
UFTP Docs

2023 UNICORE

Dec 20, 2023

UFTP DOCUMENTATION

1	UFTP Overview	3
1.1	UFTP Features	3
1.2	UFTP Architecture	4
1.3	How does UFTP work	4
1.4	UFTP Applications and Use Cases	5
2	User Documentation	7
2.1	UFTP client	7
2.1.1	Basic Usage	9
2.1.2	User Manual	9
2.1.2.1	Installation	9
2.1.2.1.1	Prerequisites	9
2.1.2.1.2	Installation and Configuration	10
2.1.2.2	Authentication	10
2.1.2.3	Usage	11
2.1.2.3.1	Listing a directory: the <code>ls</code> command	11
2.1.2.3.2	Copying data: the <code>cp</code> command	11
2.1.2.3.3	Data sharing	14
2.1.2.3.4	Server-to-server copy	17
2.1.2.4	Using a proxy server (EXPERIMENTAL)	18
2.1.2.5	Troubleshooting	18
2.1.3	Building	18
2.1.3.1	Prerequisites	19
2.1.3.2	Cloning the GitHub repository	19
2.1.3.3	Creating distribution packages	19
2.1.3.3.1	tgz	19
2.1.3.3.2	deb	19
2.1.3.3.3	rpm redhat	19
3	Admin Documentation	21
3.1	UFTPD Server	21
3.1.1	User Manual	22
3.1.1.1	Installation and operation	22
3.1.1.1.1	Prerequisites	22
3.1.1.1.2	Installation	23
3.1.1.1.3	Starting and stopping the UFTPD server	23
3.1.1.1.4	Configuration parameters	24
3.1.1.1.5	Protecting the Command socket	24
3.1.1.1.6	Firewall configuration	26
3.1.1.1.7	Logging	26

3.1.1.2	UNICORE integration	27
3.1.1.3	Testing the UFTPD server	27
3.2	Auth Server	27
3.2.1	User Manual	28
3.2.1.1	Installation	28
3.2.1.1.1	Prerequisites	28
3.2.1.1.2	Installation	28
3.2.1.1.3	Basic server configuration (memory etc)	29
3.2.1.1.4	Starting and stopping the service	29
3.2.1.2	Configuration	29
3.2.1.2.1	Features	29
3.2.1.2.2	UFTPD server(s) configuration	30
3.2.1.2.3	Round-robin use / grouping of UFTPD servers	31
3.2.1.2.4	User authentication	31
3.2.1.2.5	Attribute sources	34
3.2.1.2.6	Attribute mapping	34
3.2.1.2.7	Reservations	35
3.2.1.3	Checking the installation	36
3.2.1.4	Installing the Auth server in an existing UNICORE/X server	37
3.2.1.5	Running the Auth server behind a UNICORE Gateway	37
3.2.2	Update procedure	37
4	Getting Support	39
4.1	Support	39
5	License	41
5.1	License	41

UFTP (UNICORE File Transfer Protocol) is a high-performance data streaming library and file transfer tool with sharing capabilities. It can be also integrated into **UNICORE**, allowing to transfer data from client to server (and vice versa), as well as providing data staging and third-party transfer between UFTP-enabled UNICORE sites.

UFTP is best used using the client-side application (*UFTP client*), but is easily integrated into custom applications due to its FTP compliance.



UFTP Overview gives an overview of the UFTP architecture and features



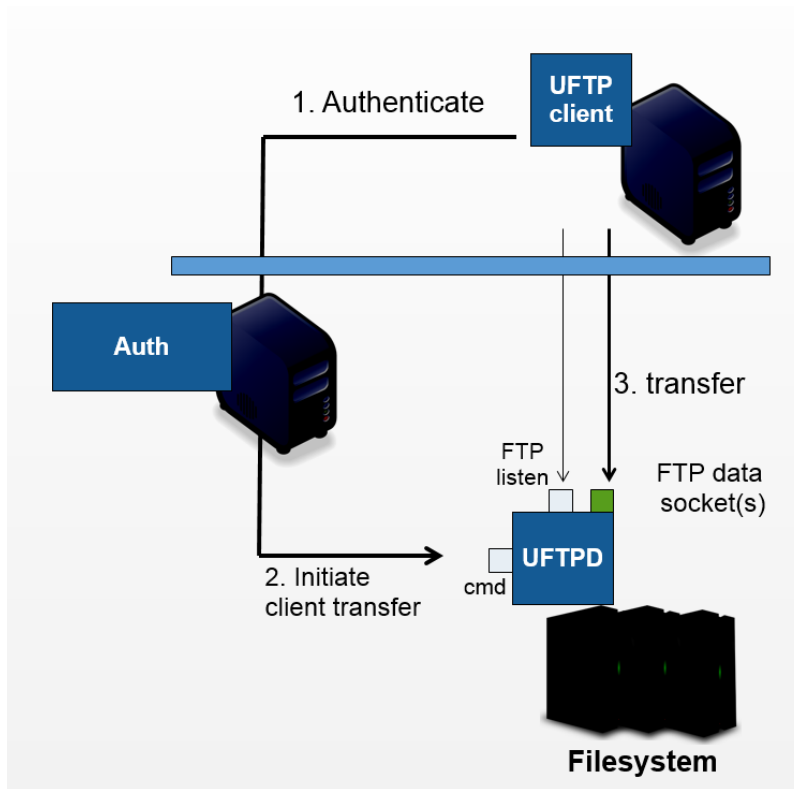
UFTP OVERVIEW

UFTP (**UNICORE FTP**) is a file transfer tool similar to Unix FTP. Its main features include high-performance file transfers from client to server (and vice versa), list directories, make/remove files or directories, sync files and data sharing. In addition, users can easily share their data even with users who do not have Unix-level access to the data.

1.1 UFTP Features

- Based on the FTP protocol with separate authentication via RESTful APIs
- Powerful *UFTP standalone client* available
- Optional: multiple FTP sessions per client for large-scale transfers, encryption and compression of the data streams
- Flexible integration options (authentication, user mapping), firewall-friendly
- System requirements: client: Java 11+, server: Python3

1.2 UFTP Architecture



The UFTP file server, called *UFTPD*, listens on two ports (which may be on two different network interfaces):

- the command port receives control commands from the authentication server(s)
- the listen port accepts data connections from clients.

The UFTPD server is *controlled* by an *Auth Server* or *UNICORE/X* via the command port, and receives/sends data directly from/to a client machine (which can be an actual user client machine or another server). The client, e.g. *UFTP client*, connects to the *listen* port, which has to be accessible from external machines. The client opens additional data connection(s) via the passive FTP protocol.

1.3 How does UFTP work

The sequence for a UFTP file transfer is as follows:

- the client (which can be an end-user client, or a service such as a *UNICORE/X server*) sends an authentication request to the *Auth server* (or another UNICORE/X server)
- the Auth server sends a request to the command port of UFTPD. This request notifies the UFTPD server about the upcoming transfer and contains the following information
 - a *secret*, i.e. a one-time password which the client will use to authenticate itself
 - the user and group id which uftpd should use to access files
 - an optional key to encrypt/decrypt the data
 - the client's IP address

- the UFTPD server will now accept an incoming client connection, provided the supplied *secret* (one-time password) matches the expectation.
- if everything is OK, an FTP session is created, and the client can use the FTP protocol to open data connections, list files, transfer data etc. Data connections are opened via *passive FTP*, which allows the firewall to dynamically open the requested ports (which can be any port, see below if you want to a fixed port range).
- for each UFTP session, UFTPD will fork a process which runs as the requested user (with the requested primary group).

1.4 UFTP Applications and Use Cases

- Secure, high-performance data access/transfer
 - Powerful *UFTP commandline client*
- Integrate data access/transfer functionality into web applications
 - RESTful APIs for authentication and FTP compliance for data access/transfer
- Data sharing in HPC environments
 - Authenticated or anonymous access
- UNICORE integration
 - Server-server file transfer and data staging for HPC applications and workflows
 - Integrated into UNICORE clients for fast file upload and download



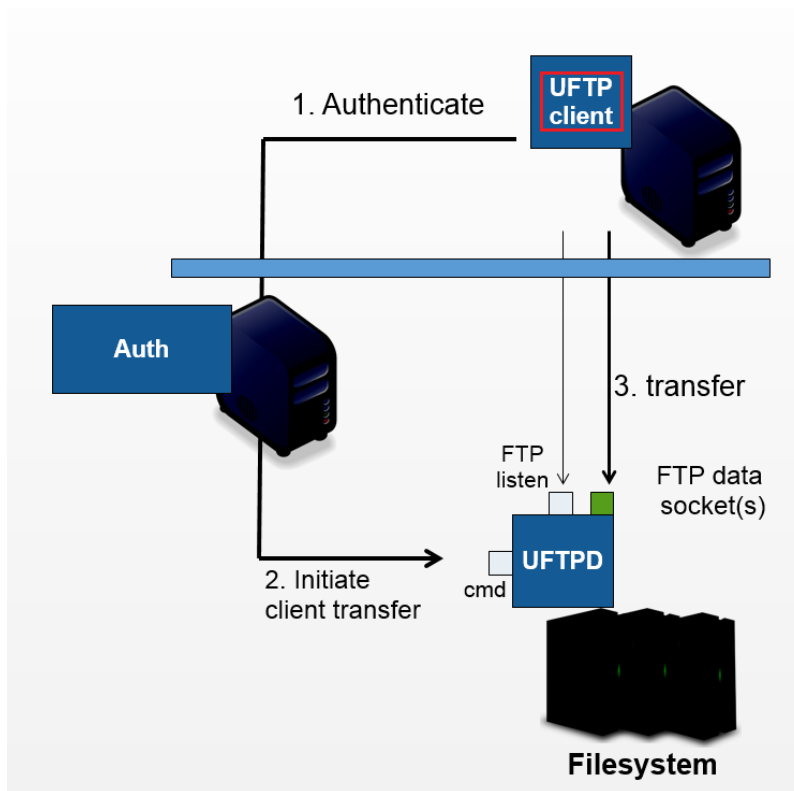
- *UFTP client* - the standalone UFTP client application

2.1 UFTP client

The UFTP commandline client enables users to

- list remote directories (`ls`)
- upload/download files (`cp`)
- compute checksums for remote files (`checksum`)
- sync remote/local files (`sync`)
- make remote directories (`mkdir`)
- delete remote files or directories (`rm`)
- manage shares and access shared data (`share`, `get-share`, `put-share`)
- perform authentication to help integrate UFTP with other tools
- launch server-to-server copy operations (`rcp`)


The UFTP client will connect to an authentication server (either a [UNICORE/X server](#) or the *Auth Server*) to authenticate and then to the *UFTPD Server* for transferring data or making a file operation.




The UFTP client supports username/password authentication, OIDC token authentication and ssh-key authentication. The UFTP client supports multiple concurrent FTP connections for highly efficient data transfers in high-performance environments.

Features

- *Commands* (UNIX-like semantics)
- Supports *multi-threaded transfers*, *encryption* and *compression* of the data streams
- *Flexible authentication*
 - sshkey incl. support for ssh-agent (on Linux only)
 - OIDC via oidc-agent
 - Username/password

 **Basic Usage** Basic usage of the UFTP Client.

 **User Manual** User Manual with detailed instructions and examples for using the UFTP Client.

 **Building** Building the UFTP Client distribution packages.

2.1.1 Basic Usage

In this manual, we use the following format to indicate commands that you can type on the command line:

```
$ some_command
```

and assume that the bin directory of the UFTP client is on your path.

- Invoking uftp without any arguments,

```
$ uftp
```

will list the available commands.

- Invoking

```
$ uftp <command> -h
```

will show help for a particular command

- Invoking

```
$ uftp -version
```

will show version information.

- For password authentication, the password can be given on the commandline, for example

```
$ uftp ls -u demo:password https://localhost:9000/rest/auth/TEST:/home/demo/
```

- When you specify the -P option, the password/passphrase will be queried interactively

```
$ uftp ls -u demo -P https://localhost:9000/rest/auth/TEST:/home/demo/
```

See also:

For detailed usage instructions and examples, refer to the [User Manual](#).

2.1.2 User Manual

The UFTP standalone Client is a Java-based commandline client for UFTP. It allows to list remote directories, copy files (with many options such as wildcards or multiple threads), and more. It can be used with either a [UFTP Authentication Server](#) or a UNICORE server to authenticate and initiate UFTP transfers.

2.1.2.1 Installation

2.1.2.1.1 Prerequisites

- Java 11 or later (OpenJDK preferred)
- Access to a UFTP authentication service (either an [Auth Server](#) or a [UNICORE/X](#) server) and to the corresponding [UFTPD Server](#).

To use the client, you need to know the address of the UFTP authentication service. You also need to have the valid credentials for the UFTP authentication.

2.1.2.1.2 Installation and Configuration

The UFTP client distribution packages are available at sourceforge.net.

If using the `zip` or `tar` archive, unpack it in a location of your choice. Add the `bin` directory to your path. Alternatively, you can link or copy the `bin/uft` script to a directory that is already on your path, in this case edit the script and setup the required directories.

If you use the `rpm` or `deb` package, install it using the package manager of your Linux distribution.

2.1.2.2 Authentication

By default, the UFTP client will use the current username (`$USER`) with SSH key authentication to authenticate to the *Auth Server*.

You can set a different default username via the `UFTP_USER` environment variable. This is useful if your remote username is not the same as your local username.

The UFTP client supports various means of authentication. Depending on the server you want to access, you can choose from

- user name with SSH key (default)
- user name with password
- `oidc-agent`
- manual specification of the HTTP Authorization header value

To explicitly specify the remote username, use the `-u <username>` option, e.g.

```
$ uftp ls -u username https://localhost:9000/rest/auth/TEST:/home/demo/
```

The credentials can be given in multiple ways.

- On the command line `-u username:password`

```
$ uftp ls -u username:password ...
```

- You can tell the uftp client to query the password interactively by giving the `-P` option, e.g.

```
$ uftp ls -u username -P ...
```

- If no password is given, the client will attempt to use an SSH key for authentication, this has to be configured on the authentication server accordingly. If you have multiple keys, use the `-i` option to select one. Otherwise, the client will check `~/.uftp/` and `~/.ssh/` for useable keys. The SSH agent is supported, too.
- The very useful `oidc-agent` tool is also directly supported via `-O <account_name>`. In this case no username is required.

```
$ uftp ls -O hbp ...
```

- You can directly specify a value for the HTTP *Authorization* header with the `-A` option. This allows to use an OIDC bearer token for authorization, e.g. `-A "Bearer <oidc_token>`. In this case no username is required.

```
$ uftp ls -A "Bearer <oidc_token>" ...
```

- If you explicitly DON'T want to send any authentication info, use `-u anonymous`.

2.1.2.3 Usage

In the following usage examples, the authentication service is located at *localhost:9000/rest/auth/* and the user name is *username*. Replace these values by the correct ones for your installation.

2.1.2.3.1 Listing a directory: the `ls` command

```
$ uftp ls https://localhost:9000/rest/auth/TEST:/home/demo/
```

will list the */home/demo* directory.

2.1.2.3.2 Copying data: the `cp` command

The `cp` command is used to copy local data to a remote server or vice versa. Remote locations are indicated by the `https://` prefix, and you need your user name, and the URL of the authentication server.

It has a number of features, which will be shown in the following.

Basic usage

- Downloading a single file:

```
$ uftp cp https://localhost:9000/rest/auth/TEST:/home/demo/test.data .
```

will download the */home/demo/test.data* file to the current directory

- Download files using wildcards:

```
$ uftp cp https://localhost:9000/rest/auth/TEST:/home/demo/data/* .
```

will download all files in the */home/demo/test* directory to the current directory

Similar commands work for upload.

- Uploading files using wildcards:

```
$ uftp cp "/data/*" https://localhost:9000/rest/auth/TEST:/home/demo/data/ .
```

The wildcards should be escaped to avoid the shell doing the expansion, which will also work, but generally be slower.

The recurse flag, `-r`, tells uftp to also copy subdirectories.

Piping data

The `cp` command can read/write from the console streams, which is great for integrating uftp into Unix pipes. The `-` is used as a special *file name* to indicate that data should be read/written using the console.

Transferring with tar and zip

For example to tar the contents of a directory and upload the tar file using uftp

```
$ tar cz dir/* | uftp cp - https://localhost:9000/rest/auth/TEST:/archive.tgz
```

The *UFTPD* server can also unpack tar and zip streams, this is very useful to efficiently transfer many small files. To enable this, add the `-a` option, and DO NOT compress the tar stream.

```
$ tar c dir/* | uftp cp -a - https://localhost:9000/rest/auth/TEST:/target_location/
```

or, using zip

```
$ zip -r - dir/* | uftp cp -a - https://localhost:9000/rest/auth/TEST:/target_location/
```

Note: Zip will compress data, so might be slower or faster than tar, depending on network bandwidth and processing speed.

Similarly, `-` can be used to write data to standard output. As an example, consider this

```
$ uftp cp https://localhost:9000/rest/auth/TEST:/archive.tgz - | tar tz
```

Or use uftp to cat a remote file

```
$ uftp cp https://localhost:9000/rest/auth/TEST:/foo.txt -
```

Using multiple FTP connections

When transferring large files (or many files) over a high-performance network, performance can be vastly improved by using multiple FTP connections. (NOTE this is different from the multiple TCP streams as set via the `-n` option).

Use the `-t` option to set the desired number of streams. Note that the server may have a limit on the allowed number of concurrent connections, if in doubt, ask your server administrator.

```
$ uftp cp -t 2 https://localhost:9000/rest/auth/TEST:/home/demo/* .
```

To split up files larger than a certain size and transfer them in chunks, you need to specify a “split size” using the `-T` option. For example, to split up files larger than 1MB

```
$ uftp cp -t 2 -T 1M https://localhost:9000/rest/auth/TEST:/home/demo/* .
```

Important: NOTE: uftp client versions before 1.7.0 have an automatic split size set to 512MB, with 1.7.0 and later you MUST use “`-T ...`” to enable file splitting.

Byte ranges

To copy just part of a file, a byte range can be given with the `-B` option. Counting starts at *zero*. For example to download only the first 1024 bytes of file (i.e. the byte range 0 - 1023), you would do

```
$ uftp cp -B 0-1023 https://localhost:9000/rest/auth/TEST:/home/demo/test.data .
```

As an additional feature, you can use the additional `-p` flag, which will write also only the given range. For example

```
$ uftp cp -B 1024-2047-p https://localhost:9000/rest/auth/TEST:/home/demo/test.data .
```

will write bytes 1024-2047 of the remote file to the local file, starting at offset 1024. The local file will have length 2048.

The same thing works for remote files!

Number of bytes to transfer

A simplified version of the byte range specification is to just give the amount of data to transfer (implying the start of the range is at byte 0)

This is handy for quick performance tests:

```
$ uftp cp -B 10G https://localhost:9000/rest/auth/TEST:/dev/zero /dev/null
```

Encryption and compression

The `cp` command supports the `-E` and `-C` options, which enable data encryption and compression (during transfer) respectively.

- Data encryption uses a symmetric algorithm, which nonetheless drastically lowers the performance.
- Data compression uses the `gzip` algorithm.

Compression and encryption can be combined.

Resuming a failed transfer

If a copy command was terminated prematurely, it can be resumed using the `-R` option. If the `-R` option is present, the UFTP client will check if the target file exists, and will append only the missing data.

So if your initial copy operation

```
$ uftp cp -u username https://localhost:9000/rest/auth/TEST:/home/demo/test.data .
```

did not finish correctly, you can resume it with

```
$ uftp cp -R https://localhost:9000/rest/auth/TEST:/home/demo/test.data .
```

Performance testing

For performance testing, you can use `/dev/zero` and `/dev/null` as data source / sink.

For example to transfer 10 gigabytes of zeros from the remote server:

```
$ uftp cp -B 0-10G https://localhost:9000/rest/auth/TEST:/dev/zero /dev/null
```

This can also be combined with the multi-connection option `-t`. To use two connections each transferring 5 gigabytes

```
$ uftp cp -B 0-10G -t 2 https://localhost:9000/rest/auth/TEST:/dev/zero /dev/null
```

Computing checksums for remote files

To compute a checksum for a remote file, use the `checksum` command:

```
$ uftp checksum https://localhost:9000/rest/auth/TEST:/data/*.dat
```

A number of different hashing algorithms are available, which can be selected using the `-a` option (MD5, SHA-1, SHA-256, SHA-256). For example

```
$ uftp checksum -a SHA-256 https://localhost:9000/rest/auth/TEST:/data/*.dat
```

Synchronizing a file: the `sync` command

Note that `sync` only supports single files, i.e. no directories or wildcards! The syntax is

```
$ uftp sync <master> <slave>
```

For example, to synchronize a local file with a remote *master* file:

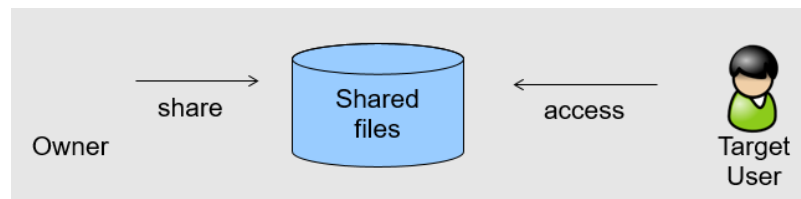
```
$ uftp sync https://localhost:9000/rest/auth/TEST:/master.file local.file
```

To synchronize a remote file with a local *master* file:

```
$ uftp sync master.file https://localhost:9000/rest/auth/TEST:/remote.file
```

2.1.2.3.3 Data sharing

Data sharing enables users to create access to their datasets for other users via UFTP, even if those users do not have Unix-level access to the data.



Data sharing works as follows:

- when you share a file (or directory), the *Auth Server* will store information about the path, the owner and the Unix user ID used to access the file in a database

- the targetted user can now access this file via the [Auth Server](#), and the Auth server will use the owner's Unix user ID to access the file.

By default, files will be shared for *anonymous* access. This will allow anyone who knows the sharing link to access the file using normal HTTP tools like `wget` or `curl`.

Shares can also be limited to certain users.

Depending on the type of share, access to the files is possible with the UFTP protocol or plain HTTPs.

Shares can be deleted by their owner, i.e. the user who created them.

Note: Not all UFTP installations support data sharing. You can check if a server has the sharing feature enabled by running `uftp info --server ...`

Server URL

If not given via the `--server` argument, the URL of the [Auth Server](#) will be taken from the environment variable `UFTP_SHARE_URL`

```
$ export UFTP_SHARE_URL=https://localhost:9000/rest/share/TEST
$ uftp share --list
```

Listing shares

```
$ uftp share --list --server https://localhost:9000/rest/share/TEST
```

The output will show both the files you have shared, as well as files that other users have shared with you.

Creating or updating a share

A share consists of a server-side path, (optional) write permissions and (optional) target user.

To share a file,

```
$ uftp share \
  --server https://localhost:9000/rest/share/TEST \
  /data/public/somefile.pdf
```

If you use a relative path, `uftp` will make it absolute.

```
$ pwd
> /data/public/
$ uftp share somefile.pdf
```

will share the path `/data/public/somefile.pdf`.

You can use the following options to modify the defaults:

- `--access <user-identifier>` to limit access to the specified user(s)
- `--write` for write acces
- `--delete` to delete a share

For example to share `/data/public/somefile.pdf` with the user `CN=User`

```
$ uftp share \  
    --server https://localhost:9000/rest/share/TEST \  
    --access "CN=User" \  
    /data/public/somefile.pdf
```

Shares can have a limited lifetime via the `--lifetime <seconds>` option.

Shares can also be limited to a single access via the `--one-time` option.

Deleting shares

To delete you need the path and the target user, which you can get via the `uftp share --list` command.

```
$ uftp share \  
    --delete \  
    --server https://localhost:9000/rest/share/TEST \  
    --access "CN=User" \  
    /data/public/somefile.pdf
```

Anonymous (https) access

For anonymous access via HTTP you need to use the correct URL. If you create (or list) shares, the UFTP client will show the required links. You can download the file e.g. using `wget`.

In case the share is a directory, `wget` will return a directory listing.

Downloading shared data using the UFTP protocol

It's possible to use the UFTP protocol to access shared data.

This can be also done anonymously by specifying “-u anonymous” on the `uftp` commandline.

The correct URLs for accessing shares via UFTP can be seen in the ‘uftp’ field of the output of the `--list` command.

To download a single shared file, use the `get-share` command

```
$ uftp get-share https://localhost:9000/rest/access/TEST:/data/public/somefile.pdf
```

In case the share is a directory, the standard `uftp ls` and `uftp cp` commands will work, too.

```
$ uftp ls https://localhost:9000/rest/access/TEST:/data/public/  
  
$ uftp cp https://localhost:9000/rest/access/TEST:/data/public/somefile.pdf ./downloaded.  
↪ pdf
```

Uploading to a share using the UFTP protocol

To upload a file to a location (file or directory) that has been shared with you, use the `put-share` command

```
$ uftp put-share data/*.pdf https://localhost:9000/rest/access/TEST:/data/public/
```

2.1.2.3.4 Server-to-server copy

REQUIRES UFTPD 3.2.0 or later (at least on one side)

The `rcp` command is used to instruct a remote UFTPD server to copy data from another UFTPD server. The client authenticates to both sides.

Basic usage

The basic syntax is similar to the normal `uftp cp` command:

```
$ uftp rcp <options> <source1> ... <sourceN> <target>
```

If the same means of authentication can be used for both source and target sides, both source and target are normal UFTP URLs. If source and target require different authentication, you need to use the `uftp auth` command first to authenticate to the one side (usually the source)

```
$ uftp auth <options> <source_URL>
```

and give the resulting host:port and one-time password to the `rcp` command via commandline options:

```
$ uftp rcp --server <host:port> --one-time-password <pwd> <source_file> <target>
```

Other supported features

The `rcp` command supports byte ranges via the `-B` option.

Reversing the copy direction

By default, the target side is instructed to download data from the source side. This can be reversed, if necessary, for example if only the source supports server-to-server copy. To do this, an environment variable can be set:

```
$ export UFTP_RCP_USE_SEND_FILE=true
$ uftp rcp ...
```

This will result in the source side uploading the file to the target side, and the `--server` and `--one-time-password` options will refer to the target side.

Known issues

There is no way to monitor or abort a running server-to-server transfer from the client.

Wildcards are not supported.

2.1.2.4 Using a proxy server (EXPERIMENTAL)

The uftp client has support for some types of FTP and HTTPs proxies.

This is configured via environment settings. I.e. in your shell you can define



- FTP proxy

```
export UFTP_PROXY=proxy.yourorg.edu
export UFTP_PROXY_PORT=21
```

- HTTP proxy

```
export UFTP_HTTP_PROXY=proxy.yourorg.edu
export UFTP_HTTP_PROXY_PORT=80
```

FTP proxying was tested with the DeleGate/9.9.13 and frox proxies and requires *UFTPD server* version 2.8.1 or later to work.

If this does not work for you, or if you require support for a different type of proxy, please contact us via a  support ticket or via  email.

2.1.2.5 Troubleshooting

How can I get more detailed logging?

In the client's **conf** directory you'll find a `logging.properties` file that allows you to increase the log levels.

I get “Invalid server response 500” and “Exception.... Authentication failure”

Probably you gave a wrong username or password. Contact your site administrator if in doubt! If using a password, make sure you give the `-P` flag.

I get “Invalid server response 405 Unable to connect to server for listing”

Check the remote URL that you use. Maybe you have a typo in the `/rest/auth/<servername>` part.

2.1.3 Building

This page shows how to build the latest version of the UFTP client directly from the sources.

2.1.3.1 Prerequisites

You need a git client, Java (11 or later) and Apache Maven.

2.1.3.2 Cloning the GitHub repository

The UFTP client is maintained as part of the [UFTP repository](#)

Clone this repository:

```
$ git clone https://github.com/UNICORE-EU/uftp.git
$ cd uftp
```

2.1.3.3 Creating distribution packages

The client code is in the `uftp-client` directory.

```
$ cd uftp-client
```

The following commands create the distribution packages in `tgz`, `deb` and `rpm` formats. The versions are taken from the `pom.xml`.

2.1.3.3.1 tgz

```
$ mvn package -DskipTests -Ppackman -Dpackage.type=bin.tar.gz
```

2.1.3.3.2 deb

```
$ mvn package -DskipTests -Ppackman -Dpackage.type=deb -Ddistribution=Debian
```

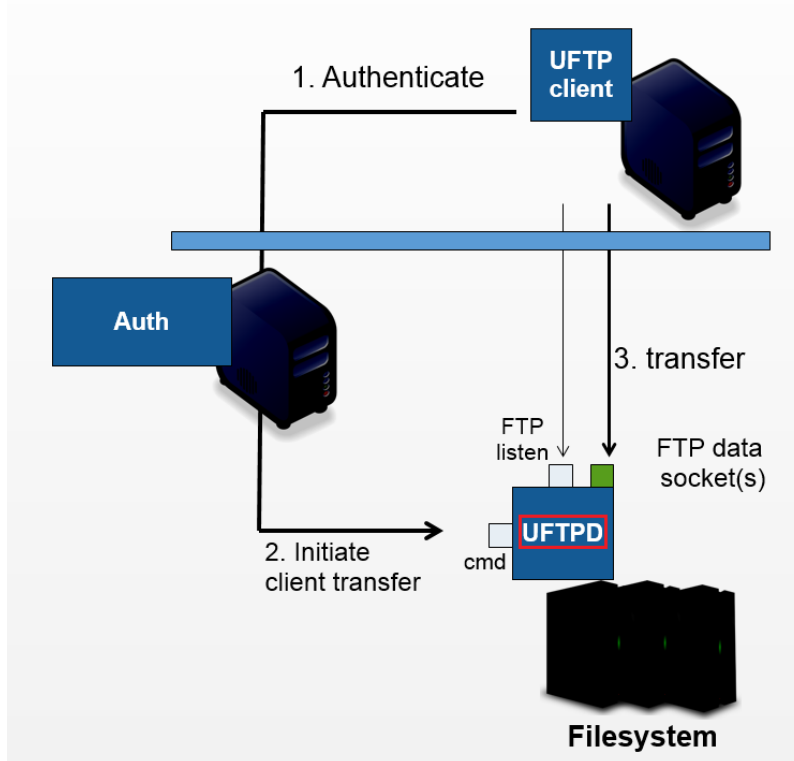
2.1.3.3.3 rpm redhat

```
$ mvn package -DskipTests -Ppackman -Dpackage.type=rpm -Ddistribution=RedHat
```


- *UFTPD Server* - the UFTP file server
- *Auth Server* - a set of services providing authentication for UFTP as well as data sharing features

3.1 UFTPD Server

The UFTPD server provides a high-performance data transfer based on passive FTP.




The UFTPD server listens on two ports (which may be on two different network interfaces):

- the command port receives control commands
- the listen port accepts data connections from clients

The UFTPD server is *controlled* by an *Auth Server* or *UNICORE/X* via the command port, and receives/sends data directly from/to a client machine (which can be an actual user client machine or another server).

Features

- FTP-compliant data server running on a POSIX file system
- Runs privileged on a server with access to the file systems to be served, fully drops privileges to current user for all operations
- Requires
 - Python 3.6 or later
 - Server certificate
- Firewall requirements
 - Allow incoming connections to *listen* port (FTP port)
 - Enable FTP connection tracking OR open port range
 - Allow incoming connection to *control* port from *Auth Server*

 **User Manual** Installation and Operating the UFTPD server.

3.1.1 User Manual

This is the user manual providing information on running and using the UNICORE UFTPD server.

Important: IMPORTANT SECURITY NOTE

The UNICORE UFTPD server is running with **elevated privileges**, it can set its UID and GID to that of any user except *root*. Make sure to read and understand the section below on *Protecting the Command socket*. Otherwise, users logged on to the UFTPD machine can possibly read and write other user's files.

3.1.1.1 Installation and operation

3.1.1.1.1 Prerequisites

- Python 3.6.0 or later
- the server's *listen* port needs to be accessible through your firewalls, declaring it an *FTP* port (FTP connection tracking). Alternatively a fixed range of open ports can be configured and used.
- the server's command port needs to be accessible from the Auth server(s)
- the UFTPD server needs access to the target file systems
- a server certificate for the UFTPD server is a **MUST** for production use in a multi-user environment (see the section on SSL below)
- the data encryption feature requires the Python "Crypto" module, which can be installed via `python3 -m pip install pycryptodome`

Attention: A functional UFTP installation requires also the *Auth Server* or a full UNICORE/X server .

3.1.1.1.2 Installation

The UNICORE UFTP server is distributed either as a platform independent and portable tar.gz or zip bundle or as an installable, platform dependent package such as RPM available at sourceforge.net.

Important: IMPORTANT NOTE ON PATHS

Depending on the installation package, the paths to various files are different.

If installing using distribution-specific package the following paths are used:

```
CONF=/etc/unicore/uftpd
BIN=/usr/share/unicore/uftpd/bin
LIB=/usr/share/unicore/uftpd/lib
```

If installing using the portable bundle, all UFTP files are installed under a single directory. Path prefixes are as follows, where *INST* is the directory where UFTP was installed:

```
CONF=INST/conf
BIN=INST/bin
LIB=INST/lib
```

These variables (*CONF*, *BIN* and *LOG*) are used throughout the rest of this manual.

Note that after installation UFTP is **NOT** automatically enabled as a `systemd` service, since you will need to edit the configuration and provide a server certificate.

3.1.1.1.3 Starting and stopping the UFTP server

If using the Linux packages, uftpd is integrated as a service via `systemd`, and you can stop/start it via `systemctl`. Also, logging is (by default) done via `systemd`, and you can look at the logs via `journalctl`.

To do things manually, you can use the start/stop and status scripts that are provided in the BIN directory.

- `unicore-uftpd-start.sh` starts the server
- `unicore-uftpd-stop.sh` stops the server
- `unicore-uftpd-status.sh` checks the server status

The parameters such as server host/port, control host/port, and others are configured in the `CONF/uftpd.conf` file.

In a production scenario with multiple users, the uftpd server needs to be started as *root*. This is necessary to be able to access files as the correct user/group and set correct file permissions.

To enable UFTP as a `systemd` service (after configuring and adding a server certificate), you can use `systemctl`:

```
$ sudo systemctl add-wants multi-user.target unicon-uftpd
```

3.1.1.1.4 Configuration parameters

The following variables can be defined in the configuration file (`uftpd.conf`):

- CMD_HOST** the interface where the server listens for control commands
- CMD_PORT** the port where the server listens for control commands
- SERVER_HOST** the interface where the server listens for client data connections
- SERVER_PORT** the port where the server listens for client data connections
- ADVERTISE_HOST** (*optional, only used in the PASV implementation*) Advertise this server as having the following IPv4 address in the control connection. This is useful if the server is behind a NAT firewall and the public address is different from the address(es) the server has bound to
- SSL_CONF** File containing SSL settings for the command port
- ACL** File containing the list of server DNs that are allowed access to the command port
- MAX_CONNECTIONS** the maximum number of concurrent control connections per user (default: 16)
- MAX_STREAMS** the maximum number of parallel TCP streams per FTP session (default: 4)
- PORT_RANGE** (*optional*) server-side port range in the form 'lower:upper' that will be used to accept data connections. By default, any free ports will be used. *Example:* set to '50000:50050' to limit the port range.
- DISABLE_IP_CHECK** (*optional*) in some situations, the client IP can be different from the one that was sent to the UFTP server by the Auth server. This will lead to rejected transfers. Setting this variable to *true* will disable the IP check. Only the one-time password will be checked.
- UFTP_KEYFILES** (*optional*) list of files (relative to current user's \$HOME) where uftpd will read public keys for authentication. List is separated by `:`. This defaults to `.ssh/authorized_keys`.
- UFTP_NO_WRITE** (*optional*) `“:”`-separated list of file name patters that uftpd should not write to.
- LOG_VERBOSE** set to *true* to get (much) more detailed logging
- LOG_SYSLOG** set to *false* to print logging output to stdout

As usual if you set the `SERVER_HOST` to be `0.0.0.0`, the server will bind to all the available network interfaces.

If possible, use an *internal* interface for the Command socket. If that is not possible, make sure the Command socket is protected by a firewall!

Attention: We **VERY STRONGLY** recommend enabling SSL for the Command socket. Please refer to the next section.

3.1.1.1.5 Protecting the Command socket

Using SSL for the Command port ensures that only trusted parties (i.e. trusted Auth and/or UNICORE/X servers) can issue commands to the UFTP server. To further limit the set of trusted users, an access control list (ACL) file is used.

In production settings where users can log in to the UFTP server machine, **SSL MUST** be enabled to prevent unauthorized data access!

Important: IMPORTANT SECURITY NOTE

Without SSL enabled, users logged in to the UFTPD server can easily create exploits to read or write files with arbitrary user privileges (except *root*).

SSL setup

To setup SSL, you need a PEM file containing the UFTPD server's credential, and a PEM file containing certificate authorities that should be trusted.

The following properties can be set in the `CONF/uftpd-ssl.conf` file.

```
credential.path=path/to/keyfile.pem
credential.password=...

truststore=path/to/ca-cert-file.pem
```

You can also use separate PEM files for key and certificate:

```
credential.key=path/to/key.pem
credential.password=...
credential.certificate=path/to/certificate.pem

truststore=path/to/ca-cert-file.pem
```

The `credential.password` is only needed and used if the key is encrypted.

Note: Backwards (in)compatibility to previous versions

UFTPD 2.x SSL config is **NOT supported**.

If you already have a p12 keystore for UFTPD 2.x, you can use `openssl` to convert it to *PEM* format.

ACL setup

The access control list contains the distinguished names of those certificates that should be allowed access.

The ACL setting in `CONF/uftpd.conf` is used to specify the location of the ACL file:

```
export ACL=conf/uftpd.acl
```

The default ACL contains the certificate DN of the UNICORE/X server from the [UNICORE core server bundle](#). In production, you need to replace this by the actual DNs of your [UNICORE/X server\(s\)](#) and [UFTP Authentication server\(s\)](#).

The ACL entries are expected in RFC2253 format. To get the name from a certificate in the correct format using `openssl`, you can use the following OpenSSL command:

```
$ openssl x509 -in your_server.pem -noout -subject -nameopt RFC2253
```

The ACL file can be updated at runtime.

3.1.1.1.6 Firewall configuration

UFTPD requires

- an open TCP port for accepting FTP connections
- additional open TCP ports for accepting data connections

The data connections can either be opened dynamically using *FTP connection tracking*, or you can use a dedicated port range and permanently open those in the firewall.

Note: Please refer to the firewall documentation on how to enable an *FTP* service on your firewall (or operating system).

With Linux `iptables`, you may use rules similar to the following:

```
$ iptables -A INPUT -p tcp -m tcp --dport $SERVER_PORT -j ACCEPT
$ iptables -A INPUT -p tcp -m helper --helper ftp-$SERVER_PORT -j ACCEPT
```

where `$SERVER_PORT` is the `SERVER_PORT` defined in `uftp.conf`. The first rule allows anyone to access port `$SERVER_PORT`. The second rule activates the `iptables` connection tracking FTP module on port `$SERVER_PORT`.

On some operating systems it may be required to load additional kernel modules to enable connection tracking, for example on CentOS:

```
$ modprobe nf_conntrack_ipv4
$ modprobe nf_conntrack_ftp ports=$SERVER_PORT
```

If you cannot use connection tracking, you will need to open a port range, and configure UFTPD accordingly.

For example, in `uftp.conf`

```
export PORT_RANGE=21000:21010
```

and the `iptables` rule

```
$ iptables -A INPUT -p tcp -m tcp --dport 21000:21010 -j ACCEPT
```

would allow incoming data connections on ports 21000 to 21010.

A fairly small range (e.g. 10 ports) is usually enough, since these are server ports.

3.1.1.1.7 Logging

By default, UFTPD writes to `syslog`, and you can use `journalctl` to read log messages. To print logging output to `stdout`, set `export LOG_SYSLOG=false` in the `uftp.conf` file.

3.1.1.2 UNICORE integration

Please refer to the [UNICORE/X manual](#) for detailed information on how to configure UFTP based data access and data transfer.

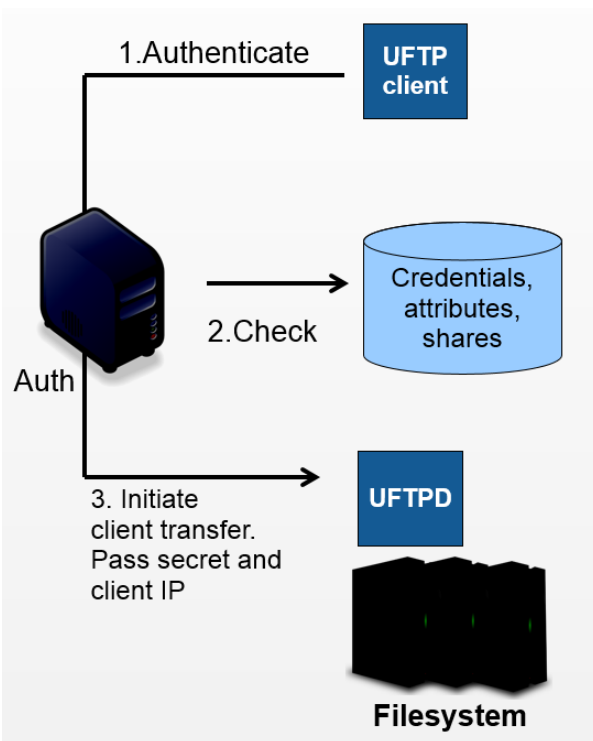
3.1.1.3 Testing the UFTPD server

You should use the `uftp client` to run tests, which contains many options such as the number of concurrent FTP connections, and can use `/dev/null` and `/dev/zero` as data source/sink.

3.2 Auth Server

The UFTP authentication service (**Auth server**) is RESTful service for authenticating users and initiating UFTP transfers. It is intended to be used with a standalone UFTP client and provides access to one or more *UFTPD servers*.

Besides data transfer via UFTP and data management features like `ls`, the Auth server also provides REST services for data sharing and accessing shared data sets.





The Auth server is based on the UNICORE Services Environment, and all usual UNICORE features and security configuration options are available as well. For example, the Auth server can be deployed behind a [UNICORE Gateway](#), or it can be configured to use [Unity](#) for authenticating users.

Features

- RESTful service

- Authentication
 - SSH keys
 - Unity (OAuth token)
 - PAM
 - Username/password file
 - ... (extensible)
- Attribute mapping
 - UNIX uid, gid
 - QoS e.g. rate limit
- Reservations

 **User Manual** Installation and Operating the Auth server.

 **Update procedure** Upgrade the Auth server to this version.

3.2.1 User Manual

This manual focuses on the configuration items specific to the Auth server. If you need more in-depth information on general configuration issues, please refer to the [UNICORE/X manual](#).

3.2.1.1 Installation

3.2.1.1.1 Prerequisites

The Auth server should be run as a non-root user (e.g. *unicore*). It requires

- Java 11
- an installed *UFTPD Server*

The Auth server needs an X.509 certificate and truststore for communicating with the *UFTPD Server*.

Users must be able to access the Auth server's https port. It is possible to deploy the Auth server behind a [UNICORE Gateway](#) (please see *Running the Auth server behind a UNICORE Gateway* below).

3.2.1.1.2 Installation

The UFTP Auth service is distributed either as a platform independent and portable `tar.gz` or `zip` bundle available at sourceforge.net. It comes with all required scripts and config files to be run as a standalone application. To install, unzip the downloaded package into a directory of your choice.

Note: You can run the service in an existing [UNICORE/X server](#) (8.0.0 or later). Please see *Installing the Auth server in an existing UNICORE/X server* below for details.

3.2.1.1.3 Basic server configuration (memory etc)

The `startup.properties` configuration file contains basic settings such as the Java command, JVM memory etc. Please review it.

The Auth server host and port are configured in the `container.properties` configuration file:

```
container.host=uftp.yoursite.com
container.port=9000

# if running behind a UNICORE Gateway or a NAT router,
# set the baseurl
container.sitename=AUTH
container.baseurl=gateway.yoursite.com:2222/AUTH/services
```

Also in the `container.properties` configuration file, the server's X.509 private key and the truststore settings need to be configured.

3.2.1.1.4 Starting and stopping the service

Use the shell scripts in the `bin` folder to start or stop the service.

3.2.1.2 Configuration

The following items need to be configured in the Auth server's `container.properties` file:

- *UFTP* server(s) to be accessed
- *User authentication*: configure the Auth server to authenticate users using *username/password*, *ssh key* or via *Unity*
- *Attribute sources* (XUADB, map file, ...) for assigning local attributes like UNIX user name to authenticated users

3.2.1.2.1 Features

This service provides two features

- AuthServer
- DataSharing

both are enabled by default. To disable data sharing, set

```
container.feature.DataSharing.enable=false
```

There are no further configuration options for these features.

3.2.1.2.2 UFTPD server(s) configuration

For each *UFTPD server* that should be accessed, you'll need to configure the relevant properties in the Auth service's config file.

The `authservice.servers` property is a list of server names. These should be meaningful, since users will need to use them, too. The other properties are used to configure the UFTPD command address and the UFTPD listen address. Please refer to the [UFTPD manual](#) for more information about these ports.

description	human-readable description of the UFTPD server
host	the IP address of the UFTPD <i>listen</i> socket
port	the port of the UFTPD <i>listen</i> socket
commandHost	the IP address of the UFTPD <i>command</i> socket
commandPort	the port of the UFTPD <i>command</i> socket
ssl	whether SSL is used to connect to the command socket. This MUST be set to its default of <code>true</code> in a production environment!
reservations.enable	whether to enable the <i>reservations</i> feature
reservations.file	JSON file containing reservations definitions

Note: The listen socket address will be communicated to clients, who will attempt to connect to that address. Therefore, this has to be a public interface. For example, if you are running UFTPD behind a NAT router, you have to use the IP configured as the `ADVERTISE_HOST` in the UFTPD configuration.

For example, we want to configure two UFTPD servers named *CLUSTER* and *TEST*:

```
# configured UFTPD server(s)
authservice.servers=CLUSTER TEST

# configuration for 'CLUSTER' server
authservice.server.CLUSTER.description=Production UFTPD server
authservice.server.CLUSTER.host=cluster.your.org
authservice.server.CLUSTER.port=64433
authservice.server.CLUSTER.commandHost=cluster-internal.your.org
authservice.server.CLUSTER.commandPort=64434
authservice.server.CLUSTER.ssl=true

# configuration for 'TEST' server
authservice.server.TEST.description=Test UFTPD server
authservice.server.TEST.host=localhost
authservice.server.TEST.port=64433
authservice.server.TEST.commandHost=localhost
authservice.server.TEST.commandPort=64434
authservice.server.TEST.ssl=false
```

To allow the Auth server access to the command port of UFTPD, you need to add an entry to UFTPD's ACL file. This is explained in the [UFTPD manual](#).

3.2.1.2.3 Round-robin use / grouping of UFTP servers

You can configure multiple UFTP servers to form a *logical* UFTP server. The idea is that multiple UFTP servers are used in a round robin fashion to provide better performance.

Also, this mode of operation will provide fail-over if one of the UFTP servers is down for maintenance or upgrades (or because of some error).

In this case the configuration for the logical server has multiple blocks numbered 1, 2, ...

Each block configures one physical server. For example:

```
# configuration for multiple UFTP instances
# providing the logical 'CLUSTER' server

authservice.servers=CLUSTER

authservice.server.CLUSTER.description=Production UFTP server on CLUSTER

authservice.server.CLUSTER.1.host=cluster1.your.org
authservice.server.CLUSTER.1.port=64433
authservice.server.CLUSTER.1.commandHost=cluster-internal-1.your.org
authservice.server.CLUSTER.1.commandPort=64434
authservice.server.CLUSTER.1.ssl=true

authservice.server.CLUSTER.2.host=cluster2.your.org
authservice.server.CLUSTER.2.port=64433
authservice.server.CLUSTER.2.commandHost=cluster-internal-2.your.org
authservice.server.CLUSTER.2.commandPort=64434
authservice.server.CLUSTER.2.ssl=true
```

3.2.1.2.4 User authentication

The Auth service is a RESTful UNICORE service, and as such all the configuration details for a UNICORE/X server apply here as well.

We summarise the most important details, please refer to the [UNICORE/X manual](#) if you want to learn about further options.

The enabled authentication options and their order are configured in `container.properties`.

```
container.security.rest.authentication.order=PASSWORD | SSHKEY | UNITY
```

The available options can be combined.

Username-password file

To use a file containing username, password and the DN,

```
container.security.rest.authentication.order=PASSWORD
container.security.rest.authentication.PASSWORD.class=eu.unicore.services.rest.security.
↳FilebasedAuthenticator
container.security.rest.authentication.PASSWORD.file=conf/rest-users.txt
```

This configures to use the file `conf/rest-users.txt`. The file format is

```
#
# on each line:
# username:hash:salt:DN
#
demouser:<...>:<...>:CN=Demo User, O=UNICORE, C=EU
```

i.e. each line gives the username, the hashed password, the salt and the user's DN, separated by colons. To generate entries, i.e. to hash the password correctly, the `md5sum` utility can be used. For example, if your intended password is `test123`, you could do

```
$ SALT=$(tr -dc "A-Za-z0-9_" < /dev/urandom | head -c 16 | xargs)
$ /bin/echo "Salt is ${SALT}"
$ /bin/echo -n "${SALT}test123" | md5sum
```

which will output the salted and hashed password. Here we generate a random string as the salt. Enter these together with the username, and the DN of the user into the password file.

Unity SAML authentication

You can also hook up with [Unity](#), passing on the username/password and retrieving an authentication assertion.

```
container.security.rest.authentication.order=UNITY

container.security.rest.authentication.UNITY.class=eu.unicore.services.rest.security.
↳UnitySAMLAuthenticator
container.security.rest.authentication.UNITY.address=https://localhost:2443/unicore-
↳soapidp/saml2unicoreidp-soap/AuthenticationService
container.security.rest.authentication.UNITY.validate=true
```

Unity OAuth bearer token authentication

To have Unity check the client's OAuth token:

```
container.security.rest.authentication.order=UNITY-OAUTH
container.security.rest.authentication.UNITY-OAUTH.class=eu.unicore.services.rest.
↳security.UnityOAuthAuthenticator
container.security.rest.authentication.UNITY-OAUTH.address=https://localhost:2443/
↳unicore-soapidp.oidc/saml2unicoreidp-soap/AuthenticationService
container.security.rest.authentication.UNITY-OAUTH.validate=true
```

SSH Key validation

This authentication option is based on the validation of a token using the user's public SSH key. The token will be checked, and if successful, the user will be assigned a distinguished name for later authorisation.

SSH keys are read from the user's `~/.ssh/authorized_keys` file, but can also be managed manually in a dedicated ssh keys file.

Note: SSH key validation will not work for users on Windows, since the UFTP stand-alone client does not yet support SSH keys on Windows. We recommend adding a username/password option for Windows users.

SSH key validation is configured as follows:

```
# authN
container.security.rest.authentication.order=SSHKEY

container.security.rest.authentication.SSHKEY.class=eu.unicore.uftp.authserver.
↪authenticate.SSHKeyAuthenticator
```

When used like this, the users get an automatically assigned DN. By default, the DN is `CN=<username>, OU=ssh-local-users`. Using the *PAM attribute source* (see [below](#)), authenticated users can be assigned the *user* role automatically without further configuration.

The user DN can be modified by configuring the DN template like this:

```
#DN template used for SSH key mapping. The %s is replaced by the username
container.security.rest.authentication.SSHKEY.dnTemplate=CN=%s, OU=ssh-local-users
```

Manual SSH key mapping

If you want to map ssh keys to DNs manually, a file is used. Entries in the file override the keys read from `~/.ssh/authorized_keys`.

```
# configure SSH keys file
container.security.rest.authentication.SSHKEY.file=conf/ssh-users.txt
```

It contains the mappings and the ssh public keys in a simple format:

```
# Example SSH users file used with the SSHKEY authentication method

#
#format: username:sshkey:DN
#
demouser:ssh-rsa keydata_was_omitted testkey:CN=Demo User, O=UNICORE, C=EU
```

The SSH key is in the same one-line format used in the `.ssh/authorized_keys` file.

You can enter multiple lines per username, to accommodate the case that a user has different SSH keys available. For example

```
# Example SSH users file with multiple keys per user

demouser:ssh-rsa <...omitted keydata...>:CN=Demo User, O=UNICORE, C=EU
```

(continues on next page)

(continued from previous page)

```
demouser:ssh-dss <...omitted keydata...>:CN=Demo User, O=UNICORE, C=EU
otheruser:ssh-rsa <...omitted keydata...>:CN=Other User, O=UNICORE, C=DE
```

3.2.1.2.5 Attribute sources

Please refer to the [UNICORE/X manual](#) on how to set up and configure attribute sources like [map file](#) or [XUADB](#).

To use the automatic SSH key mapping, please use this config snippet

```
# attribute source(s)
container.security.attributes.order=PAM
container.security.attributes.combiningPolicy=MERGE_LAST_OVERRIDES

container.security.attributes.PAM.class=eu.unicore.services.rest.security.
↳PAMAttributeSource
```

In this way users that successfully authenticate with their SSH key get the *user* role automatically.

3.2.1.2.6 Attribute mapping

After successful authentication, the user is assigned attributes such as the Unix account and group which is used for file access.

The Unix account and group are taken from the configured attribute sources (e.g. [XUADB](#)). Since it is possible to access multiple UFTP servers using a single Auth server, it may be required to configure different attributes for different UFTP servers. This is easily possible using the file attribute source (map file).

It is also possible to control which directories and files a user can access. This is done by configuring the allowed and/or the forbidden file path patterns.

The following map file entry gives a full example.

```
<entry key="CN=Demo User,O=UNICORE,C=EU">
  <attribute name="role">
    <value>user</value>
  </attribute>

  <!-- default Unix account and group -->
  <attribute name="xlogin">
    <value>somebody</value>
  </attribute>
  <attribute name="group">
    <value>users</value>
  </attribute>

  <!-- UFTP specific attributes -->

  <attribute name="uftp.CLUSTER.xlogin">
    <value>user1</value>
  </attribute>
  <attribute name="uftp.CLUSTER.group">
    <value>hpc</value>
  </attribute>
</entry>
```

(continues on next page)

(continued from previous page)

```

</attribute>

<!-- optional rate limit (bytes per second) -->
<attribute name="uftp.CLUSTER.rateLimit">
  <value>10M</value>
</attribute>

<!-- optional includes -->
<attribute name="uftp.CLUSTER.includes">
  <value>/tmp/*:/work/*</value>
</attribute>
<!-- optional excludes -->
<attribute name="uftp.CLUSTER.excludes">
  <value>/home/*:/etc/*</value>
</attribute>

</entry>

```

Here, the *CLUSTER* must match a configured UFTP server, see also *UFTP server(s) configuration*. Available attributes are

- role** the UNICORE role, usually this will be *user*.
- xlogin, group** Unix account and group to be used for this user.
- rateLimit** the number of bytes per second (per transfer) can be limited. You can use the units “K”, “M”, and “G” for kilo, mega or gigabytes, respectively.
- includes** file path patterns (separated by :) that are allowed. If not given, all the user’s files can be accessed.
- excludes** file path patterns (separated by :) that are forbidden. If not given, no files are explicitly excluded.

3.2.1.2.7 Reservations

It is possible (v2.8.2 and later) to define reservations, i.e. time slots where certain users can get more of the available bandwidth for UFTP transfers. During such a reservation, other users are rate-limited. The Auth server reads reservations from a local JSON file, which can be edited at runtime by an admin.

To enable, define the following two settings in the *UFTP configuration section*:

```

# configured UFTP server(s)
authservice.servers=CLUSTER

# enable reservations feature for 'CLUSTER' server
authservice.server.CLUSTER.reservations.enable=true
authservice.server.CLUSTER.reservations.file=/path/to/reservations.json

```

The `reservations.json` file can be added / edited at runtime, and updates will be read from it.

The format of the JSON file is the following:

```

{
  "reservations": [

```

(continues on next page)

(continued from previous page)

```

{
  "name": "reservation1",
  "from": "2023-08-31 16:00",
  "to": "2023-08-31 18:00",
  "uids": [ "user1", "user2" ],
  "rateLimit": "10m"
},
{
  "name": "reservation2",
  "from": "2023-09-22 08:00",
  "to": "2023-09-22 09:00",
  "uids": [ "user3" ],
  "rateLimit": "100k"
}
]
}

```

and should be self-explanatory. The `from` and `to` fields give the start/end time of the reservation in `yyyy-MM-DD hh:mm` format, while the `uids` lists the Unix logins of the users that should NOT be limited to the transfer rate given by `rateLimit`.

The rate limit is optional, and defaults to “10m” i.e. 10MB/sec.

Note that the rate limit can only be applied to new connections, all FTP sessions already existing at the start time of the reservation will not be affected.

3.2.1.3 Checking the installation

You can check that the server works using a simple HTTP client such as `curl` to access the Auth server’s base URL, provided you have configured username/password authentication.

The command

```

$ curl -k https://<host:port>/rest/auth \
  -H "Accept: application/json" \
  -u username:password

```

should produce a JSON document containing information about the configured UFTPD servers and their status, such as

```

{"TEST": {
  "availableGroups": [
    "somebody",
    "audio",
    "users"
  ],
  "description": "Default UFTPD server for testing",
  "gid": "users",
  "href": "https://localhost:9000/rest/auth/TEST",

```

(continues on next page)

(continued from previous page)

```
"rateLimit": 209715200,
"status": "OK [connected to UFTPD localhost:64435]",
"uid": "somebody",
}}
```

Note: If you do not get any output, try adding the `-i` option to the `curl` command, most probably the user-name/password is incorrect.

3.2.1.4 Installing the Auth server in an existing UNICORE/X server

This option is interesting if you are already running a UNICORE installation and want to allow your users the option of using the standalone *UFTP client*. This requires UNICORE/X version 8.0 or later!

- copy the `authserver-*.jar` file to the `lib` directory of UNICORE/X
- copy the XACML policy file `30uftpAuthService.xml` to the `conf/xacml2Policies` directory
- edit `container.properties` (or `uas.config`) and setup UFTPD details and, if necessary, RESTful user authentication as described above

3.2.1.5 Running the Auth server behind a UNICORE Gateway

If you want to place the Auth server behind a UNICORE gateway for easy firewall transversal, you need to configure an entry in the *Gateway connections* config file, and set the container base URL property (`container.baseurl`) in the Auth server's `container.properties`. This option is also useful when the server's listen address differs from the publicly accessible server address, such as when running the Auth server behind a NAT firewall.

3.2.2 Update procedure

As a first step and precaution, you should make backups of your existing config files and put them in a safe place.

In the following, *LIB* refers to the directory containing the jar files for the component, and *CONF* to the config directory of the existing installation.

It is assumed that you have unpacked the `tar.gz` file somewhere, e.g. to `/tmp/`. In the following, this location will be denoted as *\$NEW*:

```
$ export NEW=/tmp/unicore-authserver-2.5.0
```

- Stop the server. If not yet done, make a backup of the config files.
- Update the jar files:

```
$ rm -rf LIB/*
$ cp $NEW/lib/*.jar LIB
```

- Check for other changes

While rarely changed, sometimes the XACML policy files are updated for new releases. These can be found in `$NEW/conf/xacml2Policies/`. If necessary, copy these to your installation:

```
$ rm -rf conf/xacml2Policies/*  
$ cp $NEW/conf/xacml2Policies/* conf/xacml2Policies/
```

- Start the server.
- Check the logs for any **ERROR** or **WARN** messages and if necessary correct them.

GETTING SUPPORT

For more information, please see the *Support* page.

4.1 Support

For additional information and support, please visit:

 **Website** <https://www.unicore.eu>

 **Support list** unicore-support@lists.sf.net

 **Developer's list** unicore-devel@lists.sf.net

 **Issue Tracker** <https://sourceforge.net/p/unicore/uftp-issues>

 **Source code** <https://github.com/UNICORE-EU/uftp>



UFTP is available under the *BSD 2-Clause License*.

5.1 License

BSD 2-Clause License

Copyright (c) 1997-2023

Forschungszentrum Jülich GmbH, Fujitsu Labs Europe, ICM Warsaw,
Intel Corporation, CINECA, University of Manchester, T-Systems,
and other contributors to UNICORE: <https://www.unicore.eu>

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.