ALICE Analysis User Guide
HOWTO - Basics for distributed Analysis on the AliEn platform
V. 0.0 µ 06/15/06 11:14:32 AM
Author: Andreas Peters
Email: Andreas.Joachim.Peters@cern.ch
This document makes you familiar with basic AliEn user interfaces and tries to enable you to run analysis as an ALICE user using the AliEn infrastructure.

The first part describes the installation procedure for the application interface client package **gapi** and the functionality of the command line interface – the AliEn shell **aliensh**.

The second part describes the AliEn GRID interface for the ROOT/AliROOT framework and introduces you to distributed Analysis examples in batch style using the AliEn infrastructure.

To be able to repeat the steps described in this HowTo, you must have a valid GRID certificate and you must be a registered user in the AliEn ALICE VO. Information about that can be found under

[http://lcg.web.cern.ch/LCG/users/registration/registration.html](http://lcg.web.cern.ch/LCG/users/registration/registration.html)

and

[http://alien.cern.ch/twiki/bin/view/AliEn/HowToGetCertificate](http://alien.cern.ch/twiki/bin/view/AliEn/HowToGetCertificate)
2 The client interface API

All user interactions with AliEn are handled using a client-server architecture as seen in the figure above. The API for all client applications is implemented in a library libgapiUI.so. Every client interface can communicate over a session-based connection with an API server. The API server exports to the client interface all functions, which are bridged from the AliEn PERL-UI via the AlienAS PERL→C++ interface script. To use the shared library in a C++ program, just add

```
#include <gapiUI.h>
```

to the source and link with -lgapiUI.

2.1 Installation of the client interface library package – gapi

The standard way to install the client interfaces package is to use the AliEn installer. The source installation is explained for special use cases - you might skip this section.

2.1.1 Installation via the AliEn installer

- Download the AliEn installer from:

  [http://alien.cern.ch/alien-installer](http://alien.cern.ch/alien-installer)

  using a browser or wget. Set the permissions for execution of the installer:

  `chmod ugo+rx alien-installer`

- Run the installer:

  `./alien-installer`

  Select in the version menu the highest version number with the 'stable' tag and follow the instructions of the installer until you are asked, which packages to install. Select 'gshell - Grid Shell UI' to install the client interface and ROOT to get an AliEn enabled ROOT package and proceed.

  **Hint:** The installer asks you, where to install the packages. The default location is
To create this directory you need ROOT permissions on that machine. If you are not the administrator of that machine you can only install into your HOME directory or a directory, where you have write permissions.

### 2.1.2 Recompilation with your locally installed compiler

AliEn comes with the CERN standard compiler (currently gcc 3.2.3). If you want to compile applications with your locally installed compiler (different from AliEn) and link against the API library, you have to recompile the API library with your locally installed compiler.

To do so, you execute the script

```
/opt/alien/api/src/recomplile.api [<alien dir> [modules]]
```

The script recompiles the API library and installs over the binary installation in your AliEn installation directory.

If you execute without arguments, it detects your installation directory configured in the installer. If you want to recompile a different installation than the one installed via the installer you can pass the installation directory as the first argument. If you add 'modules' as the 2nd argument, also the PERL and JAVA modules are rebuild, but be aware that you need to have also the 'client' package (including the PERL distribution in AliEn) installed.

If you recompiled successfully you should set the variable GSHELL_GCC to the path of your gcc executable (f.e. `export GSHELL_GCC=`/bin/gcc`).

### 2.1.3 Source Installation using AliEnBits

- Create a directory to install the build sources f.e.:
  ```
  mkdir $HOME/alienbits/ ; cd $HOME/alienbits/
  ```

- RSync to the development HEAD
  ```
  ```

  or an older version f.e v2-6:
  ```
  ```

- Login in to the CVS with password 'cvs':
  ```
  cvs -d :pserver:cvs@alisoft.cern.ch:/soft/cvsroot login
  ```

- Update your rsynced directory with the latest CVS:
  ```
  cvs -d :pserver:cvs@alisoft.cern.ch:/soft/cvsroot update -dPA
  ```

- Run the configure script and define the installation location:
  ```
  ./configure --prefix=/opt/alien (or f.e. --prefix=$HOME/alien)
  ```

- Change to the api-client source directory:
  ```
  cd $HOME/alienbits/apps/alien/api
  ```

  [or the api-client-server source directory:
  ```
  cd $HOME/alienbits/apps/alien/apiservice
  ```

- Start the build:
  ```
  make
  ```
AliEn Bits will download and compile every package, which is needed to build the API. This includes the compilation of PERL and GLOBUS and will take about 1 hour.

- Install the build:

  ```
  make install
  ```

Now you have compiled everything what is used by the API package from sources on your computer.

### 2.1.4 The directory structure of the client interface

The location of the client interface installation is (if not changed with the Installer or in AliEnBits with the `-prefix` switch) under `/opt/alien/api`

The structure of the default installation is as shown:

```
- /opt/alien/api/
  - bin/
    - aliensh
      - this is the executable for the AliEn shell interface
      - there are no modification of the PATH or LD_LIBRARY_PATH environment variables needed to run the shell.
    - gbbox
      - the busy box executable to run arbitrary commands (we will give a more detailed explanation later)
    - alien_<cmd>
      - wrapped `aliensh` commands
  - etc/
    - shell scripts defining auto-completion functions for the `aliensh`
  - include/
    - gapi_attr.h
      - C++ include file defining gapi file attributes
    - gapi_dir_operations.h
      - C++ include file defining POSIX directory operations like opendir, closedir etc.
    - gapi_file_operations.h
      - C++ include file defining POSIX file operations like open, close, read etc.
      - **Warning**: currently the POSIX operations are disabled to avoid library conflicts with ROOT and xrootd
    - gapi_job_operations.h
      - C++ include file defining job interface operations like submit, kill, ps etc.
```
gapi_stat.h

C++ include file defining a POSIX stat interface

gapiUI.h

C++ include file containing the interface class for command execution and authentication on a remote apiservice server. This can be considered as the lowest level interface and is completely independent of AliEn. Other include files encapsulate more abstract functionalities, which already 'know' the AliEn functionality.

lib/

java/

the java interface to libgapiUI, if installed

libgapiUI.a

libgapiUI.la

libtool library file

libgapiUI.so

link to a current version of the gapiUI shared library

libgapiUI.so.2

link to a current version of the gapiUI shared library

libgapiUI.so.2.0.2

a current version of the gapiUI shared library (the version number might have changed in the meanwhile)
### 2.2 Using the Client Interface - Configuration

A minimum configuration to use the API client is recommended, although the API client works also without PATH or LD_LIBRARY_PATH modifications. This modifications are for user convenience to avoid typing the full executable paths:

- If you don't have already a proper GLOBUS setup, you should add the following to your login script, which set's the PATH and LD_LIBRARY_PATH for GLOBUS.

  The example is given for bash and for the default location '/opt/alien'. If you installed in another PATH use that one instead and use the syntax of your shell for environment variables!

  ```bash
  export GLOBUS_LOCATION=/opt/alien/globus/
  export X509_CERT_DIR=$GLOBUS_LOCATION/share/certificates
  export PATH=$GLOBUS_LOCATION/bin:$PATH
  export LD_LIBRARY_PATH=$GLOBUS_LOCATION/lib:$LD_LIBRARY_PATH
  ```

- If you want to have the executable of `aliensh` and the token executables in your PATH use:

  ```bash
  export PATH=/opt/alien/api/bin:$GLOBUS_LOCATION/bin:$PATH
  ```

Additional files for the server:

- `sbin/`<br>  - `gapiserver`<br>    - the server executable<br>- `scripts/`<br>  - `LocalFS.pl`<br>    - a test interface script to be used by the `gapiservice` executable<br>  - `AlienAS.pl`<br>    - a PERL script interfacing from the C++ server to the AliEn native PERL UI<br>  - `gapiserver.pl`<br>    - the startup script for the API server `gapiserver`
2.3 Using the Client Interface - Authentication

Session tokens are used for performance reasons for authentication and identification to any API server. These tokens are similar to GRID proxy certificates (limited lifetime). Additionally they are modified every time they have been used. Every token defines one well defined role, which you specify in the initialization of the token. This token is issued by the API server and shipped over an SSL connection to the client.

2.3.1 Token Location

The gapi library supports two methods to store an authentication token:

- Store in memory
  - The token is only usable by one application. This method is used f.e. in ROOT and will be explained later.
- Store in a token file.
  - This method is used for the shell implementation aliensh.

In the following we will discuss the handling of file-tokens.

2.3.2 File-Tokens

The file-token mechanism is implemented in three commands:

- `alien-token-init`
- `alien-token-info`
- `alien-token-destroy`

None of the commands touches your present user environment i.e. they don't modify your PATH or LD_LIBRARY_PATH environment variables.

2.4 Session Token Creation

Session tokens are stored in the `/tmp` directory following the naming convention:

```
/tmp/gclient_token_${UID}
```

For security reasons permissions on the token file are set to “-rw-------” for the owner of the token.

There are three authentication methods for the API service:

- using GRID proxy certificates
- using CERN AFS password
- using AliEn job token

**Hint:** It is always recommended to use a GRID proxy certificate for authentication. AFS and Job Token are described for completeness.
2.4.1  Token Creation using a GRID Proxy certificate – alien-token-init

- To obtain a session token you need a GRID proxy certificate
  (i.e execute `grid-proxy-init`)
- execute `alien-token-init [role]` to contact one of the default API services
  and to obtain a session token for this service

  ➤ the [role] parameter is optional. If you don’t specify a role, your local unix
  account name ($USER) is taken as the role in the session request. You can
  request the middle ware administrators to allow you to use other roles than your
  personal identity, e.g. the aliprod role for generic productions.

- The list of default API services is obtained automatically. The client tries to
  connect in a well defined manner to one of the available services and to
  establish a session.

  If none of the (redundant) services are available, no token will be created.

- You can define your API server endpoint and role defaults via the following
  environment variables:
  - `alien_API_HOST`:
    specifies the API server host name e.g. pcapiserv01.cern.ch
  - `alien_API_PORT`:
    specifies the API server information port, default 9000
  - `alien_API_USER`:
    specifies the role you want to request with your certificate in AliEn
  - `alien_API_VO`:
    specifies the Virtual Organization to be used. Presently this parameter
    is not used.

- The lifetime of session tokens is not chosen by the client. It depends only on the
  configuration of the API server you are connecting.

- `alien-token-init` has an auto-bootstrap feature:
  - the first time you use the command, it will run the bootstrap procedure, which
    will inform you about the creation of certain symbolic links etc.
  - if you move or copy a client interface installation to another directory, the
    installation bootstraps itself (the first time, when you use the alien-
If you miss some of the required libraries, you will get a hint, how to proceed in such a case.

Return Values ($?):

0  -  Token successfully obtained
98 -  Bootstrap error – no permissions
99 -  No Token created – authentication failed

Hint: if you have problems like shown here

```shell
globus_gsi_gssapi: Error with GSI credential
globus_gsi_gssapi: Error with gss credential handle
globus_gsi_gssapi: Error with GSI credential
globus_sysconfig: Could not find a valid trusted CA certificates directory: The trusted certificates directory could not be found in any of the following locations:
1) env. var. X509_CERT_DIR
2) $HOME/.globus/certificates
3) /etc/grid-security/certificates
4) $GLOBUS_LOCATION/share/certificates
```

set the environment variable X509_CERT_DIR to the certificate file directory. In case of a standard installation you should do:

```
'export X509_CERT_DIR=/opt/alien/globus/certificates'
```

### 2.4.2 Token Creation using a password – alien-token-init

Users can authenticate with a CERN account name and password, if they ask the administrators to allow that. This is not recommended but possible. The procedure is the same like with GRID proxy certificates and is explained for completeness.

One has to specify the account name as the `[role]` parameter to `alien-token-init` and you will be prompted to enter a password. The client uses an SSL connection to communicate the password, which is validated on the server side using the PAM library. Afterwards the password is immediately overwritten in the client memory.

```
$ alien-token-init
```

### 2.4.3 Token Creation via AliEn Job Token – alien-token-init

A third authentication method via job tokens exists: if jobs are run via the AliEn TaskQueue, you can obtain a token using the AliEn job token known in the environment of
Your GRID batch job. In this case you don’t need to specify any [role] parameter. The role is mapped automatically according to the AliEn job token found in your job sandbox and mapped on server side using the AliEn authentication service.

### 2.4.4 Manual setup of redundant API service endpoints

You can specify a list of API service machines via the environment variable `alien_API_SERVER_LIST` f.e.:

```
export alien_API_SERVER_LIST="host1:9000|host2:9000|host3:9000"
```

A default list of API services is taken if you leave the environment variable undefined.

### 2.5 Checking an API session token – alien-token-info

Like `grid-proxy-info` you can execute `alien-token-info` to get information about your session token.

```
Xapiclient@pcapiv01:~ $ /home/apiclient/proc/apiv01.cern.ch
```

- **Host** - host name of the API server, where this token is usable
- **Port** - communication port for remote command execution
- **Port2** - information port, where the initial authentication methods and protocol versions are negotiated
- **User** - role associated with this token
- **Pwd** - random session password used in the security scheme
- **Nonce** - dynamic symmetric CIPHER key
- **SID** - assigned session ID by the API server
- **Encr. Rep.** - 0 specifies, that all service replies are not encrypted. 1 specifies, that all service replies are fully encrypted.

**Hint:** the default operation mode is, that all client requests are fully encrypted, while server responses are not encrypted.

- **Expires** - Local Time, when the token will expire

If the token has expired, the last line of the command will be “Token has expired!”, in case it is valid “Token is still valid!”. If no token can be found, the output will be only “No token
Return Values ($?):

0 - Token is valid
1 - Token is not valid
2 - No Token found

2.6 Destroying an API session token – alien-token-destroy

alien-token-destroy is used to destroy on existing session token.

Return Values ($?):

0 - Token has been removed
100 - No Token to be removed

2.7 Session Environment Files

The alien-token-init command produces two other files in the /tmp directory.

- /tmp/gclient_env_$UID
  - You can source this file, if you want to connect automatically from an application to the same API server without specifying host and port name in the code. This will be explained more in detail later in the section about the ROOT TAlien interface.

- /tmp/gbbox_$UID_$PID
  - This file keeps all the 'client-states' which are not directly connected with authentication or session variables (which are written to the token file): for the moment only the CWD is stored. This file is bound to the PID of your shell to allow different CWDs in different shells.
  - You can define the 'client-state' file by setting the environment variable GBBOX_ENVFILE.
    Nevertheless the setting is done automatically in the AliEn Shell aliensh.

3 The AliEn Shell – aliensh

The aliensh provides you with a set of commands to access AliEn GRID computing resources and the AliEn virtual file system.

The aliensh can be run as an interactive shell, for single-command- and script-execution.

There are three command categories:

- informative + convenience commands
- Virtual File Catalogue + Data Management Commands
- TaskQueue/Job Management Commands
3.1 Shell Invocation

3.1.1 Interactive Shell Startup/Termination

The shell is started with the 'aliensh' executable as seen in the following example:

The shell advertises the present protocol version used on client side (aliensh 2.0.0 in this case) and displays the server message of the day (MOTD) after invocation.

You can execute commands by typing them into the shell and invoking the execution with the ENTER key. In reality aliensh is a special featured bash shell.

Shell return values are all constructed following common practice:

Return Values ($?):

0 - Command has been executed successfully
!=0 - Command has produced 'some' error

Most of the shell commands provide a help text, if executed with a '-h' flag.

You can leave the shell with the 'exit' command:

3.1.2 Shell Prompt

The shell prompt displays on the left hand side the endpoint URL specifying alien as the protocol, the user role (peters) as the user name and the API server host- and port name (pcapiserv01.cern.ch:9000). The right hand side of the prompt displays the user's CWD. If the shell is invoked the first time, the CWD is set to the user's home directory. Otherwise the CWD is taken from the previous session (as stored in the file defined via env.GBBOX_ENVFILE).

3.1.3 Shell History

The shell remembers the history of previously executed commands. These commands are
stored in the history file $HOME/.aliensh_history.

3.1.4 Shell Environment Variables

aliensh provides the following environment variables:

- alien_API_HOST
- alien_API_PORT
- alien_API_USER
- alien_API_VO
- MONALISA_NAME
  - Mona Lisa Site Name f.e. 'CERN'
- MONALISA_HOST
  - Responsible Host running a Mona Lisa Server
- APMON_CONFIG
  - Responsible Mona Lisa Server to receive ApMon UDP packets
- HOME
  - Your virtual AliEn home directory
- GBBOX_ENVFILE
  - Environment file keeping shell state variables (f.e. CWD)
- GSHELL_ROOT
  - Installation path of the API client package

3.1.5 Sourcing scripts inside aliensh (run/source)

If you want to source a script which is stored in the file catalogue or on the local disk, you can use 'run <filename>' for a file in the catalogue or 'run file:<filename>' for a file on your local disk. The traditional 'source' command sources only local files and is a bash reserved word.

3.1.6 Single Command Execution

You can execute a single command by invoking the shell with the command as the first parameter:

```
Return Values ($?):

0 - Command has been executed successfully
```
1 - Command has been executed and returned an error
127 - Command does not exist and could not be executed
255 - Protocol/Connection error

### 3.1.7 Script File Execution

You can execute an `aliensh` script by giving the script filename as the first argument to `aliensh`. Local script files have to be prefixed with 'file:', otherwise the script is taken from the AliEn file catalog!

The final line of the script output 'exit' is produced, when the aliensh terminates and is not part of the user script.

**Return Values ($?):**

- **0** - Script has been executed successfully and exited with 0
- **1** - Script has been executed and returned and exited != 0
- **127** - Script does not exist and could not be executed
- **255** - Protocol/Connection error

### 3.2 Basic aliensh Commands

#### 3.2.1 whoami

- ➔ print the user's role on stdout.

**Return Values ($?):**

- **0** - Successfully executed
- **255** - Protocol/Connection error

#### 3.2.2 clear

- ➔ clear your terminal window

#### 3.2.3 pwd

- ➔ print the CWD on STDOUT
Return Values ($\$?):

0 - Successfully executed
255 - Protocol/Connection error

3.2.4 \texttt{gbbox [-d] [-h] [-f <tag1>,[<tag2>]..]}

- Busy Box command, used to execute most of the \texttt{aliensh} commands.
  Example: 'whoami' executes f.e. in reality like 'gbbox whoami'

The API protocol returns always four streams in the response:

- STDOUT
- STDERR
- Results Structure (Array of Key-Value pairs)
- Misc. Hash (Array of Key-Value pairs for environment variables etc.)

The shell prints only the STDOUT and STDERR stream on your terminal.
With the '-d' option, you get a dump of all returned streams:

The '-o <tag>' option prints only the tag <tag> of the result stream to your terminal:

In general: the command is very useful to make a quick check, what a busy box command returns - especially with respect to the API library, where you can easily access directly any tag in the result structure.
Return Values ($?):

- 0 - Command has been executed successfully
- 1 - Command has been executed and returned an error
- 127 - Command does not exist and could not be executed
- 255 - Protocol/Connection error

3.2.5 cd [<directory>]

→ change to the new working directory <directory>. If the directory is unspecified, you are dropped into your home directory. <directory> can be a relative or absolute PATH name. Like in standard shells 'cd -' drops you in the previous visited directory – if there is no previous one, into your home directory.

Return Values ($?):

- 0 - Directory has been changed
- 1 - Directory has not been changed
- 255 - Protocol/Connection error

3.2.6 ls [-laFb|m] [<directory>]

→ list the current directory or <directory>, if specified.

- '-l' list in long format
  - the long format uses color highlighting
    - blue : plain files
    - green : directories
    - red  : JDL files
- '-a' list also hidden files starting with '.'
- '-b' list in guid format (guid + name)
- '-m' list in md5 format (md5 + name)
- '-F' list directories with a trailing '/'
- '-n' switch off the color mode for the long format
Hint: if you want to use the `ls` function in scripts and parse the output, you should always add the `-n` flag to switch off the color mode for the `-l` long format.

Return Values ($?):

0 - Directory has been listed
1 - Directory has not been listed
255 - Protocol/Connection error

3.2.7 `mkdir [-ps] <directory>`

- create the directory `<directory>`
  - `-p` automatically creates recursive all needed subdirectories
  - `-s` makes the command silent – no error output

Return Values ($?):

0 - Directory has been created
1 - Directory has not been created
255 - Protocol/Connection error

3.2.8 `rmdir [-r] <directory>`

- remove the directory `<directory>` or everything under the directory tree `<directory>`
  - `-r` removes everything including files and directories recursively under
Hint: Be aware, that if you use the `-r` flag, you remove only entries from the file catalogue, but not their physical files on a mass storage system!

### 3.3 Data Management Commands

#### 3.3.1 register

Users should not register external files, which are not physically moved into AliEn SEs using the aliensh or applications based on libgapi.so.

#### 3.3.2 cp [-s ][-d] [-v] [-m][[-n] [-t <time>] [-f] [-i <infofile>]<src> <dst>

- copy file(s) from <src> to <dst>: **This command always produces a physical copy of a file. It does not only copy an entry in the catalogue to a new name!**
- <src> and <dst> have to be given with the following URL-like syntax:

  ```
  [alien:|file:] <path|dir>[@<SE>]
  ```

**Examples:**

- **“alien:* .root”** specifies all '.root' files in the AliEn CWD
- **“alien:/alice/cern.ch/”** specifies the “/alice/cern.ch/” directory in AliEn
- **“file:/tmp/myfile.root”** specifies the local file '/tmp/myfile.root' on your computer
- **“myfile.root@ALICE::CERN::Castor2”** specifies the file “myfile.root” in the CWD in AliEn in the ALICE::CERN::Castor2 Storage Element

- if a file has several replicas and you don’t specify a source SE, the closest SE specified in the environment variable alien_close_SE is taken. If this is not set, the closest one to ALICE::CERN::* is selected as the source.
- if you don’t specify a target SE, the SE specified in the environment variable alien_close_SE is taken. If this is not set, the default one of CERN is used (Castor).
- if <src> selects more than one file (f.e. ‘*.root’) <dst> must be a
if <src> is a directory (f.e. '/tmp/mydir/') <dst> must also be a directory!

**Options:**

- **'s'** be silent, in case of no error, don't print anything
- **'d'** switch on the debug output. This can be useful to find reasons for command failures
- **'-v'** more verbosity – print summaries about source, destination, size and transfer speed
- **'-m'** MD5 sum computation/verification
  - local to grid file: compute the md5 sum of the local file and insert it in the catalog. This is 'on' by default in this operation mode.
  - grid to local file: verify the md5 sum of the downloaded file
  - local to local file: flag ignored
  - grid to grid file: recompute the md5 sum of the source file and enter it in the catalog with the destination file
- **'-n'** new version – this option is only active, if the destination is a file in AliEn: if the destination file exists already, the existing destination is moved into the subversions directory and labeled with a version number starting with v1.0. If the new version to be copied has the same md5sum, the file is not copied at all!

**Example:**

- If you overwrite your file 'myfile' with the '-n' option, the original 'myfile' is moved to '.myfile/v1.0' and 'myfile' contains the last copied file. If you repeat this, 'myfile is moved to '.myfile/v1.1' a.s.o.
- **'t' <seconds>**
  
specifies the maximum waiting time for all API service requests (which are executed within the copy command). The default is 10 seconds.

- **'-f'** 'force' the transfer. This sets the first connection timeout to 1 week and the retry timeout (if a connection is interrupted during a transfer) to 60 seconds. Use this flag, if you absolutely want a transfer being executed. The default is 15s for the first connection timeout and 15 seconds for retries. In interactive shells however, it is more user friendly, if a copy command times out automatically.

- **'-i <file>'**
  
writes an info file to the local file <file>, containing the <src> and <dst> parameters used, f.e. you can get the information, from which SE a file has been read, if you didn't specify it in <src>.
Return Values ($\?):

0  -  File(s) copied successfully or you have just requested the '-h' option for the usage information

1  -  Directory has not been listed

20 -  access to the source file has been denied

21 -  access to the destination file has been denied

22 -  timeout while getting an access envelope

23 -  you tried to copy multiple files, but the destination was not a directory

24 -  could not create the target destination directory

when copying multiple files

by 'i infofile.

25 -  copy worked, but couldn't write the info file given

30 -  the md5sum of the source file is not '0' and the md5sum of the downloaded file differs from the catalog entry.

200 -  Control-C or Sig Interrupt received

250 -  an illegal internal operation mode has been set

(can happen only in 'buggy' installations/setups)

251 -  <src> has not been specified

252 -  <dst> has not been specified

253 -  you miss the xrootd xcp or xrdcp program

- the first place to search for is in your PATH environment. If it can't be found there, it tries to find it in your API installation directory or one level (../) higher

255 -  the copy operation itself to/from the SE failed

Examples:

• copy local to AliEn file:
• copy AliEn to local file:

```
[alice@aliprodBcpapers01 cern.ch]:/alice/cern/user/aliprod/demo]$ cp testfile:/tmp/testfile
```

• copy local to AliEn file with verbose option:

```
[alice@aliprodBcpapers01 cern.ch]:/alice/cern/user/aliprod/demo]$ cp -v -f testfile:/tmp/testfile
```

• copy local to existing AliEn file creating new versions:

```
[alice@aliprodBcpapers01 cern.ch]:/alice/cern/user/aliprod/demo]$ cp -v -f testfile:/tmp/testfile
```

• copy AliEn to local file – write copy information file

```
[alice@aliprodBcpapers01 cern.ch]:/alice/cern/user/aliprod/demo]$ cp -v -f testfile:/tmp/testfile
```

3.3.3 **cp (virtual)**

Users cannot copy catalog entries without copying the physical files attached behind. This is important to enforce permissions set in the virtual file catalog.

3.3.4 **rm [-s] [-f] <file>**

- remove the entry `<file>` from the file catalogue
  - `-s` - be silent – don’t print ERROR messages!
Return Values ($?):

0   -   File has been removed
1   -   File could not be removed
252 -   <file> parameter missing
255 -   Protocol/Connection error

3.3.5  **cat <file>**

👉 print the file <file> on STDOUT

👉 <file> has an URL-like syntax and can reference an AliEn or local file:

- “cat file:/tmp/myfile.txt” - print the local file '/tmp/myfile.txt'
- “cat myfile.txt” - print the AliEn file 'myfile.txt' from the CWD
- “cat alien:/alice/myfile.txt” - print the AliEn file 'alice/myfile.txt' (the protocol 'alien:' is optional within the shell).

Return Values ($?):

0   -   File has been printed with 'cat'
1   -   File could not be printed with 'cat'
246 -   the local copy of the file could not be removed
250 -   an illegal internal operation mode was set (can happen only in 'buggy' installations/setups)
252 -   <file> parameter missing
255 -   Protocol/Connection error

3.3.6  **more <file>**

👉 use 'more' to print the file <file> on STDOUT

👉 <file> has an URL-like syntax and can reference an AliEn or local file:

- “more file:/tmp/myfile.txt” - print the local file '/tmp/myfile.txt'
- `more myfile.txt` - print the AliEn file 'myfile.txt' from the CWD
- `more alien:/alice/myfile.txt` - print the AliEn file '/alice/myfile.txt' (the protocol 'alien:' is optional within the shell).

Return Values ($?):

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>File has been printed with 'more'</td>
</tr>
<tr>
<td>1</td>
<td>File could not be printed with 'more'</td>
</tr>
<tr>
<td>246</td>
<td>the local copy of the file could not be removed</td>
</tr>
<tr>
<td>250</td>
<td>an illegal internal operation mode was set (can happen only in 'buggy' installations/setups)</td>
</tr>
<tr>
<td>252</td>
<td>&lt;file&gt; parameter missing</td>
</tr>
<tr>
<td>255</td>
<td>Protocol/Connection error</td>
</tr>
</tbody>
</table>

### 3.3.7 less <file>

- use 'less' to print the file <file> on STDOUT
- <file> has an URL-like syntax and can reference an AliEn or local file:
  - `less file:/tmp/myfile.txt` - print the local file '/tmp/myfile.txt'
  - `less myfile.txt` - print the AliEn file 'myfile.txt' from the CWD
  - `less alien:/alice/myfile.txt` - print the AliEn file '/alice/myfile.txt' (the protocol 'alien:' is optional within the shell).

Return Values ($?):

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>File has been printed with 'less'</td>
</tr>
<tr>
<td>1</td>
<td>File could not be printed with 'less'</td>
</tr>
<tr>
<td>246</td>
<td>the local copy of the file could not be removed</td>
</tr>
<tr>
<td>250</td>
<td>an illegal internal operation mode was set (can happen only in 'buggy' installations/setups)</td>
</tr>
<tr>
<td>252</td>
<td>&lt;file&gt; parameter missing</td>
</tr>
<tr>
<td>255</td>
<td>Protocol/Connection error</td>
</tr>
</tbody>
</table>

### 3.3.8 edit [-c] <file>

- use to edit local or AliEn files with your preferred editor. The file <file> is copied into the /tmp directory and you work on this local copy. If you close your editor, the file is saved back to the original location, if you modified it.
  - the default editor is 'vi'
  - you can switch to another editor by setting the environment variable `EDITOR`:
    - `for vim`: `EDITOR='vim'`
• for emacs: EDITOR='emacs' or 'emacs -nw'
• for xemacs: EDITOR='xemacs' or 'xemacs -nw'
• for pico: EDITOR='pico'

Hint: do this setting in the aliensh rc-file $HOME/.alienshrc!

-<file> has an URL-like syntax and can reference an AliEn or local file:
  - “edit file:/tmp/myfile.txt” - edit the local file '/tmp/myfile.txt'
  - “edit myfile.txt” - edit the AliEn file 'myfile.txt' in the CWD
  - “edit alien:/alice/myfile.txt” - edit the AliEn file '/alice/myfile.txt' (the protocol 'alien:' is optional within the shell).
  - “edit alien:/alice/myfile.txt@ALICE::CERN::Tmp” - edit the AliEn file '/alice/myfile.txt'. The file is read preferably from the SE 'ALICE::CERN::Tmp', if this is not possible from another 'closest' SE. The file will be stored back into 'ALICE::CERN::Tmp'.

AliEn files are per default stored back into the same storage element, unless you specified a different one by appending '@<SE-Name>' to <file>.

Every edited AliEn file is stored as a new version in the file catalog. See the 'cp -v' command for information about versioning.

You can only edit files, which are existing. If you want to edit a new empty file use:
  - '-c' : create a new empty file and save it to <file>. If <file> is a local file and was existing, it is overwritten by the new edited file. If <file> is an AliEn file and was existing, it is renamed to a different version and your edit is stored as <file>
  - '-h' : print the usage information for this command

Return Values ($?):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>File has been edited and rewritten, or the '-h' flag</td>
</tr>
<tr>
<td>1</td>
<td>File could not be written back. You get information printed about your temporary file to rescue it by hand.</td>
</tr>
<tr>
<td>246</td>
<td>the local copy of the file could not be removed</td>
</tr>
<tr>
<td>250</td>
<td>an illegal internal operation mode was set (can happen only in 'buggy' installations/setups)</td>
</tr>
<tr>
<td>252</td>
<td>&lt;file&gt; parameter missing</td>
</tr>
</tbody>
</table>

3.3.9 erase <file>

remove physically all replicas of <file> in storage elements and the catalogue entry
3.3.10 **purge <file>|<directory>**

- with `<file>` parameter: removes all previous versions of `<file>` besides the latest
- with `<directory>` parameter: remove all previous versions of all files in directory `<directory>` besides the latest ones!

**Return Values ($?)**:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>File has been erased</td>
</tr>
<tr>
<td>1</td>
<td>File could not be erased</td>
</tr>
<tr>
<td>255</td>
<td>Protocol/Connection error</td>
</tr>
</tbody>
</table>

3.3.11 **whereis [-l] <file>**

- list all replicas of file `<file>`. It includes the GUID, the TURL and the SE name.
  - `-l` : list only the names of the SEs and the GUID
  - `-h` : print the usage information

**Return Values ($?)**:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td>1</td>
<td>Command failed</td>
</tr>
<tr>
<td>255</td>
<td>Protocol/Connection error</td>
</tr>
</tbody>
</table>

**Example:**

- locate a files SEs only:
  ![Example output](image1.png)

- locate a files GUIDs/TURLs and SEs:
  ![Example output](image2.png)
- **for experts:** use the busy box command, to print only one output field of the command:

![Busy Box Command Example](image)

### 3.3.12 mirror <lfn> <se>

If you want to replicate files from one SE into another, you can use the mirror command. <lfn> is the file you want to replicate, <se> is the target storage element.

**Return Values ($?):**

- 0 - Command successful
- 1 - Command failed
- 255 - Protocol/Connection error

### 3.3.13 df [<SE-name>]

- report the disk space usage of the default SE or <SE-name>

**Return Values ($?):**

- 1 - in any case

### 3.3.14 find [-<flags>] <path> <fileName|pattern> [[<tagname>[:<condition>] [ [and|or] [<tagname>[:<condition>]]]]]*

The find command helps to list catalog entries according to certain criteria. The search is always tree oriented following the hierarchical structure of the file catalog (like the UNIX find command).

The simplest find syntax is:

```
find /alice/cern.ch/demo/data % . root
```

![Find Command Example](image)

If you want to use pattern matching for the files you are looking for (in this example all files with .root suffix) you should use the % instead of the * for wild cards as you might be used to. This might change in future versions.

You can switch on the summary of how many files have been found with the '-v' flag:
Depending on your needs you can also select, which information you want to print per entry using the `-p <field1>,<field2>,<field2>...` option. E. g.:

```
/home/peters@pcapiserv01/cern.ch:/tmp/a
/home/peters@pcapiserv01/cern.ch:/tmp/a
/home/peters@pcapiserv01/cern.ch:/tmp/a
/home/peters@pcapiserv01/cern.ch:/tmp/a
/home/peters@pcapiserv01/cern.ch:/tmp/a

If you add the `-r` flag you can additional print fields describing the location of that file:

- longitude
- location
- msd
- domain

In combination with ROOT the flag `-x <collection name>` is very useful since it prints the

Available fields are (only the 'interesting' ones are commented):

- seStringlist
- aclId
- lfn - logical AliEn file name
- dir
- size - size of the file in bytes
- gowner - group
- guid - the GUID of the file
- owner - owner (role)
- ctime
- replicated
- entryId
- expiretime
- selist
- type
- md5 - the MD5 sum of that file
- perm - the file permissions in text format
find result as an XML collection, which can be read directly by ROOT.

You can easily store this in a local file 'find ..... > /tmp/collection.xml'.

There are additional flags and the tagname:condition meta data query parameters. The documentation of these parameters is left out in this document at present.

3.4 Job Management Commands

3.4.1 top [-status <status>] [-user <user>] [-host <exechost>] [-command <commandName>] [-id <queueId>] [-split <origJobId>] [-all] [-all_status]

  ➔ print job status information taken from the AliEn task queue

  - 'status <status>' : print only jobs with status <status>
  - 'user <user>' : print only jobs from user <user>
  - 'host <exechost>' : print only hosts on exechost <host>
  - 'command <command>' : print only tasks executing
  - 'id <queueId>' : print only task with ID <queueId>
  - 'split <masterJobId>' : print only tasks belonging to masterjob
<masterJobId>

- '-all' : print jobs of all users
- '-all_status' : print jobs in any state

Return Values ($?):

anything not 255
- Command has been executed
255 - Protocol/Connection error

3.4.2 ps [.....]

- similar functionality like 'top': report process states
- if the environment variable alien_NOCOLOUR_TERMINAL is defined, all output will be black&white. This is useful, if you want to pipe or parse the output directly in the shell.
- The following options are defined (parameters like <list> are comma separated names or even just a single name):
  - '-F {l}' - "f" = long (output format)
  - '-f ' <flags/status>'
    - e.g. '-f DONE' lists all jobs in status done
  - '-u <userlist>'
    - list jobs of the users from <userlist>. '
      -u %' selects jobs from ALL users!
  - '-s <sitelist>'
    - lists jobs which are or were running in <sitelist> list of sites
  - '-n <nodelist>'
    - lists jobs which are or were running in <nodelist> list of nodes
  - '-m <masterjoblist>'
    - list all subjobs which belong to one of the master jobs given in <masterjoblist>
  - '-o <sortkey>'
    - sort the output with a specific key. The defined keys are:
      "execHost, queueId, maxrsz, cputime, ncpu, executable, user, sent, split, cost, cpufamily, cpu, rsz, name, spyurl, commandArg, runtime, mem, finished, masterjob, status, splitting, cpuspeed, node, error, current, received, validate, command, splitmode, merging, submitHost, vsize, runtimes, path, jdl, procinfoftime, maxvsize, site, started, expires"
' -j <jobidlist>'
  - list all jobs with job ids from <jobidlist>.

' -l <query-limit>'
  - set the maximum number of jobs to query. For a non privileged user the maximum is 2000 by default.

' -q <sql query>'
  - if you are familiar with the table structure of the AliEn task queue, you can specify with this parameter your own SQL database query using the keys mentioned above. If you need a specific query, ask one of the developers for help.

' -X'  - active jobs in extended format

' -A' - select all your owned jobs in any state

' -W' - select all YOUR jobs which are waiting for execution

' -E' - select all YOUR jobs which are in error state

' -a' - select jobs of ALL users

' -jdl <jobid>'
  - display the job jdl of <jobid>.

' -trace <jobid> [trace-tag[,trace-tag]]'
  - display the job trace information. If tags are specified, the trace is only displayed for the corresponding tags.
  - per default, all 'proc' tags are disabled. to see all available trace tags, use 'ps -trace <jobid> all'.
  - the used tags are:
    - “proc” - resource information
    - “state” - job state
    - “error” - error statements
    - “trace” - job actions (downloads etc.)

Return Values ($?):

  0  - Command has been executed
  255  - Wrong command parameters specified

3.4.3 submit [-h] <jdl-file>

  - submits the jdl file <jdl-file> to the AliEn task queue.
  - local jdl files are referenced with “file:<local-file-name>”
files in the AliEn file catalogue are referenced with “<alien-file-name>” or “alien:<alien-file-name>”.

Return Values ($?):

- 0 - Submission successful
- 255 - Submission failed

**Warning**: local JDL files can only be submitted, if they don't exceed a certain size. If you reference thousands of input data files, it is safer to register this JDL file in the file catalogue and submit this one. The proper way for many input data files is to use the InputDataCollection tag and to register an XML input data collection in AliEn as explained later.

### 3.4.4 `kill <job-id>`

- kills a job from the AliEn task queue
- `<job-id>` can be a 'single' or a 'master' job. Killing a 'master' automatically kills all its sub jobs.
- to be able to kill a job, you must have the same user role like the submitter of that job or be a privileged user.

Return Values ($?):

- 0 - Job successfully killed
- 255 - Job couldn't be killed
3.4.5 queue [ info, list, priority .....]

The queue command provides TaskQueue states and parameters to the user.

It accepts three subcommands:

- **queue info** [ <Queue Name> ]
  - prints for one( <Queue Name> ) or all sites the status information and counters of individual job states. Here you can see e.g. if one site, where you want to run your GRID job is currently blocked, closed or in any error state.

- **queue list** [ <Queue Name> ]
  - prints for one( <Queue Name> ) or all sites the status information, configuration and load parameters.

- **queue priority**
  - Prints for one ( <user-name> ) or all users information about their priorities. Priorities are specified by a nominal value of parallel jobs and a maximum number of parallel jobs. Currently the maximum is not enforced. Every new user gets this two parameters set to 20. If the system is not under load, you will be able to run more than 20 jobs in parallel. If the system is loaded in such a way, that all nominal jobs fill up exactly the capacity, you can run exactly 20 jobs in parallel. If the system is over the nominal load, you will run less than 20 jobs in parallel according to a fair share algorithm. If you need a higher job quota, contact the ALICE VO administrators.

- **queue priority jobs** [ <user-name> | % ] [ <max jobs> ]
  - prints the job ranking for all jobs or <user-name>'s jobs. The <max jobs> parameter limits the list, if there are too many jobs in the queue. If you want to see the ranking of the first 10 jobs, which would start next (if an agent could pick them up), do e.g.
    
    ```
    queue priority jobs % 10
    ```

3.5 Package Management

3.5.1 packages

This command lists the available packages defined in the package management system.

Return Values ($?):

- 0 - Command has been executed
3.5.2 Structure and definition of Packages

You select a certain package by specifying in your job description:

```
Packages={ <package name 1> [, <package name 2> ...] }
```

Packages are divided into user and VO packages. The VO packages can be found in the file catalog under `/alice/packages`, while user packages are found in the home directory of users under `$HOME/packages`.

3.5.3 Create an AliEn package from a standard ROOT CVS source

If you want to publish your self-compiled ROOT version:

- cd root/
- make dist
- cd ../
- create a file `*.alienEnvironment`

```bash
# # # # # # # # # # # # # # # # # # # # # #  PackMan Setup File for ROOT # # # # # # # # # # # # # # # # # # # # # #
echo "*** PackMan Setup Start ***"
export ROOTSYS=$1/root
echo "***************************************************************************"
echo "$ROOTSYS set to $ROOTSYS"
export PATH=$ROOTSYS/bin:$PATH
echo "***************************************************************************"
echo "PATH set to $PATH"
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
echo "***************************************************************************"
echo "LD_LIBRARY_PATH set to $LD_LIBRARY_PATH"
echo "***************************************************************************"
echo "*** PackMan Setup End ***"
# The following two lines MUST be there!
shift
$*
```

- unzip the dist file created by ROOT: e.g. `unzip root*.tgz`
- add the `.alienEnvironment` file: `tar rvf root*.tar .alienEnvironemnt`
- zip the ROOT archive file: `gzip root*.tar`
- publish the new package in AliEn as your private version 5.10.0:
  ```bash
  mkdir -p $HOME/packages/ROOT/5.10.0/
  ```
4 The ROOT AliEn Interface

4.1 Installation of ROOT with AliEn support

This document proposes three different ways to install ROOT. If you are developing within the ROOT framework and you need to modify ROOT itself, it is recommended to follow 4.1.1. If you don’t need to develop parts of ROOT, you can use 4.1.2 (which recompiles ROOT on your machine) or 4.1.3 (which installs a precompiled binary).

4.1.1 Manual Source Installation from CVS

- Login to the ROOT CVS server with 'cvs' as password:
  
  `cvs -d :pserver:cvs@root.cern.ch:/user/cvs login`

- Checkout the ROOT source code
  
  - the CVS Head:
    
    `cvs -d :pserver:cvs@root.cern.ch:/user/cvs co`
  
  - a tagged version (≥ v5-10-00):
    
    `cvs -d :pserver:cvs@root.cern.ch:/user/cvs -r v5-10-00 co`

- The AliEn module in ROOT needs GLOBUS to be enabled
  
  - you need to set the environment variable GLOBUS_LOCATION f.e. the version installed by the AliEn installer is referenced by:

    `export GLOBUS_LOCATION=/opt/alien/globus` (or `setenv`)

  - Run the configure script enabling AliEn:

    `./configure --enable-alien`

    - the script will look in the default location of the API package
      
      `/opt/alien/api` . If the API is installed in another location, you can specify this via the `--with-alien-incdir=<>` and `--with-alien-libdir=<>` switches. F.e. if you have the API installed under `$HOME/api`, execute:

      `./configure --enable-alien
      --with-alien-incdir=$HOME/include
      --with-alien-libdir=$HOME/lib`

      - Compile ROOT

      `make`

For all questions concerning the ROOT installation in general, you can consult the ROOT web page [http://root.cern.ch](http://root.cern.ch)
4.1.2 Source installation using AliEnBits

- Follow the instructions of 2.1.3 how to install the AliEnBits framework until (including) the 'configure' statement. Change into the ROOT application directory:

   cd $HOME/alienbits/apps/alien/root

- Start the Compilation

   make

   ➔ If you installed previously the API via AliEnBits, the ROOT configuration and compilation will start immediately. If not, the AliEnBits system will start to download all dependency packages and compile them beforehand.

   Hint: the AliEnBits system will install the ROOT version defined in the build system for the AliEn release you are using. It is defined as the entry 'GARVERSION' in $HOME/alienbits/apps/alien/root/root/Makefile. You cannot easily switch to another CVS tag with this procedure.

4.1.3 Binary Installation using the AliEn installer

Follow 2.1.1 but select 'ROOT' as the package to be installed. If you selected the default location '/opt/alien/' in the installer, you will find ROOT installed under '/opt/alien/root'.

4.2 ROOT Startup with AliEn support - a quick test

To validate your installation, repeat the following quick test:

- Use the alien-proxy-init command to retrieve a shell token from an API service (see chapter 2.4).
- It is convenient to write a small startup script for the ROOT startup to avoid environment setting:

```bash
#!/bin/bash
test -z $ROOTSYS && export ROOTSYS=/opt/alien/root
export PATH=$ROOTSYS/bin:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH:/opt/alien/api/lib
if [ -e /tmp/gclient_env_$UID ]; then
    source /tmp/gclient_env_$UID;
    root $*
fi
```
If you got the ROOT prompt, execute

```cpp
TGrid::Connect("alien://", 0, 0,"t");
```

this uses your already established session token and prints the 'message of the day (MOTD)' on your screen. This method is described more in detail in the following subsections.

If you got here, everything is properly configured.

4.3 The ROOT TGrid/TAlien module

All ROOT support for AliEn is based on the classes TAlien and TAlienFile.

ROOT provides a virtual base class for GRID interfaces encapsulated in the TGrid class. TAlien is a plugin implementation for the TGrid base class and is loaded, if you specify in the static factory function of TGrid 'alien:' as the protocol to load.

F. e. the factory function for TGrid is the 'Connect' method:

```cpp
TGrid* alien = TGrid::Connect("alien://");
```

this triggers the loading of the AliEn plugin module and returns an instance of TAlien

TFile is the base class for all file protocols. A TAlienFile is created in the same manner via the static factory function 'Open' of TFile:

```cpp
TFile* alienfile = TFile::Open("alien://...");
```

The following sections highlight the most important aspects of the interface and are not meant to be complete. For details see the inlined ROOT documentation in the source
code, which is located in the 'root/alien' directory of the source installation.

**Hint:** Examples in this chapter are always complete, which means they start always with a `Connect` statement, which you further have to execute only once in a ROOT session!

### 4.3.1 TGrid::Connect - Authentication and Session Creation

The first thing you have to do (see the quick test in 4.2), to get access to AliEn data management and job management functionalities, is to authenticate to an API service or to use an already existing session token.

As described in 2.3.1 we can store a session token within the ROOT application or we share a session token, which was established outside the application to be used by aliens.

```cpp
//--- Load desired plugin and setup connection to GRID
static TGrid *Connect (const char *grid, const char *uid = 0,
const char *pw = 0, const char *options = 0);

Syntax 1: TGrid::Connect
```

Consider these example statements to initiate connections:

- **Connect creating a memory token to a default API service:**
  ```cpp
tGrid::Connect(“alien://”);
  ```
- **Connect creating a memory token to a user specified API service:**
  ```cpp
tGrid::Connect(“alien://myhost.cern.ch:9000”);
  ```
- **Connect creating a memory token with a certain role:**
  ```cpp
tGrid::Connect(“alien://myhost.cern.ch:9000,”aliprod”);
  ```
- **Connect using the existing file token:**
  ```cpp
tGrid::Connect(“alien://”,0,0,”t”);
  ```

**Hint:** the first example should be the case in 99% of all use cases. If you use programs which forks, you should always use the file based token mechanism or call in every forked process the `Connect` method again. If you are using threads, you must lock the `Command` statements later on with a Mutex.

**Return Values:**

- (TGrid*) 0 - Connect failed
- (TGrid*) != 0 - Connection established

ROOT sets automatically the global variable `gGrid` with the result of the `TGrid::Connect` call. If you deal only with one GRID connection, you can just use that one to call any of the TAlien methods f.e. `gGrid->Ls()` , but not `TAlien::Ls()`!

**For Experts:** as you might have guessed, `TGrid::Connect(“alien://”)` is equivalent to the call `new TAlien(...)`, which bypasses the plugin mechanism of
4.3.2 TAlien::Command - arbitrary command execution

You can execute every command seen in aliensh with the Command method (besides the cp command).

```cpp
TGridResult *Command(const char *command, bool interactive = kFALSE, UInt_t stream = kOUTPUT);
```

Syntax 2: TAlien::Command

As you have seen already in section 3.2.4 discussing the gbbox busy box command the API protocol returns four streams. The stream to be stored in a TGridResult is by default the result structure of each command (TAlien::kOUTPUT). Another stream can be selected with the stream parameter:

- for STDOUT:
  ```
  stream = TAlien::kSTDOUT
  ```

- for STDERR:
  ```
  stream = TAlien::kSTDERR
  ```

- for the result structure:
  ```
  stream = TAlien::kOUTPUT
  ```

- for the misc. hash:
  ```
  stream = TAlien::kENVIR
  ```

**Hint:** You need to add `#include <TAlien.h>` to your ROOT code, to have the stream definitions available!

If you set `interactive=kTRUE` the STDOUT+STDERR stream of the command will be printed automatically to your terminal STDOUT+STDERR.

The result structure and examples for the Command method are explained in the next section.

Continue reading until 4.3.4 and try then the example given.

4.3.3 TAlienResult - the result structure of TAlien::Command

TAlienResult is the plug-in implementation of a TGridResult returned by TAlien::Command. A TAlienResult is based on a TList which contains a TMap as list entries - to make it easy: it is a list of key-value pairs! If you want to find out names of returned key names, just use the TAlienResult::Print("all") method or use the 'gbbox -d' command (see 3.2.4).
4.3.4 TAlien::Ls - Example of Directory Listing in the File Catalogue

Consider this example, which executes the 'ls -la' function, dumps the complete result and finally loops over the result and prints all file names in the /alice directory:

```cpp
TGrid::Connect("alien://");
TGridResult* result = gGrid->Command("ls -la/alice",0,TAlien::kOUTPUT);
result->Print("all");
while (result->getKey(i,"name"))
    printf("Listing file name: %s\n", result->getKey(i++, "name"));
```

The keys defined for an 'ls' operation are currently:

- `ls -la` → “name, md5, size, group, path, permissions, user, date"
- `ls -m` → “md5, path” (here path is the full path name)
- `ls -b` → “guid, path” (here path is the full path name)
- `ls` → “name, path”

Since this is a very common use case, there are convenience functions defined to simplify the syntax of the listing example:

```cpp
TGrid::Connect("alien://");
TGridResult* result = gGrid->Ls("/alice");
while (result->GetFileName(i))
    printf("Listing file name: %s\n", result->GetFileName(i++));
```

4.3.5 TAlien::Cd + TAlien::Pwd - Example how to navigate the CWD

The CWD allows you to reference files without absolute path names. The CWD is by default (if you don't use a file token, where the CWD is shared and stored on disk) your home directory after connecting.

To see the current working directory use the `pwd` command:

```cpp
TGrid::Connect("alien://");
printf("Working Directory is %s\n", gGrid->Pwd());
```

It returns a `const char*` to your current working directory name.

To navigate the CWD use the `cd` command:

```cpp
TGrid::Connect("alien://");
Bool result = gGrid->Cd("/alice");
```
The `cd` command returns `kTRUE` if it was successful, otherwise `kFALSE` and the CWD stays unchanged.

### 4.3.6 TAlien::Mkdir - Example how to create a directory

```cpp
Bool_t TAlien::Mkdir(const char* ldn, Option_t* options, Bool_t verbose)
```

- `ldn` specifies the logical directory name you want to create f.e. `/alice/cern.ch/mydirectory`
- `options` are flags for the command (see 3.2.7).
- `verbose=kTRUE` switches on verbosity of the command

```cpp
TGrid::Connect("alien://");
Bool result = gGrid->Mkdir("mydirectory");
```

**Return Values**:

- `kTRUE` - Directory created
- `kFALSE` - Directory creation failed

### 4.3.7 TAlien::Rmdir - Example how to remove a directory

```cpp
Bool_t TAlien::Rmdir(const char* ldn, Option_t* options, Bool_t verbose)
```

- `ldn` specifies the logical directory name you want to remove f.e. `/alice/cern.ch/mydirectory`
- `options` are flags for the command (see 3.2.7).
- `verbose=kTRUE` switches on verbosity of the command

```cpp
TGrid::Connect("alien://");
Bool result = gGrid->Rmdir("mydirectory");
```

**Return Values**:

- `kTRUE` - Directory removed
4.3.8 TAlien::Rm - Example how to remove a file entry

Boo_t TAlien::Rm(const char* lfn, Option_t* options, Boo_t verbose)

- lfn specifies the logical file name you want to remove f.e. "/alice/cern.ch/myfile"
- options are flags for the command (see 3.2.7).
- verbose=kTRUE switches on verbosity of the command

TGrid::Connect("alien://");
Bool result = gGrid->Rm("myfile");

Return Values:

- kTRUE - File entry removed
- kFALSE - File entry deletion failed

Hint: as said previously - this function removes only file catalogue entries, no physical files

4.3.9 TAlien::Query - Querying files in the Catalogue

The query function is very convenient way to produce a list of files, which can be converted later on into a ROOT TChain (f.e. to run a selector on your local machine or on a PROOF cluster).

virtual TGridResult *Query(const char *path, const char *pattern,
const char *conditions = "", const char *options = "");

The syntax is straight forward:

- path specifies the node (directory) from where to start searching in the directory tree
- pattern specifies a pattern to be matched in the full filename f.e.
  - "*.root" matches all files with 'root' suffix
  - "*" matches all files
  - 'galice.root' matches exact file names
- conditions are conditions on meta data for the queried files.
- options are options to the query command (see )
ROOT has a plug-in mechanism for various file access protocols. These plug-ins are called via the static TFile::Open function. The protocol specified in the URL refers to the appropriate plug-in. TAlienFile is the implemented plug-in class for file registered in AliEn. Transfers are done using internally the TXNetFile class (xrootd). To reference a logical file in AliEn, use the 'alien://' protocol or add '/alien' in front of the logical file name space:

- TFile::Open("alien:///alice/cern.ch/demo/data/0115.00108/miniesd.root")
  - opens an AliEn file in READ mode.
- TFile::Open("/alien/alice/cern.ch/demo/data/0115.00108/miniesd.root");
  - is equivalent to the first statement.
- TFile::Open("alien:///alice/cern.ch/user/t/test/myfile.root@ALICE::CERN:::se01","RECREATE")
  - opens an AliEn file in WRITE mode using the file versioning system at the storage element 'ALICE::CERN:::se01'.

### 4.3.10 File Access using TFile - TAlienFile

ROOT has a plug-in mechanism for various file access protocols. These plug-ins are called via the static TFile::Open function. The protocol specified in the URL refers to the appropriate plug-in. TAlienFile is the implemented plug-in class for file registered in AliEn. Transfers are done using internally the TXNetFile class (xrootd). To reference a logical file in AliEn, use the 'alien://' protocol or add '/alien' in front of the logical file name space:

- TFile::Open("alien:///alice/cern.ch/demo/data/0115.00108/miniesd.root")
  - opens an AliEn file in READ mode.
- TFile::Open("/alien/alice/cern.ch/demo/data/0115.00108/miniesd.root");
  - is equivalent to the first statement.
- TFile::Open("alien:///alice/cern.ch/user/t/test/myfile.root@ALICE::CERN:::se01","RECREATE")
  - opens an AliEn file in WRITE mode using the file versioning system at the storage element 'ALICE::CERN:::se01'.

### 4.3.11 File Copy operations in ROOT

The class TFileMerger implements a copy function to copy from 'arbitrary' source to destination URLs. Instead of using aliensh commands, you can copy a file within ROOT code following this example:

```c
TFileMerger m;
m.Cp("alien:///alice/cern.ch/demo/data/0115.00108/miniesd.root","file:/tmp/miniesd.root");
```

This works also to copy local to local files or AliEn to AliEn files.
5 Using AliEn for Analysis

5.1 First Steps with Batch Analysis
In the following section we are showing you, how to run your own batch analysis with AliEn.

5.1.1 Analysis Example running on your local laptop with local data
We start with a very simple example, which does not use selectors yet:

We are trying to write an analysis program for already existing ROOT files registered in the AliEn File Catalogue. As the first step, we are downloading some example data to properly test our analysis program on the local machine. In the second step we are running the same analysis code distributed with AliEn.

You can find the example data in the file catalogue in the directory

'/alice/cern.ch/demo/example1/

First you are creating a collection file for this data using aliensh on your machine:

```
find -x example1 /alice/cern.ch/data/ *.root > /tmp/example1.xml
```

This collection file references directly the data in the AliEn storage. Now, we want to download this data to our computer (create the local directory /tmp/example1), to run our first test locally using aliensh:

```
cp /alice/cern.ch/demo/data/*.*.root file:/tmp/example1/
```

This should like this:

Make a copy of your collection file and change in the copy the TURL entries to local URLs. Any local file must be specified like 'file:/tmp/example1/00115.00108.minesd.root'.
Now we are writing a simple analysis program, which loops over all the files on our local disk:

```c
// ############################################################################
// #### to execute this macro do:       #
// #### > alien-token-init <role>  #
// #### > source /tmp/gclient_env_$UID #
// #### > root -x example1.C\("/tmp/example1.xml\")
// #### or exchange the TGrid::Connect #
// #### line which is now commented #
// #### and do directly #
// #### > root -x example1.C\("/tmp/example1.xml\")
// ############################################################################

void example1(const char* collectionfile){
  gSystem.Load("libVMC.so");
  // gSystem.Load('libESD.so");
  TGrid::Connect("alien:///");
  // or TGrid::Connect("alien://pcapiserv01:9000","<role>");
  TAlienCollection* mycollection = new TAlienCollection(collectionfile);
  if (!mycollection) exit(-1);
  mycollection->Reset();
  while (mycollection->Next()) {
    printf("Looping over File %s\n",mycollection->GetURL(""));
    TFile* analysisfile = mycollection->OpenFile("");
    analysisfile->ls();
    delete analysisfile;
  }
}
```

To start it, follow the instructions in the header of that program. Since our files are written in the ESD format, we would need to load `libESD.so` to have the appropriate classes for reading available, but for this generic example we skip this part and discuss it in a section later, where we show how to use PAR archives and selectors.
5.1.2 Analysis Example running on your local laptop with data in AliEn storage elements

We can run the same analysis program directly with AliEn files using the collection file we have produced initially by:

```bash
find -x example1 /alice/cern.ch/data/ *.root > /tmp/example1.xml
```

This time the files to analyze will be read remotely, since they are specified with file URLs like `alien://<path>` in the collection file. The analysis code still runs on your computer.

5.1.3 Analysis Example running batch jobs with AliEn

The last step is to run the same analysis in the AliEn environment.

The biggest part of the work we have done in the example before will now be done directly by AliEn. We don’t need anymore to create an XML file with the collection to be analyzed. We have to specify our input data in the **InputData** section of the job description file, which we call in the following JDL file. AliEn will write into the Job sandbox this XML file, if we request it in the JDL. Moreover we have to copy our example macro in one of the AliEn storage elements and create a small script, which will execute ROOT with the right parameters.

We create the file `/tmp/example1.run`, which looks like:

```bash
#!/bin/bash
root -b -x example1("example1.xml");
```

- **Start aliensh**
  - # go to your home directory
    cd
  - # create a directory 'analysis'
    mkdir analysis
  - # copy your analysis example macro into the new directory
    cp file:/tmp/example1.C analysis/@ALICE::CERN::se01
  - # create a 'bin' directory, if you don't have it already
    mkdir bin
  - # copy your executable into the 'bin' directory
    cp file:/tmp/example1.run $HOME/bin/@ALICE::CERN::se01
  - # leave the shell
    exit
Create a file /tmp/example1.jdl

Executable="example1.run"
InputFiles= {"LF:<your home directory>/analysis/example1.C"};
InputData= ["LF:/alice/cern.ch/demo/data/00115.00108.miniesd.root,nodownload",
            "LF:/alice/cern.ch/demo/data/00115.00156.miniesd.root,nodownload",
            "LF:/alice/cern.ch/demo/data/00115.00229.miniesd.root,nodownload"];
InputDataList="example1.xml"
InputDataListFormat="xml-single"
Packages="{"ROOT::5.10.01", "APISCONFIG::V1.1"};
Requirements = ( member(other.CE,"ALICE::CERN::LSF") );

(if you don't know your home directory do 'aliensh echo $HOME')

The JDL tags used:

- **Executable**: any file, which can be found under /bin/, /alice/bin, or <HOME>/bin
- **InputFiles**: this are files, which are copied into the Job sandbox, they don't need to be in the storage element, where your job is executed
- **InputData**: this are files, which are copied into your input sandbox unless you add "nodownload" to the URL: in our example we don't want the files to be staged into the sandbox, we want to open them within our analysis program.
- **InputDataList**: this is the filename in which AliEn writes the InputData list. The format of this file is specified in InputDataListFormat.
- **InputDataListFormat**: this is the list format of the InputData list. 'xml-single' means, that every file is equivalent to one event. If you specify 'xml-group' a new event starts, every time the first base filename appears again, f.e.

  "LF: ..../01/galice.root", ← 1st event
  "LF: ..../01/Kinematics.root",
  "LF: ..../02/galice.root", ← 2nd event
  "LF: ..../02/Kinematics.root",
  ......

- **Packages**: our analysis program starts up in 'example1.run' the root command. We have to tell AliEn to configure a certain ROOT version for our GRID job. This is done via the Packages key in the JDL. To have an automatically configured API service connection, we can require the package 'APISCONFIG' as a package, which does the necessary setup, to connect without an URL 'alien://' in the TGrid::Connect statement. You can use the 'packages' aliensh command to see the latest version of this package.
● Copy the JDL to AliEn:
  ➜ mkdir $HOME/JDL
  ➜ cp file:/tmp/example1.jdl $HOME/JDL/@ALICE::CERN::se01

● Submit your 1st analysis job:
  ➜ submit $HOME/JDL/example1.jdl

  • if it worked, the system will print your job ID, otherwise it will dump you your JDL to check for syntax errors.

● Now check from time to time the status:
  ➜ ps

  or
  ➜ top

● You can trace your job:
  ➜ ps -trace <job ID> all

● If your job turns into an error state, you should check with the trace command, what went wrong

● If your job is finally 'D' or 'DONE,
you should find the output (in this case we can find only STDOUT/STDERR because our job does not write any new file) under (in aliensh!)

  /proc/<username>/<job ID>/job-output/

This directory is called the output sandbox of a job!

  ➜ list the contents with 'ls /proc/.....'

  ➜ check the STDOUT and STDERR of your job:

  • 'cat /proc/...../job-output/stdout'
  • 'cat /proc/...../job-output/stderr'


5.1.4 Storing Outputfiles in AliEn Batch Jobs

If you now modify the analysis program you have created in the previous section to produce f. e. a histogram file “histograms.root”, we can add a new tag to our JDL file to store back this output file into our job output sandbox:

  ➜ OutputFile="histogram.root"

For analysis it is very convenient to store all the histogram outputfiles in ONE storage element, which is close to you and disk based, f. e.

  ➜ OutputFile="histogram.root@ALICE::CERN::se01,noarchive";

The 'norarchive' option is needed, to avoid, that the ROOT file will be packed together with
other files in an archive file.

5.1.5 Split your AliEn batch jobs

If you analyze many files and these files are distributed over several storage systems it is useful to split your job.

You can split your job in a way, that every subjob processes only some of the input data files.

To split your analysis job just add one line to your JDL:

```plaintext
split="se"
```

The result of this tag is, that if you have f.e. 5 files at CERN, 5 at FZK, 5 at CNAF, you will get ONE job in each mass storage domain.

This is not yet satisfactory, because if all your files are at CERN, you gain nothing. To be more specific, you can refine this splitting per mass storage domain.

You can set the maximum input data volume in MB per single job with:

```plaintext
SplitMaxInputFileSize="<x [Mb]>"
```

You can also limit the maximum number of files for a single job with

```plaintext
SplitMaxInputFileNumber="<n>"
```

To check that, add to the JDL from the very simple example the two lines:

```plaintext
split="se"
SplitMaxInputFileNumber="1"
```

- Submit this 'master job'
- Your master job should be split into one sub job for each file.

While not all sub jobs are terminated, the output of each job is found like for a standard job under
if all sub jobs have finished, the output of the sub jobs is moved automatically to

```
/proc/<username>/job-output/
```

Hint: if you want to see only the sub jobs of a certain master job, you can use the `ps` command as shown here:

The '-m' flag specifies the ID of the master job, the '-Fl' selects the long output format, where you can see all the details, where your jobs are/have been running.

## 5.2 Running ROOT Selectors

### 5.2.1 ROOT Selectors

Selectors are the ROOT way of writing analysis programs for data stored in ROOT trees. They allow you to put your analysis code into an automatic event loop. You can run selectors 'by hand' on your laptop or you run exactly the same code using PROOF with the gain of interactivity and parallelism. Have a look at [ftp://root.cern.ch/root/doc/chapter22.pdf](ftp://root.cern.ch/root/doc/chapter22.pdf) to get some more insight into selectors.

### 5.2.2 Create a Selector

To create a default selector, download the AliESD PAR archive and one ESD file to your local machine. To compile the ESD library you need to have a correct ROOT environment setup (ROOTSYS/PATH/LD_LIBRARY_PATH).

Now follow this example in your normal shell:
ROOT has generated two files 'esdTree.h' and 'esdTree.C' for you. These are the selector files, where you implement your analysis code. Try to read the comments for each of the functions found in 'esdTree.C' to understand the structure. The following methods are defined in this file:

- **Begin()**
  - called everytime a loop on the tree starts, a convenient place to create your histograms.

- **SlaveBegin()**
  - called after Begin(), when on PROOF called only on the slave servers.

- **Process()**
  -
called for each event, in this function you decide what to read and fill your histograms.

**SlaveTerminate()**
- called at the end of the loop on the tree, when on PROOF
- called only on the slave servers.

**Terminate()**
- called at the end of the loop on the tree,
- a convenient place to draw/fit your histograms.

For a first test, implement the following Process() method:

```cpp
Bool_t esdTree::Process(Long64_t entry)
{
    printf("Processing Entry: %ld\n",entry);
    return kTRUE;
}
```

To run this selector, you do the following:

```bash
# setup ESD package
root [0] .x AliESD/PROOF-INF/SETUP.C
# open ESD file
root [1] TFile::Open("/tmp/esd-analysis/0115.00108/miniesd.root");
# get the ESD tree out of the file
root [2] TTree* esdtree = (TTree*)gFile->Get("esdTree");
# run the selector
Processing Entry: 0
```

As we see, our ESD tree contained only one single event!

### 5.2.3 An advanced example of an ESD analysis selector

To run now a serious analysis we need to implement histogram booking, analysis and cuts on physics parameters and the storing and displaying of the histograms. Download an example of an ESD tree analysis provided thanks to Peter.Hristov@cern.ch.

```bash
/opt/alien/bin/aliensh cp /alice/cern.ch/demomacro/esdTree.C file:/tmp/esd-analysis/
/opt/alien/bin/aliensh cp /alice/cern.ch/demomacro/esdTree.h file:/tmp/esd-analysis/
```

Now run the selector as you learned in the previous example.
The selector will open four Canvases in the Terminate() function on your screen with the K0 candidates, lambda- and anti-lambda masses.

5.2.4 Run selectors as AliEn batch jobs
To run a selector in AliEn you have to provide the following files:

- an executable to run
  - this must be located in the file catalog in
  - /bin
  - /alice/bin
  - $HOME/bin
- a PAR file (if needed)
- the standard package and selector startup macro
  - batchSelector.C:
    - you can find this file under /alice/macros
- your personal selector files
  - esdTree.C and esdTree.h

To have an easy procedure to submit analysis jobs for ESD files, there is a prepared aliensh script, which copies your analysis code into AliEn and submits a single job for it. Try to follow this example aliensh script:
Run the example analysis in a single job by executing in `aliensh`:

`'run /alice/cern.ch/demo/scripts/EsdAnaSingle.alish'`

Run the example analysis as a split job by executing in `aliensh`:

`'run /alice/cern.ch/demo/scripts/EsdAnaMulti.alish'`

**What do these scripts do?**

- copy your local selector files into your AliEn directory `$HOME/macros`
  - file: `/tmp/esd-analysis/esdTree.*`
- copy the execution program from the demo directory into your `$HOME/bin`
  - `/alice/cern.ch/demo/bin/batchSelector.run`
- copy the JDL file into your `$HOME/jdl`
  - single: `/alice/cern.ch/demo/jdl/example2.jdl`
  - multi: `/alice/cern.ch/demo/jdl/example3.jdl`
- copies the AliESD par archive into your `$HOME/par`
- submit the JDLs from `$HOME/jdl` with parameters,
  - where to find you selector file
- which PAR archive to add
- which Tree name to use

**How do I follow my jobs and get the output?**

Try the following commands to see the status of your job. The `<job id>` was printed when you executed the `aliensh` script:

- 'ps'
  - list all your jobs
- 'ps -l'
  - list all your jobs in extended format
- 'ps -j <job id>,'
  - list only the job specified by <job id> - this displays only single jobs
- 'ps -m <master job id>,'
  - list only the sub jobs of master job <master job id>
- 'ps -trace <job id | master job id> all'
  - tracks all steps in the lifetime of your job

**Common Observations:**

- “My job stays a long time in status INSERTING (I)”
  - Check with “ps -f INSERTING” how many other jobs have to be inserted before you. The printout order is insert order! You might have to wait
  - Some central service is down - you cannot change that

- “My job stays a long time in status WAITING (W)”
  - Check with
    - the ranking of your job
      - “queue priority list”
      - and
      - “queue priority jobs | grep $alien_API_USER”
    - the ranking of your job. The job on rank one is the first to be started aso.

- “My job stays a long time in status SAVED (SVD)”
  - Every job, which is in status SAVED has to be processed by a central optimizer, which inserts the job output in the file catalogue. Check with “ps -f SAVED” how many other jobs have to be processed before you: The printout order is processing order! You might have to wait for others to be processed.

- “My jobs ended up in ERROR_SAVING (ESV)”
  - most likely the SE where your output files have to be registered is not functional or misconfigured. Inform or ask one of the VO administrators. Look at the error reported in the tacking of the job 'ps -trace <job id>'.
Check the output:

- The output files are under '/proc/<job id>/job-output/' for sub jobs, if not all jobs are in a final state. When all jobs reached a final state, the output is moved to '/proc/<master job id>/subjobs/<job id>/job-output.

Hint: Since both locations have only a finite lifetime, it might be useful to change the output directory for the master job into your AliEn home directory. This you can enforce with the extra JDL tag 'OutputDir="<your desired output dir>"'. Anyway you need write permissions in that directory!

- copy or cat the output files

  "But the command hangs forever!"

  This is still a common observation. The reason is mostly the same: a SE is unavailable due to misconfiguration or service interruption. Try to take the file from an alternative location (if existing) appending the '@<se name>' syntax. If not, contact one of the VO administrators for help.

5.2.5 Storing and Merging Your Histograms

It is always recommended to store your analysis/histogram output files into a SINGLE storage element and preferably a disk based SE, i.e. you should always write in your JDL something like:

```
OutputFile={"plots.root@ALICE::CERN::se01,noarchive"};
```

Ask a VO administrator for the 'best' storage element to be used.

You can merge automatically files using the TGrid:Query method and the TFileMerge class. To understand the mechanism execute the following macro 'root -x alienmerge.C'. You can find it in the file catalogue under '/alice/cern.ch/demo/macros/alienmerge.C'.
The Loading of libVMC is only needed, because in your little example we are merging indeed ESD trees, not histograms. The merging works transparently for histograms and trees and all objects, which define a Merge function.

6 Run Test Productions with AliEn

To start with a test production, the easiest is to follow an example which you find in the AliEn catalog under `/alice/cern.ch/demo/production/flow.jdl`:
This JDL confronts you with a lot of new JDL syntax, which is explained in the Appendix. It will produce 10 simulated events (no. 1-10) with the given AliRoot version. To submit it, you have to give the run number to be used as an argument in the submission:

- submit flow.jdl 001
  - to submit simulation of run '001'

Run and Event number are used as the SEED for the random generator to be able to repeat the simulation with the same random generator initialization.
7 Appendix JDL Syntax

Every JDL tag follows the syntax

- for single values:
  
  \[
  \text{<tag-name> = "<value>"};
  \]
  
  or
  
  \[
  \text{<tag-name> = {"<value>"};}
  \]

- for a value list:
  
  \[
  \text{<tag-name> = { "<val1>" , "<val2>" , "<val3>" [ ... "<valN>" ]};}
  \]

7.1 JDL Tags

- **Executable**
  
  This is the only compulsory field in the jdl. It gives the name of the Lfn that will be executed. It has to be a file in either /bin or /<VO>/bin or /<HOME>/bin

- **Arguments**
  
  These will be passed to the executable

- **Packages**
  
  This constrains the execution of the job to be done in a site where the package is installed. You can also require a specific version of a package. For example, you can put Packages="AliRoot", and it will require the current version of AliRoot, or Packages="AliRoot::3.07.02"

- **InputFile**
  
  These files will be transported to the node where the job will be executed. They can be either lfn or pf

- **InputData**
  
  It will require that the job is executed in a site close to the files specified here. You can specify patterns, like "LF:/alice/simulation/2003-02/V3.09.06/00143/*tpc.tracks.root", and then all the LFN that satisfy this pattern will be included.
  
  If you don't want the files to be staged into the sandbox (typical for Analysis), you can specify "LF:/alice/....../file.root,nodownload".
  
  use this tag only for few files. If you have to specify many files, use InputDataCollection
- **InputDataList**
  - this is the filename in which the Job Agent writes the InputData list. The format of this file is specified in **InputDataListFormat**.

- **InputDataListFormat**
  - this is the list format of the InputData list. Possible formats are:
    - “xml-single”
    - “xml-group”

  'xml-single' means, that every file is equivalent to one event. If you specify 'xml-group' a new event starts, every time the first base filename appears again, f.e.

  “LF: ..../01/galice.root”, ← 1st event
  “LF: ..../01/Kinematics.root”,
  “LF: ..../02/galice.root”, ← 2nd event
  “LF: ..../02/Kinematics.root”,
  ......

- **InputDataCollection**
  - this is a file which contains the InputData in the XML list format
    - you can produce such a file using the find command f.e.

    ```
    find -x example1 /alice/cern.ch/data/ *.root > /tmp/example1.xml
    ```

  - afterwards you copy the file into AliEn using the `cp` command.
  - you should use this mechanism, if you have many input files
    - the submission is much faster
    - it is better for the job optimizer services
    - you don't need to specify the **InputData** field

- **OutputFile**
  - The files that will be registered in the catalogue when the job finishes. You can specify the storage element by adding “@<SE-Name>” to the file name.

  ```
  OutputFile="histogram.root@ALICE::CERN::se01"
  ```

  Example:

- **OutputArchive**
  - Here you can define, which output files are archived in ZIP archives. Per default AliEn puts all OutputFiles together in ONE archive. Example:

  ```
  OutputArchive =
  {
    "root_archive.zip:*root@Alice::CERN::Castor2",
    "log_archive:*log,stdout,stderr@Alice::CERN::se01"
  };
  ```
This writes two archives: one with all the log files + STDOUT + STDERR stored in the SE ALICE::CERN::se01, another archive with ROOT files, which are stored in SE ALICE::CERN::Castor2.

- **Validation command:**
  - This specifies the script to be executed as a validation script. If the return value of that script is !=0 the job will terminate with status ERROR_V, otherwise SAVED→DONE.

- **Email**
  - If you want to receive an email when the job finishes, you can specify your email address hear. This does not yet work for master jobs.

- **TTL**
  - Here you specify the maximum run time for your job. The system takes care to run your job on a worker node which provides the requested run time for jobs.

- **Split:**
  - If you want to split your job in several sub jobs, you can define the method to split the jobs. Valid are:
    - `file`: there will be one sub job per file in the InputData section
    - `directory`: all the files of one directory will be analyzed in the same sub job.
    - `event`: all the files with the same name of the last subdirectory will be analyzed in the same sub job
    - `userdefined`: Check the field SplitDefinitions
    - `production:(<#start>-<#end>)`: this kind of split does not require any InputData. It will submit the same JDL several times (from #start to #finish). You can reference this counter in SplitArguments using “#alien_counter#”

- **Split Arguments**
  - Here you can define the arguments field for each sub job. If you want f.e. to give the sub jobs counter produced by the Split="production:1-10" tag, you can write f.e. something like
    ```
    SplitArguments = "simrun.C --run 1 --event #alien_counter#";
    ```
  - If you define more than one value each sub job will be submitted as many times as there are items in this array, and the sub jobs will have the element in the array as arguments.

- **SplitMaxInputFileNumber**
  - Defines the maximum number of files that are in each of the subjobs. For instance, if you split per SE, but putting 'SplitMaxInputFileNumber=10', you will be sure that no subjob will have more than ten input data.
- **SplitMaxInputFileSize**
  - Similar to the previous one, but puts the limit in the size of the file. The size has to be given in bytes.

- **SplitDefinitions**
  - This is a list of JDLs. If the user defines them, AliEn will take those jdl's as the subjobs, and all of them would behave as if they were subjobs of the original job (for instance, if the original jobs gets killed, all of them will get killed, and once all of the subjobs finish, their output will be copied to the master job).

8 Appendix Job Status

The following flow chart shows the job status transitions after you have successfully submitted a job. It will help you to understand the meaning of the various error conditions.

8.1 Status Flow Diagram
8.2 Non-Error Status Explanation

- **INSERTING (I)**
  - The job is waiting to be processed by the Job Optimizer. If this is a split job, the Optimizer will produce many sub jobs out of your master job. If this is a plain job, the Optimizer will prepare the input sandbox for this job.

- **WAITING (W)**
  - The job is waiting to be picked up by any Job Agent, that can fulfill the job requirements

- **ASSIGNED (A)**
  - A Job Agent has matched you job and is going to pick it up

- **STARTED (ST)**
  - A Job Agent is now preparing the input sandbox downloading the specified input files.

- **RUNNING (R)**
  - Your executable is finally started and running

- **SAVING (SV)**
  - Your executable has finishehd and the agent saves your output to the specified storage elements.

- **SAVED (SVD)**
  - The agent has successfully saved all the output files. They are still not visible in the catalog.

- **DONE (D)**
  - A central Job Optimizer has registered the output of your job in the catalog.

8.3 Error Status Explanation

- **ERROR_I (EI) - ERROR_A (EA)**
  - this errors are normally not based on a 'bad' user job, but due to service failures

- **ERROR_IB (EIB)**
  - this is a common error during the download phase of needed input files into the sandbox. Usually it's origin is, that a certain input file is not existing in the assumed storage element or the storage element is not reachable from the job worker node.

- **ERROR_V (EV)**
  - the validation procedure failed i.e. your validation script (which you have specified in the JDL) exited with a value !=0.

- **ERROR_SV (ESV)**
  - one or several of the output files could not be saved as it was requested in the JDL. Probably one of the storage elements required was not available

---

1 The 'ps' command prints the job status with this abbreviations
- **ZOMBIE/EXPIRED (Z/EXP)**

  - Your job got lost on a worker node. This can be due to a node failure or a network interruption. The only solution is a job resubmission.