrodata07.gac-grid.org/...    1899999843231 1900000000000

Default file-space: 1

```

The URLs at the bottom end of the output are abbreviated for better readability. They are, to their full extend `gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/fs01`, `gsiftp://astroda-`

`ta10.gac-grid.org/store/05/zah` and `gsiftp://astrodata07.gac-grid.org/store/02/ADM`.

The command-line client uses the default file-space to place new files or to retrieve files that are already under ADM control, unless told otherwise or the default file-space is not available. Beyond unavailability of file-spaces, which can be caused e.g. by network failure or local administrative issues, there are reasons for overriding the default setting and manually selecting another file-space, e.g. duplicating crucial data for backup or shorter transfer distances across the internet. This is where the flags `-s` and `-b` come into play. If `-s` (`--file-space`) is present but `-b` is not, the client tries to access exactly the file-space given as the flag's argument. When the specified file-space is inaccessible, the client immediately gives up and displays an error message. However, if the `-b` (`--fallback`) flag is also present, the client will try one file-space after the other in order to access the desired file and it won't give up until the last file-space fails as well.

### 3.1.4 File Replication and Cleanup

A replica of a file is a one-to-one copy of that file on a different file-space. Two replicas of a file can never reside on the same file-space. Replicas are created for several reasons, e.g. to back up significant data or to reduce the network transfer load by locating a file as near to the desired computing resource as possible, to mention just two frequently named requirements. Unless told otherwise, ADM implicitly selects an appropriate file-space, when `adm replicate` is called:

```
# How to create replicas of files
```

```
agrid064@alnitak:~$ adm replicate /tutorial/vfs_tour/my_jobdescription.rsl ↔
Source: gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/fs01/
Dest:   gsiftp://astrodata10.gac-grid.org/store/05/zah/
       aab3c89633c6af44407ecedeb98f4fb5
```

While `adm list` displays the files and directories on the specified level in the filesystem hierarchy, `adm resolve` takes a filename argument and displays a list of locations of all replicas of that file, so `adm resolve` is a kind of counterpart of `adm list`:

```
# How to view available replicas and their locations
```

```
agrid064@alnitak:~$ adm resolve /tutorial/vfs_tour/my_jobdescription.rsl ↔
gsiftp://astrodata10.gac-grid.org/store/05/zah/aab3c89633c6af44407ecedeb98f4fb5
gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/fs01/aab3c89633c6af44407e-
cedeb98f4fb5
```

After browsing the virtual filesystem in order to find a specific file, the retrieval starts operating by means of `adm get` from the default file-space:

```
# How to get a file out of the ADM
```

```
agrid064@alnitak:~$ adm get /adm-tutorial/vfs_tour/my_jobdescription.rsl
Source: gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/fs01/ ↔
Dest:   file:///home/Aggrid/agrid064/
       aab3c89633c6af44407ecedeb98f4fb5 -> my_jobdescription.rsl
```

The commands `adm replicate` and `adm get` can provide the flags `-s` and `-b`, introduced in the previous subsection *File-space Concept*, in order to manually select the file-space where the replica should be placed or where the file should be retrieved from, respectively.

Finally, files are deleted from the virtual filesystem by means of `adm remove` (abbreviated `adm rm`) whereas directories are wiped out by means of `adm rmdir`. Non-empty directories cannot be removed and file and directory removal is based on ownership, i.e. any user can delete filesystem entries that he owns, only:

```
# How to remove a file from the ADM
```

```
agrid064@alnitak:~$ adm remove /vfs_tour/jsdl/my_jobdescription.rsl ↩
agrid064@alnitak:~$ adm rmdir /vfs_tour/jsdl/ ↩
agrid064@alnitak:~$ adm rmdir /vfs_tour ↩
```

### 3.1.5 Outlook

Careful readers might have noticed, that the commands concerning the user-defined file properties, namely `adm prop(del|get|list|set)` are not yet described in this tutorial. In order to accommodate the way of thinking and the requirements of common scientific AstroGrid-D users, a revised future version is supposed to guide readers along a typical scenario, i.e. an  $n$ -body or  $\varphi$ -grape numerical simulation, rather than "*my\_file.txt*" and "*my\_directory*". Apart from those two items, the authors welcome suggestions what should be included in this text; e-mail addresses are [rwahner@ari.uni-heidelberg.de](mailto:rwahner@ari.uni-heidelberg.de) (tutorial, this deliverable) and [tbruese@ari.uni-heidelberg.de](mailto:tbruese@ari.uni-heidelberg.de) (development). Please consider, that ADM has been released in Spring 2008 and development and documentation need some time to accumulate the users' experience. Thanks for you interest in the AstroGrid-D Data Management.

## 3.2 Application Programming Interface<sup>4</sup>

The ADM application programming interface allows to access the virtual filesystem directly from inside any program written in C. The ADM API is equivalent to the command-line client `adm` with respect to functional range. Usage of the ADM API is declared by means of `#include <adm>`. The ADM API contains the following functions:

`adm_addfile()` registers a file in the virtual filesystem and store it on a file-space.

```
int adm_addfile(adm_handle *handle, char *local_path, char *vfs_path)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `local_path`: The absolute or relative path to a local file which should be stored in ADM. `vfs_path`: The absolute destination path in the ADM virtual filesystem.

*Return value:* `adm_addfile()` returns 0 on success or one of the following error codes: `ADM_ERROR`, `ADM_ESERVICE`, `ADM_EINTERRUPT`.

`adm_addrp()` creates a replica of a file by copying the file to another file-space.

```
int adm_addrp(adm_handle *handle, char *vfs_path)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `vfs_path`: The absolute destination path in the ADM virtual filesystem.

---

<sup>4</sup>Contributed by: Thomas Brüsemeister ([tbruese@ari.uni-heidelberg.de](mailto:tbruese@ari.uni-heidelberg.de))

*Return value:* adm\_addrp() returns 0 on success or one of the following error codes: ADM\_ERROR, ADM\_ESERVICE, ADM\_EINTERRUPT.

adm\_finalize() frees the resources used by the ADM library. After this function has been called the handle is no longer valid.

```
void adm_finalize(adm_handle *handle)
```

*Parameters:* handle: The ADM (libadm) instance handle, see adm\_init().

*Return value:* adm\_finalize() has no return value.

adm\_get() retrieves a file from the ADM virtual filesystem by choosing a replica and transferring the file using GridFTP.

```
int adm_get(adm_handle *handle, char *vfs_path, char *local_path)
```

*Parameters:* handle: The ADM (libadm) instance handle, see adm\_init(). vfs\_path: The absolute path to a file in the ADM virtual filesystem. local\_path: The absolute or relative path where the file should be stored locally or NULL to store the file in the current working directory and preserve the filename.

*Return value:* adm\_get() returns 0 on success or one of the following error codes: ADM\_ERROR, ADM\_EINTERRUPT.

adm\_init() initializes the ADM library and returns a handle.

```
adm_handle *adm_init(char **msg)
```

*Parameters:* msg: If adm\_init() fails an error message is stored there.

*Return value:* adm\_init() returns an ADM instance handle on success or NULL in case of an error.

adm\_link() creates a link to a file in the virtual filesystem (similar to a Unix hardlink)

```
int adm_link(adm_handle *handle, char *spath, char *dpath)
```

*Parameters:* handle: The ADM (libadm) instance handle, see adm\_init().

spath: The absolute path to a file in the virtual filesystem for which a link should be created.

dpath: The absolute path to a file which should point to the same LFID like the path in spath.

*Return value:* adm\_link() returns 0 on success or one of the following error codes: ADM\_ERROR, ADM\_ESERVICE.

adm\_mkdir() creates a directory in the virtual filesystem.

```
int adm_mkdir(adm_handle *handle, char *vfs_path)
```

*Parameters:* handle: The ADM (libadm) instance handle, see adm\_init(). vfs\_path: The absolute path in the ADM virtual filesystem. The parent directory must exist.

*Return value:* adm\_mkdir() returns 0 on success or one of the following error codes: ADM\_ERROR, ADM\_EEXIST.

adm\_move() moves or renames a file or directory in the virtual filesystem.

```
int adm_move(adm_handle *handle, char *src_vfs_path, char *dest_vfs_path)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `src_vfs_path`: The source path of the file or directory in the virtual filesystem. `dest_vfs_path`: The destination path of the the file or directory in the virtual filesystem.

*Return value:* `adm_move()` returns 0 on success or one of the following error codes: `ADM_ERROR`, `ADM_ESERVICE`.

`adm_propdel()` removes a property from a file in the virtual filesystem.

```
int adm_propdel(adm_handle *handle, char *vfs_path, char *prop_name)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `vfs_path`: The absolute path of the file in the ADM virtual filesystem. `prop_name`: The name of the property which should be deleted.

*Return value:* `adm_propdel()` returns 0 on success or one of the following error codes: `ADM_ERROR`, `ADM_ESERVICE`.

`adm_propset()` adds a property to a file.

```
int adm_propset(adm_handle *handle, char *vfs_path, char *prop_name, char *prop_value)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `vfs_path`: The absolute path of the file in the ADM virtual filesystem. `prop_name`: The name of the property. `prop_value`: The value (content) of the property.

*Return value:* `adm_propset()` returns 0 on success or one of the following error codes: `ADM_ERROR`, `ADM_ESERVICE`.

`adm_readdir()` returns a list of directory entries from the ADM virtual filesystem.

```
adm_list_t *adm_readdir(adm_handle *handle, char *vfs_path)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `vfs_path`: The absolute path to a directory in the ADM virtual filesystem.

*Return value:* `adm_readdir()` returns a list of directory entries on success or `NULL` on error.

`adm_rmdir()` removes a directory from the virtual filesystem.

```
int adm_rmdir(adm_handle *handle, char *vfs_path)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `vfs_path`: The absolute path in the ADM virtual filesystem.

*Return value:* `adm_rmdir()` returns 0 on success or one of the following error codes: `ADM_ERROR`, `ADM_ESERVICE`.

`adm_rmfile()` removes a file from the virtual filesystem.

```
int adm_rmfile(adm_handle *handle, char *vfs_path)
```

*Parameters:* `handle`: The ADM (`libadm`) instance handle, see `adm_init()`. `vfs_path`: The absolute destination path in the ADM virtual filesystem.

*Return value:* `adm_rmfile()` returns 0 on success or one of the following error codes: `ADM_ERROR`, `ADM_ESERVICE`.

`adm_rmrep()` removes a replica from a file in the virtual filesystem.

```
int adm_rmrep(adm_handle *handle, char *vfs_path, int fspace)
```

*Parameters:* handle: The ADM (libadm) instance handle, see adm\_init(). local\_path: The absolute or relative path to a local file which should be stored in ADM. vfs\_path: The absolute destination path in the ADM virtual filesystem. fspace: The file-space ID which identifies the replica uniquely for a given file (*vfs\_path*) in the ADM virtual filesystem.

*Return value:* adm\_rmrep() returns 0 on success or one of the following error codes: ADM\_ERROR, ADM\_ESERVICE.

### 3.2.1 Sourcecode Example

Following the reference documentation of the ADM API in the previous section this short section presents an example for accessing the virtual filesystem from within a C program. In order to use the ADM library the *adm.h* header file must be included:

```
#include <stdio.h>
#include <stdlib.h>
#include <adm.h>

int main()
{
    adm_handle *handle = NULL; /* libadm instance handle */
    FILE *fp = NULL;          /* Just a test-file */
    handle = adm_init(NULL);  /* libadm initialization */

    if (handle)
    {
        /* Now lets create a directory in the ADM virtual filesystem */
        if (adm_mkdir(handle, "/mydir") == 0)
        {
            printf("Directory /mydir successfully created.\n");
        }
        else
        {
            printf("Could not create directory: %s\n", adm_geterror(handle));
            return 1;
        }

        /* Create a file and put it in ADM */
        fp = fopen("myfile", "w+");
        if (fp)
        {
            fprintf(fp, "Hello ADM!\n");
            fclose(fp);
            if (adm_addfile(handle, "myfile", "/mydir") == 0)
            {
                printf("File successfully put in ADM /mydir\n");
            }
            else
            {
                printf("Could not put file in ADM: %s\n", adm_geterror(handle));
                return 1;
            }
        }
    }
}
```

```
    }
    else
    {
        printf("Could not open myfile\n");
    }
}
return 0;
}
```

The sample program creates a directory and registers a new file with the virtual filesystem. The bottom line is to remember three things: 1. The ADM library must be initialized by means of the function `adm_init()`. 2. Each ADM function needs an ADM handle as its first argument. The corresponding data type is defined in the ADM library. 3. When an ADM function fails, error messages are accessible by means of `adm_geterror(handle)`.

### 3.2.2 Compiling the Sourcecode

Point the environment variable `ADM_LOCATION` to the *libadm* installation directory. The sources of the the code example can now be compiled and linked to a program using the following command:

```
cc admtest.c -o admtest -I${ADM_LOCATION}/include -L${ADM_LOCATION}/lib -ladm
```

Currently, ADM provides bindings for C, only. A gateway for Java and Perl is scheduled for soon release.

## 4 Installation, Configuration and Administration of ADM

ADM is a common client-server-application. In order to distinguish between the server host accommodating the ADM server from the latter itself, i.e. hardware from software, this text prefers the notion of "ADM *service*" instead of "ADM server". This makes sense, regarding the fact, that *one host* usually provides *more than one service*.

The ADM distribution consists of two parts, namely the client program, described in Section 3.1 *Command-line Interface* and the service program, or tersely speaking "the service". The distribution is available for download from SVN at the following addresses:

```
# Checkout a working copy of the adm client
```

```
agrid064@alnitak:~$ svn co svn://svn.gac-grid.org/software/adm/trunk ↩
```

```
# Checkout a working copy of the adm server
```

```
agrid064@alnitak:~$ svn co svn://svn.gac-grid.org/software/admservice/trunk ↩
```

The service needs PostgreSQL and PL/pgSQL; see below. The client requires the libraries *libcurl*, *libcurl-dev*, *libssl* and *libssl-dev*, which usually ship with each common Linux distribution. If not, the libraries are available from <http://curl.haxx.se> or <http://www.openssl.org>, respectively.

Let `admsrvroot` be the superuser of the ADM server, `admsrvuser` be an AstroGrid-D-member, `admsrvhost` the host where the ADM server is to be installed and `admsrvport` the port where the ADM server ist listening for requests sent by ADM clients.



## 4.1 Client Installation and Configuration

Installing the ADM client consists of five steps: 1. Check out the software from the URL given above and change to the *adm* directory. Figure 5 on page 25 shows the most important items in *adm*. 2. Run the *bootstrap.sh* shell script. 3. Invoke *configure* with an appropriate *--prefix* depending on the user privileges and the preferred location. 4. Call *make install*. 5. Finally, extend *.bashrc* by the following command:

```
export ADM_SERVICE_URL=http://admsrvhost:admsrvport ↩
```

If the ADM service is up and running the client program *adm* can now connect.

## 4.2 Service Installation and Configuration

The ADM service requires Java 6 as well as a PostgreSQL data base and this text assumes, that the data base server is already installed and running and that the readers permissions are sufficient in order to create new tables and data sets. In addition, the ADM service needs the Procedural Language/PostgreSQL Structured Query Language (PL/pgSQL), which is used to implement custom methods applied to the data base tables. The PL/pgSQL is provided to PostgreSQL data base by means of

```
# How to provide PL/pgSQL on the command-line
agrid064@alnitak:~$ createlang plpgsql; ↩
```

*Remark.* This command requires superuser privileges with respect to the data base (not the operating system) and has therefore not been included in the set-up-script *rebuild.sh*; see below.

Installing the ADM service consists of six steps: 1. Check out the software from the URL given above. 2. Install the PL/pgSQL extension package if not already present. 3. Change to the

```
adm
|
+-- admadmin/           // ADM administration tool "admadmin"
|
+-- admclient/         // ADM command-line client program "adm"
|
+-- bootstrap.sh, configure.in, Makefile.am
|
+-- dist/              //
|
+-- doc/               // ADM Tutorial
|
+-- libadm/
|
+-- README
```

Figure 5: ADM client installation. Contents of the *adm* directory.

*admservice* directory and invoke `ant` there (don't modify *build.xml*). 4. Configure the file-spaces to be registered with the ADM service by means of the SQL script *provider.sql*. 5. Customize and invoke *rebuild.sh* to create the data base tables required by the virtual filesystem. 6. Finally, customize *admserv.conf* and invoke the service start-up script *admserv.sh*. Change to the directory *admservice*. Figure 6 on page 27 shows the most important items in *admservice*. For historical reference, *admservice* also contains the directory *dist*.

First off, the ADM service needs at least one file-space. Integrating file-spaces with the ADM service requires superuser privileges with respect to the PostgreSQL data base. It is recommended to customize the SQL script *provider.sql* in the *admservice/setup/* directory to set up the file-spaces, which is implicitly called by *rebuild.sh*, another script in the ADM service distribution; see below. The SQL script *provider.sql* reads:

```
\set ON_ERROR_STOP

-- +-----
-- | PROVIDER DATA
-- +-----
INSERT
  INTO contact (
    contact_id, firstname, lastname,
    email, telephone, institute
  ) VALUES (
    1, 'Thomas', 'Bruesemeister',
    'tbruese@ari.uni-heidelberg.de', '06223/54-1834',
    'Astronomisches Rechen-Institut Heidelberg'
  );

INSERT
  INTO provider (
    provider_name, description, contact_id
  ) VALUES (
    'ARI',
    'Astronomisches Rechen-Institut am Zentrum fuer Astronomie in Heidelberg',
    1
  );

-- +-----
-- | FILE SPACES
-- +-----
INSERT
  INTO file_spaces (
    file_space_id, url, status,
    provider_id, total_space, free_space
  ) VALUES (
    1, 'gsiftp://alnitak.ari.uni-heidelberg.de/opt/d-grid/adm/dev_fs01', 'a',
    1, 40000000000, 40000000000
  );
```

File-space registration involves three data base tables, namely *contact*, *provider* and *file\_spaces*. The *contact* table describes the people responsible for the set up and maintenance of the ADM service. The above version of *provider.sql* contains only one entry honoring the developer who brought the AstroGrid-D Data Management into being, Thomas Brüsemeister. The *contact* table can hold as many people as needed. Since a provider can dedicate more than one file-space to the

```

admservice
|
+-- bin/                               // Server configuration, startup and logging
|   |
|   +-- adm.{log|log.lck}
|   |
|   +-- admserv.conf
|   |
|   +-- admserv.sh
|
+-- build.xml
|
+-- lib/                               // ...
|   |
|   +-- admService.jar
|   |
|   +-- postgresql-8.2-506.jdbc3.jar
|
+-- PROTOCOL
|
+-- setup/                             // Data base table set up
|   |
|   +-- rebuild.sh
|
+-- src/de/astrogrid/adm/ // Java source code

```

Figure 6: ADM service installation. Contents of the *admservice* directory.

AstroGrid-D community, providers and file-spaces are maintained by means of different tables. The above entries in *provider* and *file\_spaces* are supposed to be self-explanatory so that no further discussion is needed. Also these tables can hold as many entries as required.

The shell script *rebuild.sh* creates the data base tables, ADM needs in order to implement the virtual filesystem and especially runs the above SQL script *provider.sql* setting up the file-spaces available for ADM. There are two modifications required before running *rebuild.sh*:

```

#!/bin/sh

DB=adm                                # Leave value unchanged
DBUSER=admsrvroot                     # (1) ADM superuser
DBHOST=admsrvhost                     # (2) Host where the ADM service resides

sql[1]=admdb.sql
sql[2]=functions/getpath.sql
sql[3]=functions/getnodeid.sql
sql[4]=provider.sql

for i in ${sql[*]}; do
  psql -h ${DBHOST} -f $i ${DB}
  if [ $? -ne 0 ]; then
    echo "Failed setting up the database schema."
    exit 1;
  fi


```

done

Even though, installing the ADM service is independent from the configuration file *admserv.conf*, the basic customization can be prepended in one go. According to Figure 6 on page 27 *admserv.conf* which resides in the *bin* directory. The basic configuration only needs the parameters *adm.db.host* and *adm.db.user* which are equivalent to the fields *DBUSER* and *DBHOST* in the above shell script *re-build.sh*. The configuration file *admserv.conf* reads as follows, where all parameters are highlighted:

```
adm.http.port=12000
adm.http.secure=no
javax.net.ssl.trustStore=../../cakey
javax.net.ssl.keyStore=../../hostcert.p12
# Sets the log level
# Possible values:  severe, warning, info, config, fine, finer, finest
adm.log.level=info
# Use grid-mapfile authorization?
adm.auth.gridmap=true
# Default grid-mapfile location is /etc/grid-security/grid-mapfile
adm.auth.gridmap-file=/home/Tux/rwahner/admservice/grid-mapfile
adm.db.host=admsrvhost
adm.db.name=adm
adm.db.user=admsrvroot
adm.db.password=foo
# Cache for the most recently used entries in the VFS
adm.cache.vfs.mru-cache-size=256
```

After successfully installation, the ADM service is started by means of the shell script *admserv.sh* located in the *bin* directory:

```
# Start ADM service
agrid064@alnitak:~$ ./admsrv.sh 
```

The *-h* flag of *admserv.sh* allows to start-up the ADM service as a background daemon. In the current release, the ADM service requires a restart after modifying the configuration file *admserv.conf*.

## 5 Experiences in Using ADM

We evaluated the ADM by performing basic functionality tests (cf. Section 5.1) and by using the ADM for managing data of the NBody use case (cf. Section ??).

### 5.1 Basic Functionality Tests

The ADM provides an easy to use interface. Particularly, the easy and uniform access to help pages are a welcome improvement over the Globus Replica Location Service. It also solves the dead link

problem by testing if a replica still exists or not. Because the ADM uses a basic storage protocol, i.e., `gsiftp`, it is possible to manipulate the physically stored files. In particular, a malicious user might replace replicas with faulty copies without notice of the ADM.

Further improvements of the ADM include the reduction of data transfers and the selection of the storage space. Currently, the ADM transfers all registered files from the source to some storage space. These data transfers are a potential bottleneck and may not be necessary at all. While the ADM supports multiple storage spaces, in the present version, it belongs to the user to select the space (other than the default) manually.

## 5.2 Performance and Scalability Test Environment Setup

On the long run, ADM should be subjected to systematic investigation concerning its behavior with respect to the number and size of the files and directories in the virtual filesystem as well as heavy multiple client access. The *performance of the virtual filesystem* is a measure for the ability to process many requests against the ADM service from one or more ADM clients. Those requests are creation, modification or deletion of files and directories. The *scalability* is a measure for the dependency of the performance on the "charging level" of the filesystem, e.g. the depth as well as the number of entries in the filesystem tree. Usually, a filesystem is said to scale strong (bad), if an increasing stock of entries causes operations to slow down, whereas the scaling is weak (good), otherwise. Obviously, weak scaling should be preferred, if it doesn't imply further drawbacks.

In order to care for a systematically configured testing environment, the ADM client distribution provides the Perl program `tree-gen.pl`, contributed by Ralf Wahner in January 2008. `tree-gen.pl` can be found either in the `dist` directory of the client distribution or in the directory `3_4/misc` which belongs to the L<sup>A</sup>T<sub>E</sub>X sourcecode of this deliverable (`svn export svn://svn.gac-grid.org/documents/wg-3/deliverables/3_4` downloads a non-working copy of Deliverable 3.4 *Distributed File Management 2.0 and Adaptation of Use Cases and Testing*).

Clear and brief, `tree-gen.pl` builds a file and directory structure in the virtual filesystem, based on three user-specified parameters: depth of the directory tree (`-t`), number of subdirectories per directory (`-s`) and number of files per directory (`-f`). There are more sophisticated flags available and each flag has a corresponding long-version; see below:

```
# tree-gen.pl: how to build a sample tree in the virtual filesystem
agrid064@alnitak:~$ tree-gen.pl -t 2 -s 2 -f 2 ↩
+-----+-----+-----+-----+
| Type | Directories | Files | Total |
+-----+-----+-----+-----+
| Inner |           3 |     6 |     9 |
| Leaf  |           4 |     8 |    12 |
| Total |           7 |    14 |    21 |
+-----+-----+-----+-----+
[ 1 of 21:   4%]: adm mkdir /performance-scalability/d0_0
[ 2 of 21:   9%]: adm add  /performance-scalability/d0_0/f1_0
[ 3 of 21:  14%]: adm add  /performance-scalability/d0_0/f1_1
[ 4 of 21:  19%]: adm mkdir /performance-scalability/d0_0/d1_0
[ 5 of 21:  23%]: adm add  /performance-scalability/d0_0/d1_0/f2_0
[ 6 of 21:  28%]: adm add  /performance-scalability/d0_0/d1_0/f2_1
[ 7 of 21:  33%]: adm mkdir /performance-scalability/d0_0/d1_0/d2_0
```

```

[ 8 of 21: 38%]: adm add /performance-scalability/d0_0/d1_0/d2_0/f3_0
[ 9 of 21: 42%]: adm add /performance-scalability/d0_0/d1_0/d2_0/f3_1
[10 of 21: 47%]: adm mkdir /performance-scalability/d0_0/d1_0/d2_1
[11 of 21: 52%]: adm add /performance-scalability/d0_0/d1_0/d2_1/f3_0
[12 of 21: 57%]: adm add /performance-scalability/d0_0/d1_0/d2_1/f3_1
[13 of 21: 61%]: adm mkdir /performance-scalability/d0_0/d1_1
[14 of 21: 66%]: adm add /performance-scalability/d0_0/d1_1/f2_0
[15 of 21: 71%]: adm add /performance-scalability/d0_0/d1_1/f2_1
[16 of 21: 76%]: adm mkdir /performance-scalability/d0_0/d1_1/d2_0
[17 of 21: 80%]: adm add /performance-scalability/d0_0/d1_1/d2_0/f3_0
[18 of 21: 85%]: adm add /performance-scalability/d0_0/d1_1/d2_0/f3_1
[19 of 21: 90%]: adm mkdir /performance-scalability/d0_0/d1_1/d2_1
[20 of 21: 95%]: adm add /performance-scalability/d0_0/d1_1/d2_1/f3_0
[21 of 21: 100%]: adm add /performance-scalability/d0_0/d1_1/d2_1/f3_1
Remark: use "tree-gen.pl -t 2 -s 2 -f 2 -c" to cleanup this tree.
done.

```

Figure 7 on page 30 illustrates the file and directory structure set up by means of the above parameters. The depth specified by `-t` concerns the directory part of the tree. The root node `d0_0` has depth 0. Two depth levels follow. Below `d2_0` and `d2_1` are the leaf file nodes, so the depth of the file tree is one more than the depth of the directory tree. The ASCII table immediately below the above command-line displays basic properties of the file and directory structure specified by the switches `-t`, `-s` and `-f`. The progress display has been included in the output in order to keep track of the program operation for large trees. Calling `tree-gen.pl` on the command-line without parameters reveals the full stock of flags available:

```
# tree-gen.pl: command-line flags
```

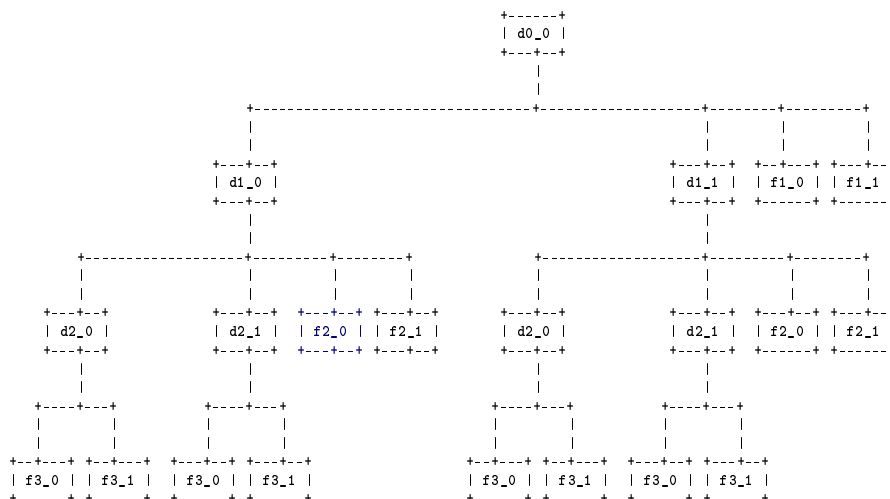


Figure 7: File and directory tree created with `tree-gen.pl`. The character `d` denotes a directory, `f` denotes a file. The underscore-separated integer numbers indicate depth of the node with respect to the root node and the zero-based "item count", e.g. `f2_0` (the blue node) is the first file node with respect to its parent directory node `d1_0` and resides on the second level of depth below the root node `d0_0`, which is always a directory.

```

agrid064@alnitak:~$ tree-gen.pl ↵
usage: tree-gen.pl
  -t|--dir-tree-height      <height of directory tree>
  -s|--dir-child-nodes      <number of subdirs per node>
  -f|--file-child-files     <number of files per node>
  [-d|--die-on-error]      <abort programm if adm return code != 0>
  [-u|--unify-file-content] <use unique dummy file content>
  [-a|--adm-directory-prefix] <mount point of d00_00>
  [-c|--cleanup]           (noarg, boolean switch)
  [-p|--tree-properties-only] (noarg, boolean switch)

```

The switches `-t`, `-s` and `-f` have been described above. By convention, programs return 0 after successful operation and a different value otherwise and so does the ADM client program `adm`. Due to its presetting, `tree-gen.pl` does not care for the return value provided by `adm`. The `-d` flag can be set to tell `tree-gen.pl` to interrupt its operation, in case `adm` returns something different from 0. By default, `tree-gen.pl` builds the file and directory structure below `/performance-scalability` in the virtual filesystem. The `-a` flag allows to choose a different directory. Also by default, the content of the dummy file generated by `tree-gen.pl` is unique inside one run, however it is deterministic. This means, that two different users running `tree-gen.pl` with the same command-line parameters will try to place the same files in the virtual filesystem. (Recall, that ADM uses a 32 bit hash value as file name which is generated from the *file content* by means of the Message Digest 5 algorithm, i.e. congruent files have identical hash values.) The `-u` flag extends the dummy file content by user-specific data and a timestamp, making the content unambiguous. Even though the `-c` flag is a remnant from those days when ADM could not yet delete directories recursively, it is still useful and cleans up the directory structure according to the compulsory command-line parameters `-t`, `-s` and `-f`.

Finally, the `-p` switch does *not* build the file and directory tree but displays the "data sheet" matching the command-line parameters `-t`, `-s` and `-f`; see above output of `tree-gen.pl -t 2 -s 2 -f 2`. The `-p` switch is quite useful to check the tree size before starting `tree-gen.pl` in real operation, since the node count rapidly increases due to exponential parameters in the formula. The number of directory nodes evaluates to a geometric series:

$$D = 1 + s + s^2 + \dots + s^t = \sum_{i=0}^t s^i = \frac{1 - s^{t+1}}{1 - s}$$

where  $t$  and  $s \neq 1$  correspond to the flags `-t`, i.e. the depth of the directory tree, and `-s`, i.e. the number of subdirectories. If  $s = 1$ , i.e. one subdirectory per directory, the number of directory nodes is equal to  $t + 1$ . Let furthermore  $f$  correspond to the flag `-f`, i.e. the number of files per directory. Now the number of file nodes computes to  $F = f \cdot D$  and the full file and directory tree owns a total of  $D + F = (1 + f) \cdot D$  nodes. There are  $s^t$  leaf directory nodes and  $f \cdot s^t$  leaf file nodes, which means that there are  $D - s^t$  inner directory nodes and  $F - f \cdot s^t$  inner file nodes. The "data sheet" behind the `-p` flag relies on these formulas.

## References

- [1] AstroGrid-D Data Management (ADM) build-in documentation, accessible by means of `adm help` (general information) or `adm help <subcommand>` (manual page for individual subcommand).

- [2] Deliverable 3.2: *Distributed File Management, Data- and Replica-management in AstroGrid-D*, (Version 1.0.0, public release with comments incorporated).
- [3] Deliverable 3.3: *Distributed File Management, Tests of the Data- and Replica-Management Software for Selected Use Cases*, (Version 1.0.0).
- [4] Collins-Sussman, Ben; Fitzpatrick, Brian W. and Pilato, C. Michael: *Version Control with Subversion*, for Subversion 1.4 (Compiled from r2866), see <http://svnbook.red-bean.com> or file *svn-book.pdf* located in directory *.../misc*.
- [5] The official `globus-url-copy` website: *globus-url-copy — Multi-protocol data movement* at <http://www.globus.org/toolkit/docs/4.0/data/gridftp/rn01re01.html>.