Waarp R66 manual

Presentation

Waarp is a project that provides among other an open source massive file transfer monitor in Java. Its goal is to unify several protocols (FTP, FTPS, SSH, HTTP and proprietary protocols) in an efficient and secured way. Its purpose is to enable bridging between several protocols and to enable dynamic pre or post action on transfer or other commands. Currently FTP(S) and efficient and secure R66 protocols are implemented. HTTP is supported but not yet in production release.

This project was developed initially for the French Ministry of Finance. The 3 main components (originally named GoldenGate, GoldenGate FTP and OpenR66) are in production in the Ministry since end of 2007. It is also in production in the French Gendarmerie Nationale since 2012. It is an Open Source project based on an independent community.

Support

A company named **Waarp** holds the ability to provide professional services:

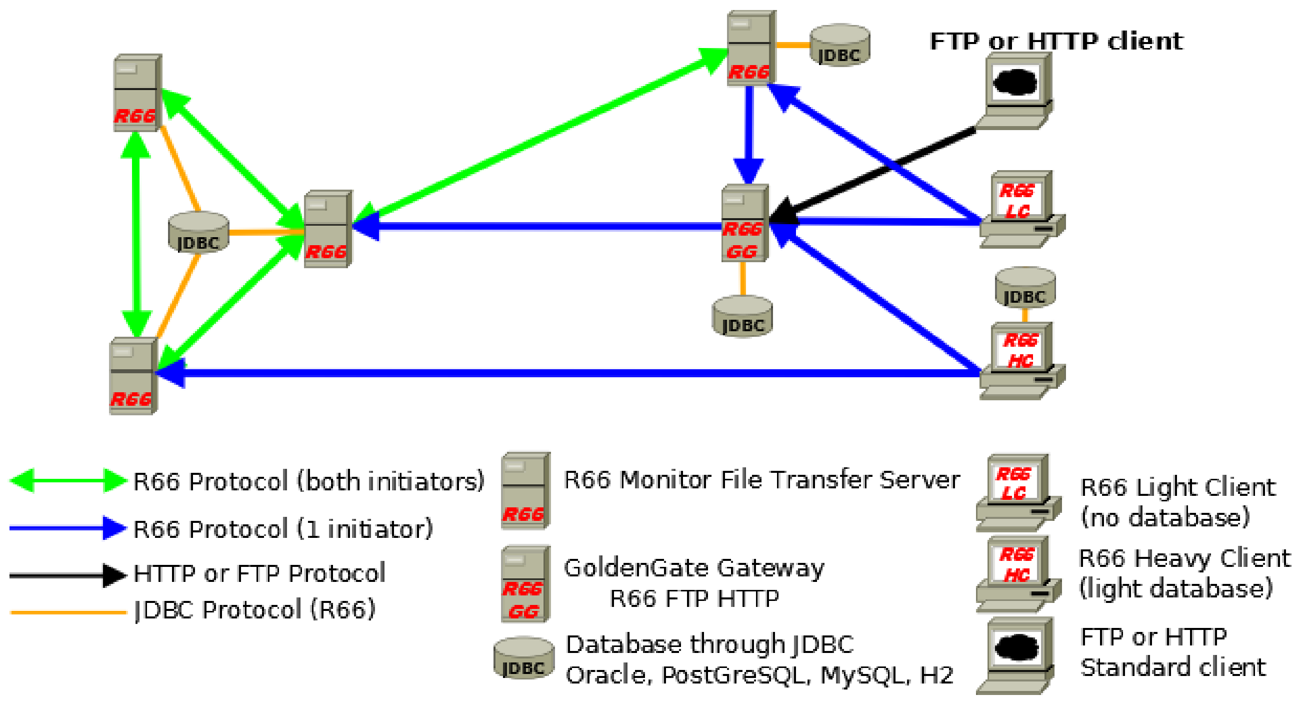
- Installation and parameters
- Integration, additional development
- Support, Maintenance, phone support

http://www.waarp.it

Index

Waarp R66: software for massive file transfer with monitoring: Waarp Route66

This software is a file transfer monitor in protected and powerful contexts.

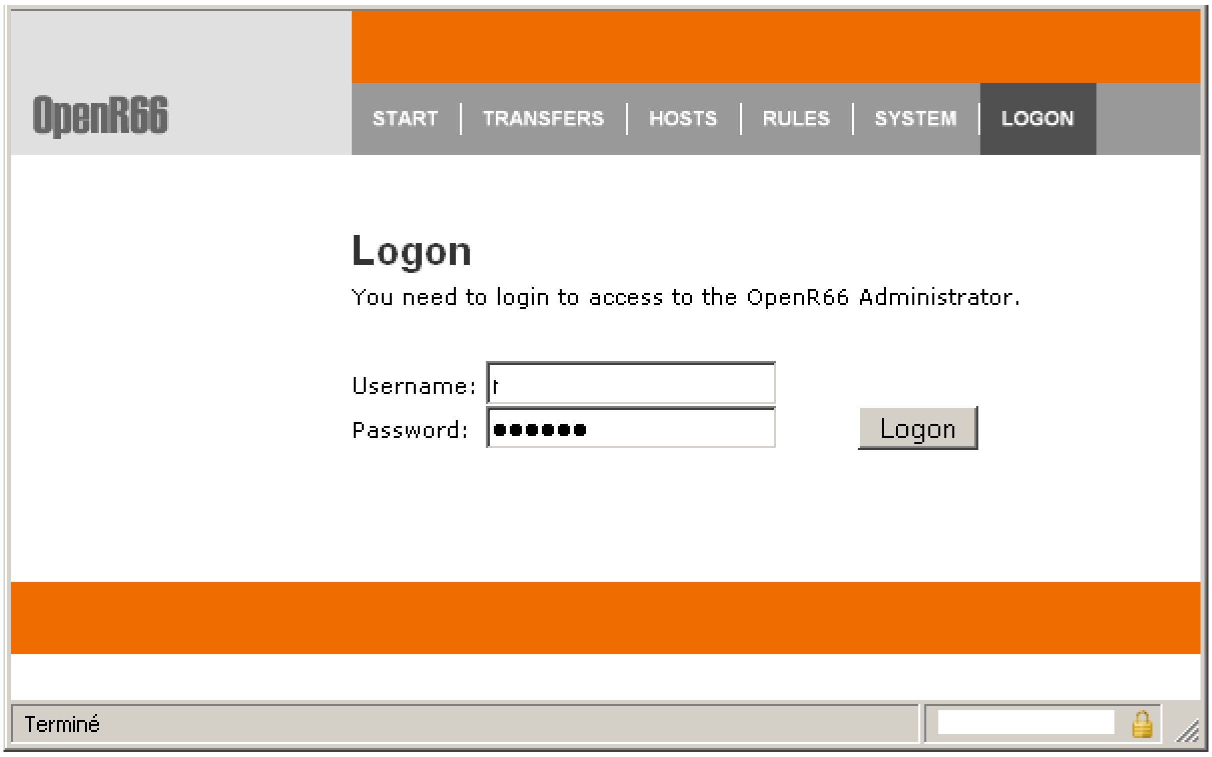

FTP or HTTP client
JDBC
R66
R66 GG
R66 LC
R66 HC
R66 Protocol (both initiators)
R66 Protocol (1 initiator)
HTTP or FTP Protocol
JDBC Protocol (R66)
R66 Monitor File Transfer Server
GoldenGate Gateway R66 FTP HTTP
Database through JDBC Oracle, PostGreSQL, MySQL, H2
R66 Light Client (no database)
R66 Heavy Client (light database)
FTP or HTTP Standard client

Waarp R66 is Massive a File Transfer Monitor as:

- Efficient and secured
- Adaptation to functional needs
- Integration within IT (SNMP compatible)
- Strong Administration and Production tools
- History of transfers, tracked events
- Independent of Server Platforms
- Totally Open Source
- Usable with different kinds of Database, centralized or distributed
- Integration within Waarp file transfer gateway (translating from HTTP/FTP to R66)
    - FTP is available through Waarp Gateway FTP Exec
    - FTP Client is enabled through Tasks
    - HTTP could be made available through explicit specifications in order to provide an adaptation since it is more business dependant
- Integration within an application through extra Java Classes, both on client and server side, allowing Application 2 Application model

This Software has the following properties:

- Full Java, so platform independent (tested under Windows, Linux, AIX)
- Allows to manage up to 2^64 simultaneous transfers
- Uses a database (JDBC: H2, MySQL or Oracle or PostGreSQL indifferently) to retain configuration and log of transfer but is not required for clients to be functional
- Allows the sending and the reception (in Push or Pull mode) between two identified partners
- The transfer is protected (controlled optionally by MD5 or SHA1 or others, with restart of transfer, SSL support)
- Allows the execution of pre operation (before transfer) or of post operation (after transfer) or error operation (after an error occurs)
- Multiplexing of network connections between two servers (firewall compliant)
- Proxy/Reverse Proxy for R66 protocol (for DMZ)
- KeepAlive internal and TimeOut control
- Control on Bandwidth, CPU, simultaneous transfer limitation usage
- Functions of control and statistics, traces for all transfer operations
- Web support for monitoring production
- SNMP compatible as Agent both in pull or notification mode with SNMP V2 and V3 compatibility
- Usual client submitted transfer through database as asynchronous operation
- Synchronous client which directly manages its transfer, using however the database as control
- Synchronous thin client which does not used the database at all (useful for instance for light client on personal computer and not server in production) - those clients are not listening to incoming transfer requests but can only initiates transfers (in push or pull)



Logon

You need to login to access to the OpenR66 Administrator.

Username: r

Password: ●●●●●●●

Download

The source and binary (precompiled jar), as well as the current documentation, can be found at:

http://waarp.github.io/Waarp/index.html

Various packages exist:

Packages

- Waarp Digest
- Waarp Common
- Waarp Exec
- Waarp Snmp
- Waarp XmlEditor
- Waarp Password Gui
- Waarp R66
- Waarp Proxy R66
- Waarp R66 Client GUI
- Waarp FTP
- Waarp Gateway Kernel (R66 linked)
- Waarp Gateway FTP (R66 linked)
- Waarp Thrift (R66 linked)
- Waarp FTP Client (Gateway and R66 linked)
- Waarp WaarpAdministrator (R66 linked)

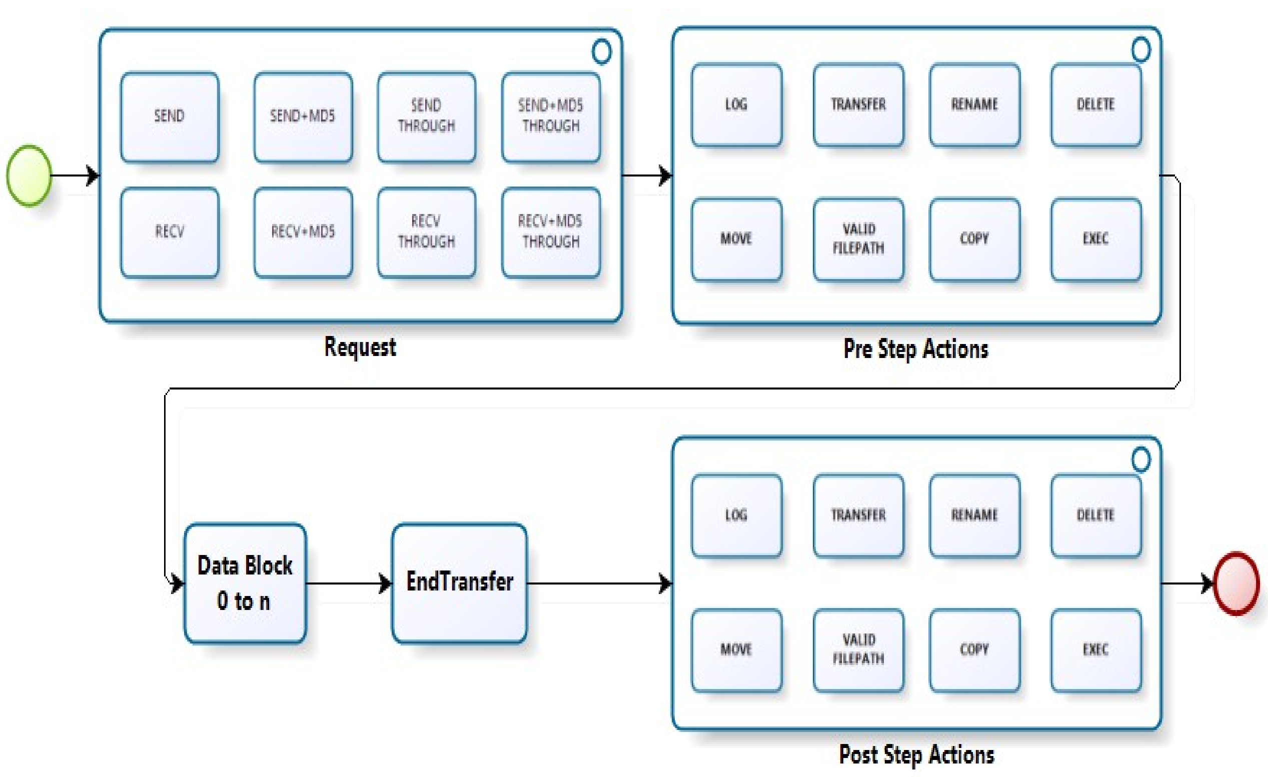
Presentation of Waarp R66: Massive File Transfer Monitor

Efficient and secured

- Unlimited concurrent number of transfer
- Possibility to limit the bandwidth (in point 2 point or globally)
- Possibility to limit the CPU usage and the simultaneous number of transfers
- Guaranty of delivery (persistence through database and retry)
- Virtualization of access paths
- Track of meta-data associated with transfers
- Encrypted connection support (SSL) with optional strong authentication and Packet Integrity support
- Easy integration in security rules (low number of chosen ports, flow multiplexing)
- Partner authentication (through encrypted password and optionally through strong SSL Client Authentication)
- Rules validation of usage by partner
- Possibility to have multiple OpenR66 File Transfer Monitors acting as a single one behind a Load Balancer to enable more reliability and scalability
- Possibility to setup a Proxy/Reverse Proxy for DMZ implantation

Adaptation to functional needs

- Initiator of transfer in different modes SEND or RECV through Rules
- Pre and Post Actions according to transfer Rules associated with each transfer
- Integration through script
- Integration through submission client in Java
- Integration through interaction with the database of Waarp R66
- Integration through a Graphical User Interface (Light Client only)
- Integration through native Java code (business code) either through BusinessFactory or through R66Runnable
    - This model allows the implementation of Application 2 Application MFT
- Transmit in synchronous or asynchronous mode
    - "Query and Forget"
    - "Wait for the end of the transmission"



Pre and Post transfer procedures

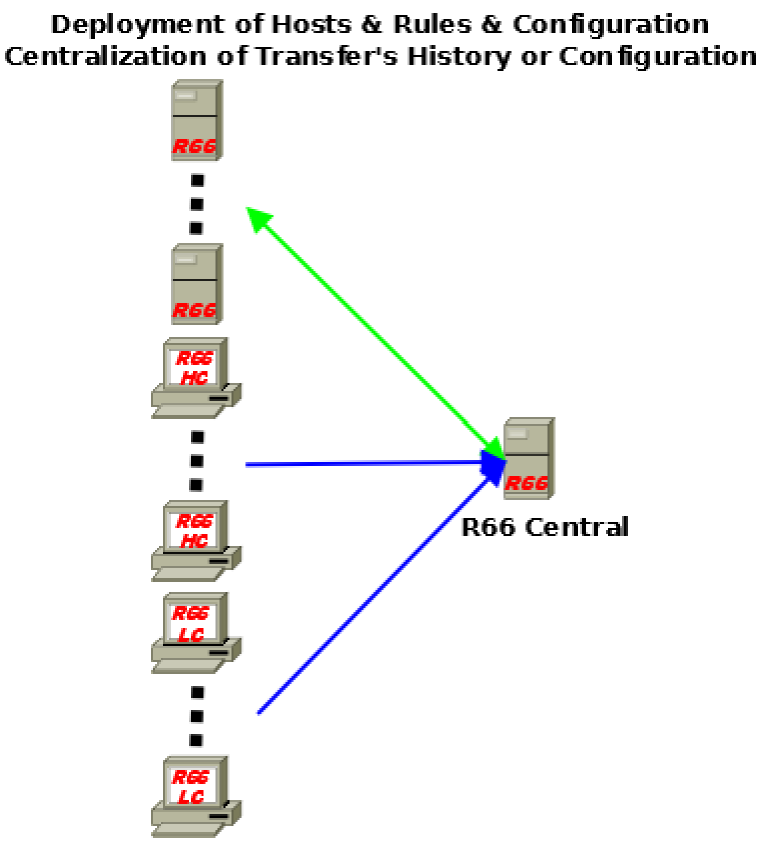
- Native operations
  - `LOG`: Write in the log file the given information
  - `RENAME`: Rename the file
  - `MOVE`: Move the file
  - `VALIDFILEPATH`: Valid the constructed file path
  - `DELETE`: Delete the file
  - `COPY`: Copy the file
  - `TRANSFER`: Transfer again the file to another OpenR66 server
  - `RESCHEDULE`: Reschedule Transfer task if the error code is one of the specified codes and if the new schedule is valid
  - `TAR/ZIP`: To tar or zip according to arguments
  - `TRANSCODE`: ability to transcode from one Charset to another one (as ISO-8859-15 to IBM01147)
  - `SNMP`: ability to send a trap/info from task execution
  - `FTP`: ability to use synchronous transfer in tasks
- External operations

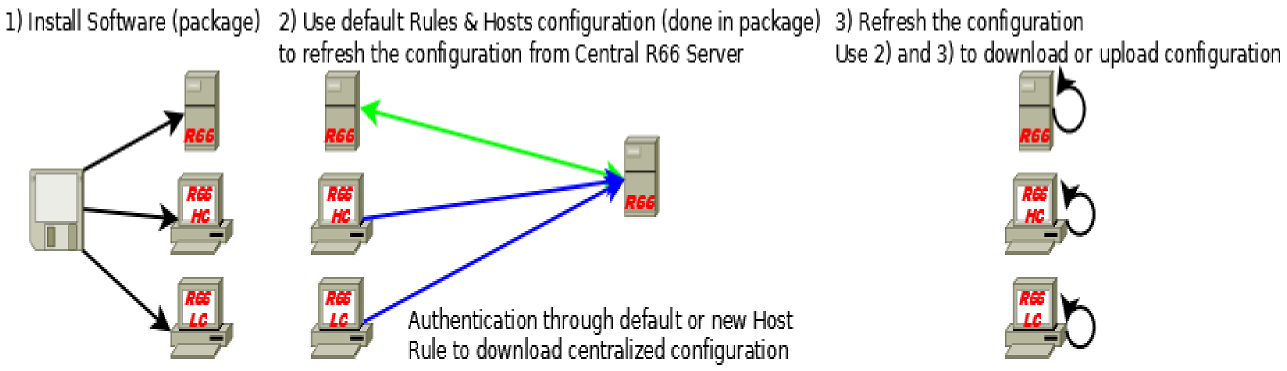
    - `EXEC` (or `EXEC RENAME`): Execute an external procedure (system call) on the basis of args given or constructed (and rename the file according to the result of the command)
    - `EXECOUTPUT`: Execute an external procedure (system call) on the basis of args given or constructed but in case of error, it uses the output to setup information status using

      ```
      <STATUS>status</STATUS><ERROR>error message</ERROR>
      ```
    - `EXECJAVA`: Execute an external procedure (Java Class implementing R66Runnable) on the basis of args given or constructed

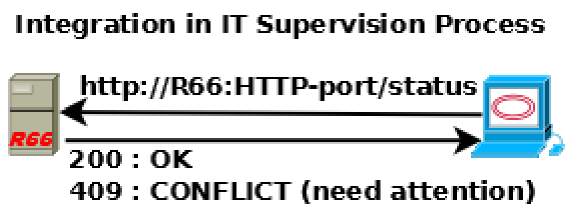
Integration, Administration and Production

- Easily integrated into Security rules
- Easily integrated into a production plan
- Easily integrated into a supervision tool (HTTP/SNMP)
- Allow to block any new request, such that shutdown could be started once all active requests are over.
- Allow to handle the configuration in a central point
- Allow to handler the history of the transfers in a central point
    - In push or pull mode to or from the central server point
    - Updated at the desired rhythm
    - No lock of the OpenR66 monitors in case the central server is unavailable


Deployment of Hosts & Rules & Configuration
Centralization of Transfer's History or Configuration
R66
R66
R66 HC
R66 HC
R66 LC
R66 LC
R66
R66 Central


1) Install Software (package)
2) Use default Rules & Hosts configuration (done in package) to refresh the configuration from Central R66 Server
3) Refresh the configuration
Use 2) and 3) to download or upload configuration
R66
R66 HC
R66 LC
Authentication through default or new Host Rule to download centralized configuration

- HTTPS native interface for the administration with access control
- Allow the administration through script
  - For instance, allow to modify the bandwidth dynamically
- SNMP support as Agent (MIB included) in SNMP V2 or SNMP V3
  - Pull mode (from Monitoring software)
  - Push mode (notification as trap or inform)
- HTTP native interface for the supervision


Integration in IT Supervision Process
http://R66:HTTP-port/status
R66
200 : OK
409 : CONFLICT (need attention)

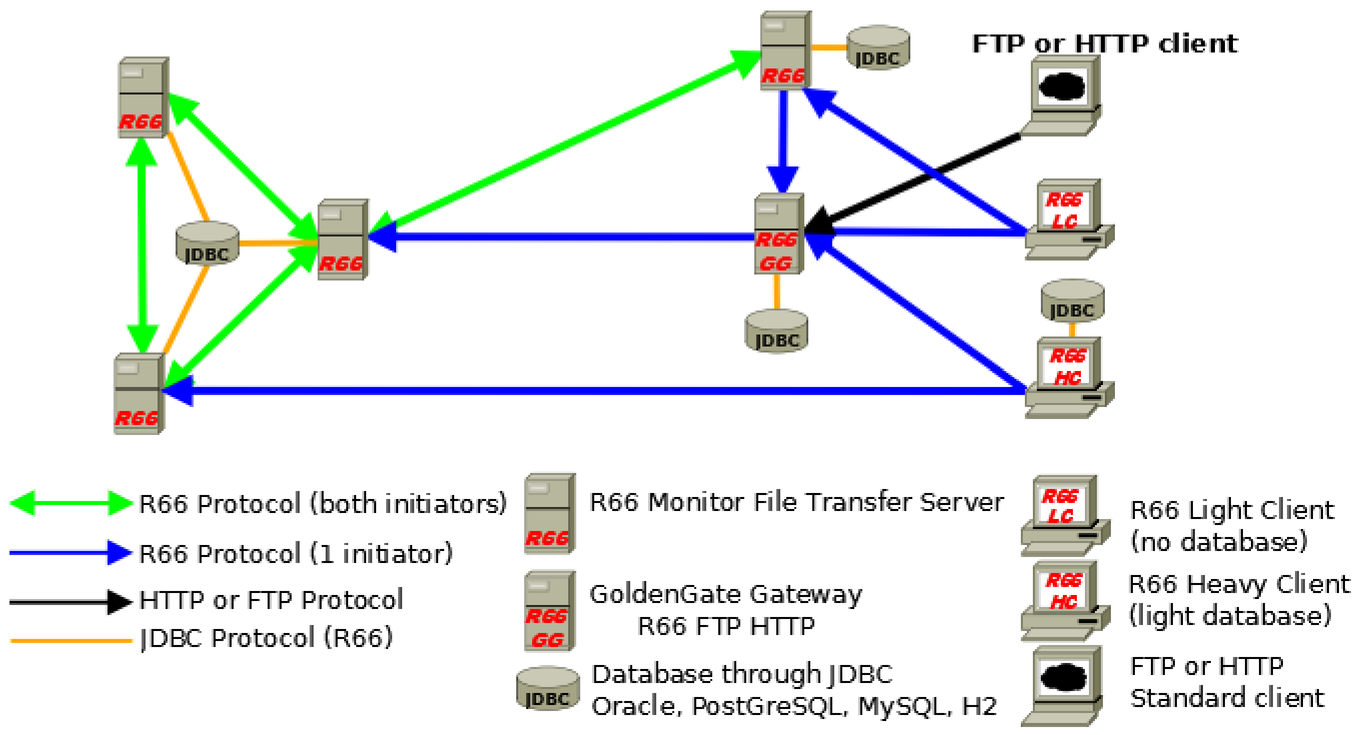
History of transfers

- All transfers are tracked into the database
- The database data can be exported into XML format

Independence with server platforms

- Full Java (minimum JDE 6)
- Tested on Windows, Linux, AIX
  - Some clients report that it runs also under Solaris and ZSeries
- Solution totally Open Source

Usage mode


FTP or HTTP client
R66
JDBC
R66 GG
R66 LC
R66 HC
R66 Protocol (both initiators)
R66 Protocol (1 initiator)
HTTP or FTP Protocol
JDBC Protocol (R66)
R66 Monitor File Transfer Server
GoldenGate Gateway R66 FTP HTTP
Database through JDBC Oracle, PostGreSQL, MySQL, H2
R66 Light Client (no database)
R66 Heavy Client (light database)
FTP or HTTP Standard client

- **Server (R66)**
  - To have all the functionality

  - Possibility to setup a cluster using a loadbalancer in front and sharing some filesystems and the database
- **Heavy Client (R66 HC)**
  - Limited to self-initiated transfers
- **Light Client (R66 LC)**
  - Limited to self-initiated transfers, do not have any database for history (except in log files) neither restart capability

Partners: who are they?

In R66, there are 2 kinds of partners: client and server.

- A **client** can request a transfer (in send or recv), but it cannot be the target of a request since it is supposed to be inactive while not requesting a transfer.
  - A **Light Client** is a real client with no database at all. It can however manage to store the "past" in XML log files for each transfer.
  - A **Heavy Client** is a client with a database support. It can be used as a server if wanted, or only as a Client, meaning inactive while no request is initiated by itself.
- A **server** can request a transfer and can be the target of a request.

In R66, once the request is started, there are 2 roles: requester and requested.

- A **requester** is the one that initiate the request. It could be a client or a server.
- A **requested** is the one that receives a new request from a requester. It could only be a server.

The requester is responsible for the request, since it initiates it. It means that it is the only one that could initiate a restart of this request in case of failure. The main reason is that the request of transfer is generally related to a condition on the requester side. However a requested host could try to request the requester to resend its request. At the time being, it can only be done through the web admin interface.

In R66, once the request is running, there are 2 actors: sender and receiver.

- A **sender** is the one that will send the file.
- A **receiver** is the one that will receive the file.

Considering a rule, sender and receiver will match accordingly the requester and requested:

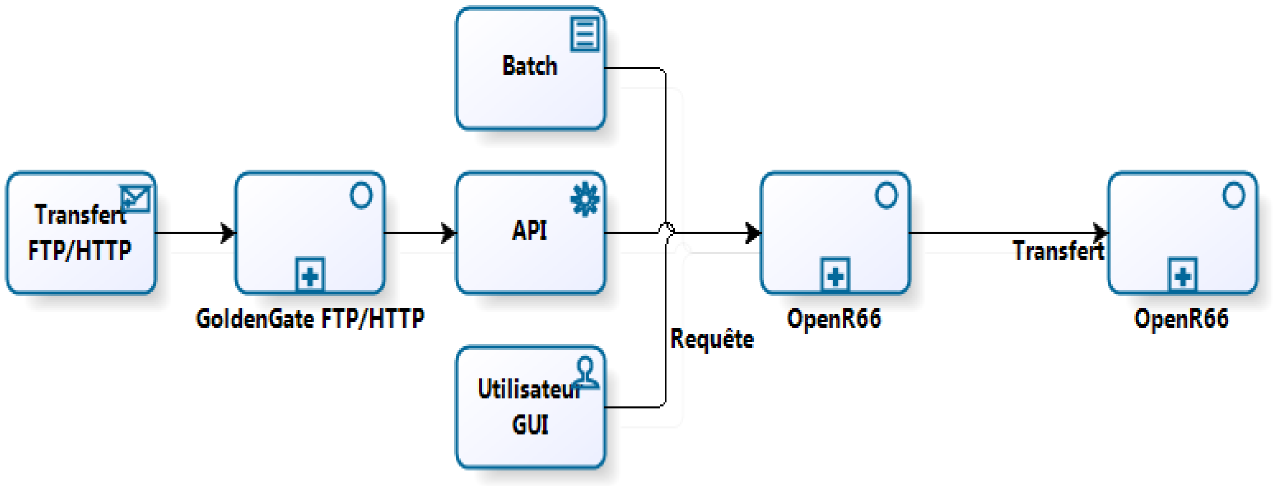
- **SEND** mode rule: ***requester*** = sender and ***requested*** = ***receiver***

- **RECV** mode rule: ***requester*** = ***receiver*** and ***requester*** = ***sender***

This has to be understood correctly when creating the rule, and in particular to understand which part of the "send" or "recv" tasks will be executed by one or the other of the 2 partners involved in the transfer.

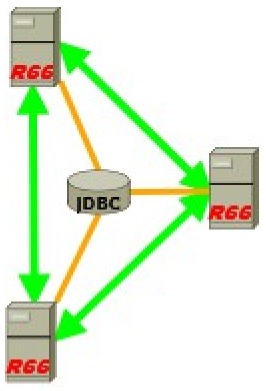
How to operate R66 transfers

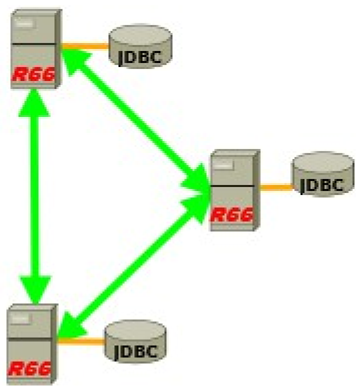
Below is presented the different way to operate R66 transfers.



- **In Batch mode**
  - A script is executed which submits a query of transfer (synchronous or asynchronous)
- **In User mode** through the Graphical Interface (GUI)
  - The transfer request is executed immediately and synchronously
- **In API mode**
  - Used by Waarp Gateway to forward an FTP or HTTP transfer into R66 mode up to its final destination
    - Waarp can be used to generate another action than transfer again in post reception task
  - Can be used to instrument one application to use R66 as file transfer natively (Application 2 Application model)

Usable with many kind of Databases(centralized or distributed)





- Tested with several databases through JDBC JDBC: Oracle, PostGreSQL, MySQL, H2 (for a low footprint on a PC)
- Database schema is sharable between several OpenR66 servers within one Data Center
- In case of a set of Servers acting as a Single One, they do share the same ID and Database and Storage
- Possibility to not rely on any database, but then some functionality will be off (like extended monitoring or the ability to restart transfer in error, except through XML file analysis - if active -)

Some basic definitions and understanding

Often one asks what are requests and tasks and how they are executed.

What is client/server/heavy client

First, you must understand that 2 types of partners exist: client and server.

1. A client

   A client can only be the source of a request (whatever the way, push or pull, send or recv). It cannot be the target of a request. This could be compared to a FTP client which can only initiate a connection, but cannot be the target of a new connection (a part from internal FTP protocol for Data between Passive or Active mode for instance).

2. A server

   A server can ask for new request (initiating a connection) or can be the target of a new request. However, contrary to a FTP server, it can initiate itself a request.

3. An heavy client

   An heavy client is like a client but with a "light" database, meaning that all its logs and management will be in the light database (H2 Database in Java). Note that this mode could be extended to a Server mode, using this database, as for "light" server.

Who is the requester/requested

For a request, we have two kinds of partners:

1. the requester
2. the requested

When a request is initiated, the initiator (named requester) will try to connect to the requested (target). It tries 3 times with waiting time between, until it reaches the target. If the connection is not ok, then after 3 tries, the request is in error. We will see later on what happens then.

If the initiator is a client, the request can only be restarted from the client, since it cannot be the target of a new request (the client is not listening to any new request). That's the main reason why a stopped request can be restarted from initiator only. However, in some conditions, R66 protocol will try to restart a request even from the target, if the initiator is not a client and if it can in fact send a request of restart to this remote original requester partner.

Another reason for the requester to be the root of a restart in case of error, is that most of the time, this is where the request could have much more meanings, either in term of tasks to execute or in term of file preparation or responsibility.

What are rule types

Any request can be in SEND or RECV mode (push or pull).

This are the main drivers to know which tasks will be executed by who.

Once the request is initiated and both partners ok to do so, the sender (could be different than the requester) will execute Send tasks, while the other one will execute the Recv tasks. For a specific rule, if the rule is defined as a SEND mode, the requester will be the sender, while in the opposite if the rule is defined as a RECV mode, the requester will be the receiver.

Examples:

- Host A = requester of a request using a rule SEND
- Host B = requested for the same request (defined also as SEND mode)

The rule will be defined in both partners as follow:

- Mode = SEND
- Recv tasks (pre/post/error) => will be executed by Host B
- Send tasks (pre/post/error) => will be executed by Host A

The opposite rule in RECV mode will be as follow:

- Mode = RECV
- Recv tasks (pre/post/error) => will be executed by Host A
- Send tasks (pre/post/error) => will be executed by Host B

The independence of the configuration for server A and server B ensure that locally the rules are defined according the local needs. Note however that if the 2 servers share the same database for their respective configuration, they will share also the rule definitions and therefore the task definitions.

Using the local configuration information (local to each partner), as the directory configuration, the local resolution of variables (as host name, file name, ID, ...), it is therefore possible to share the very same rule definitions with several partners, while they will act differently locally. For instance, both partners could reference the very same name of external script, but those scripts will be different and adapted to the local context and therefore acting differently.

What happens with the task

Once the role of each partner is determine (sender or receiver), the tasks will be executed as follow:

1. Pre tasks on requested side
2. Pre tasks on requester side
3. File transfer
4. Post tasks on receiver side
5. Post tasks on sender side

In case of error, whatever the side, the error message is sent back to the partner and both sides execute the Error tasks.

What are the synchronous/asynchronous mode

Two modes are available:

- Synchronous: the requester will wait until the end of the transfer to finish the request operation
- Asynchronous: the requester will just submit the request and forget it. It could check later on the result or using tasks to be informed back on the result on the request.

For the asynchronous mode, the command line SubmitTransfer has to be used.

For the synchronous mode (or to come back to a synchronous operation once launched using asynchronous mode):

- Command: DirectTransfer
- SubmirTransfer + RequestTransfer
- SubmitTransfer + post operations to call back the caller of SubmitTransfer

What are the various usages of folders

- Work: the directory where the temporary received file will be located (_can be specified per Rule_)
- In: the directory where the received files will be placed once fully received (and before any post TASK) (_can be specified per Rule_)
- Out: the directory where the to be sent files could be placed (before any pre TASK) if the path is relative (the out file can be specified with a full path, then ignoring the "Out" directory) (_can be specified per Rule_)
- Arch: the directory where the archived files will be placed (for instance log or configuration export) (_can be specified per Rule_)
- Conf: the directory where the configuration files for no-database client/server

Installation

The primary installation of a server is done using the following steps:

1. Download the “***All Jars Waarp R66***” package from:

   http://waarp.github.io/Waarp/Downloads.html

2. Create the XML configuration files (Server, Limit, Rule, Authent, SNMP). The minimal configuration is Server and Authent (eventually limited to the created server).

   An example of the various configuration files can be found in the “Waarp R66 Configuration example” package from:

   https://raw.github.com/waarp/WaarpMaven2/master/AllJars/WaarpR66-example-config.zip

3. Create the necessary GGP (Waarp Password files) using the des Key generated both by Waarp Password

4. Create the necessary SSL KeyStore for the HTTPS Administration interface and another one for the SSL server authentication (server only, or server AND client, depending on your choices)

5. Create the database (dependent on the one choose between Oracle, PostgreSql, MySQL or H2 database, the initial creation is out of this documentation).

6. Initialize or update the database (once created and the user/password and the JDBC access fill in the Server configuration XML file)

7. Create the logback.xml file according to your need (to setup for instance the level of trace from DEBUG to WARN). See the logback.xml example or the LogBack web site.

8. Now you can run the server and use the various commands or Administration interface to continue the configuration (Limit, Rule, Authent).

Below those several steps are illustrated.

Note that the various tools are consolidated into one named: Central Administrator (Xml Editor, Password, R66Gui) plus some other functions.

XML Configuration

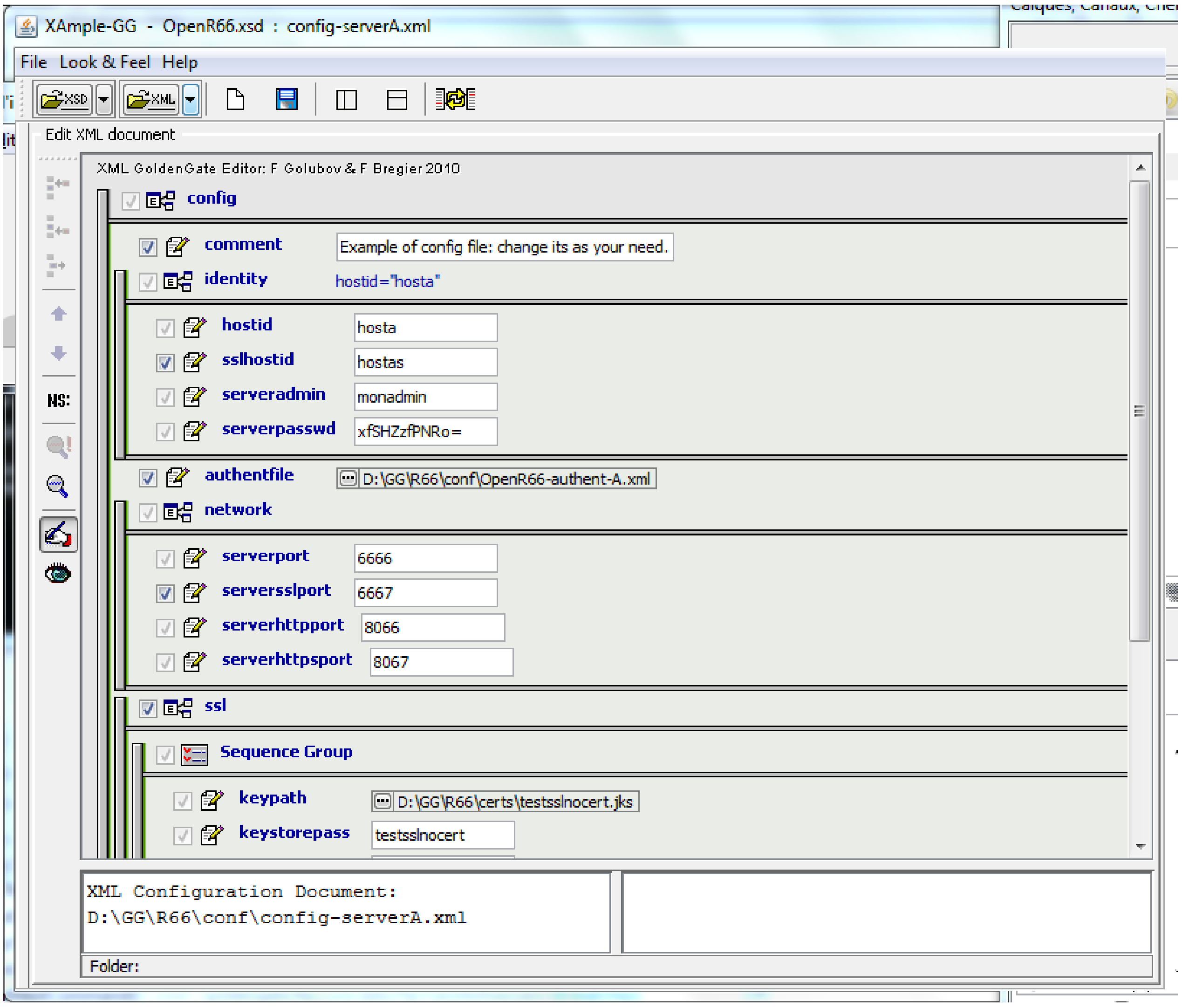
To configure a Waarp R66 Server, you need to first create the XML files as needed (see the Xml Config files page): main config file, authent file, rules and limit config files.

A specific package contains default example configuration files. This package is named "***Waarp R66 configuration example***" and can be found here:

https://raw.github.com/waarp/WaarpMaven2/master/AllJars/WaarpR66-example-config.zip

A definition of those XML files can be found in XML configuration.

To help the administrator to generate correct files, XSD models are defined to be used with an extension of the project Xample (XML Gui Editor) from Felix Golubov, see Waarp Xml Editor.


XAmple-GG - OpenR66.xsd : config-serverA.xml
File Look & Feel Help
XSD
XML
Edit XML document
NS:
XML GoldenGate Editor: F Golubov & F Bregier 2010

config

comment	Example of config file: change its as your need.
identity	hostid="hosta"
hostid	hosta
sslhostid	hostas
serveradmin	monadmin
serverpasswd	xfSHZzfPNRo=
authentfile	D:\GG\R66\conf\OpenR66-authent-A.xml
network	
serverport	6666
serversslport	6667
serverhttpport	8066
serverhttpsport	8067
ssl	
Sequence Group	
keypath	D:\GG\R66\certs\testsslnocert.jks
keystorepass	testsslnocert

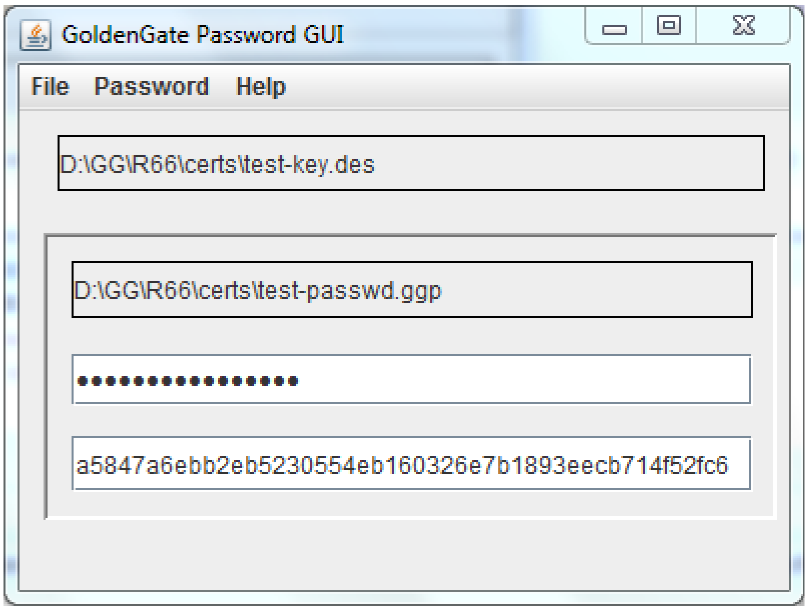
```
XML Configuration Document:
D:\GG\R66\conf\config-serverA.xml
```

Folder:

It is also possible to use JAXE with jaxe-xxx-config.xml files.

Password configuration

For the administrator you need a specific DES Key for the encrypted password support. DES encryption support and generation are available through the GoldenGate Password GUI project.



D:\GG\R66\certs\test-key.des

D:\GG\R66\certs\test-passwd.ggp

••••••••••••••••

a5847a6ebb2eb5230554eb160326e7b1893eecb714f52fc6

Note that under **Mainframe**, some people reports that adding **`-Dfile.encoding=UTF8`** fixes some issues with passwords.

See Waap Password Tool

Cryptography

For the administrator you need a KeyStore containing a RSA key for SSL support in HTTPS.

You need another KeyStore for the SSL support in the Waarp R66 protocol, but only if you want to use this SSL channel.

If you don't want that OpenR66 uses SSL between two hosts, then you only need one KeyStore for the HTTPS support.

In the source, you will find in the certs directory 2 such keystore named admin66.store and openr66.store. You can named the files as you wanted (using the standard "jks" extension for instance).

To generate those files, you can also ue the keytool command from the jdk, or using the cool free tool KeyTool IUI (last known version in 2.4.1).

- `keypath` for OpenR66 channels (`admkeypath` for administrator) is the full or relative path to the keystore file
- `keystorepass` (`admkeystorepass`) is the password for the keystore

- `keypass` (`admkeypass`) is the password for the key

For instance:

```
keytool -genkey -alias myalias -keyalg RSA -validity xxx -keystore mystore.store
```

Below you will find the full detail on how to create those KeyStore and TrustedKeyStore according to your needs:

To generate the stores for Waarp R66 for instance, you need to create 2 JKS keyStore. To generate those files, you can use the "keytool" command from the JDK or using the free tool KeyTool IUI (last known version in 2.4.1). Below we show how to use the Keytool IUI.

For SSL connection without authentication of clients

- Server side (also valid for Administration HTTPS site)
  - Create one jks <u>KeyStore</u> (server.jks) with one Private Key Version 3 using RSA algorithm.
  - Use this KeyStore as <u>KeyStore</u> for the Server.
  - To do that, suing KeyTool IUI:
    1. Create an empty KeyStore
    2. Create a Private Key Version 3 with RSA algorithm added to this KeyStore
- Client side
  - Create one jks <u>TrustStore</u> (clientTrust.jks) with the Certificate Chain of the Server Key from the Server as a Regular Certificate.
  - Use this KeyStore as <u>Trustore</u> for the Client.
  - To do that, using KeyTool IUI:
    1. From the Server Keytore, export Private Key (2 files with one Certificate Chain)
    2. Create one jks TrustStore (Empty KeyStore)
    3. Import the Trusted Certificate as Regular Certificate (Certificate Chain as .der file)

For SSL connection with authentication of clients

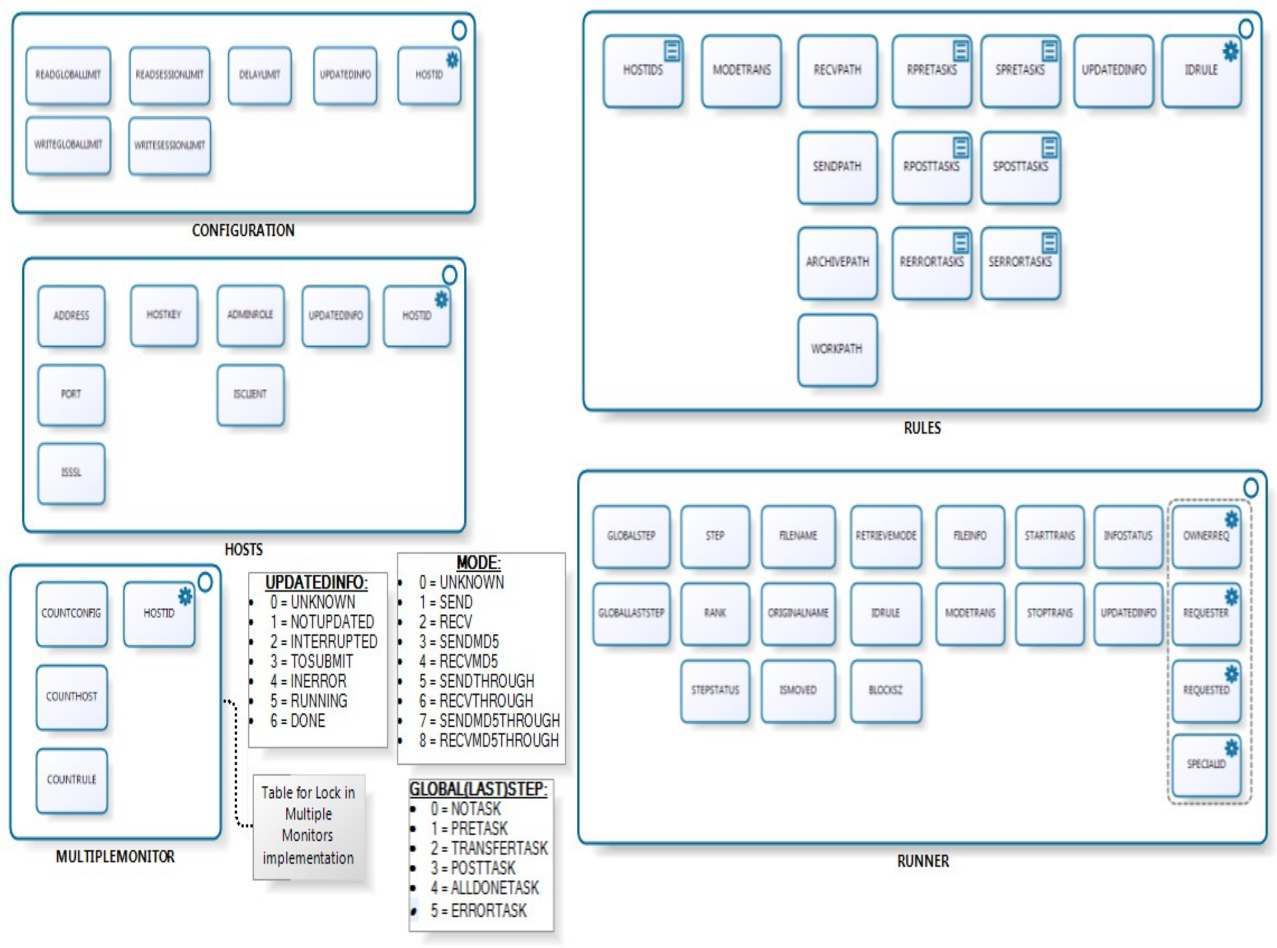
- First do as without authentication of clients for the server authentication side: server.jks for the Server's <u>KeyStore</u> and clientTrust.jks for the Client's <u>TrustStore</u>.
- Handle the reverse authentication of multiples clients within the server
  - Client side
    - Create one jks KeyStore with one Private Key Version 3 using RSA algorithm for each Client (client1.jks, client2.jks, clientn.jks).
    - Use this KeyStore as KeyStore for the Client.
    - To do that, using KeyTool IUI:

        1. Create an empty KeyStore
        2. Create a Private Key Version 3 with RSA algorithm added to this KeyStore
  - Server side
    - Create one jks TrustStore (serverTrust.jks) with the Certificate Chain of the Client Key from the Client as a Regular Certificate.
    - Use this KeyStore as Trustore for the Server.
    - To do that, using KeyTool IUI:
        1. Create one jks TrustStore (Empty KeyStore)
        2. From the Client Keytore, export Private Key (2 files with one Certificate Chain)
        3. Import the Trusted Certificate as Regular Certificate (Certificate Chain as .der file)
        4. Import all Trusted Certificates as with 2 and 3 in the same TrustStore

Then you have to fill the ports to use (serverssl port for Waarp R66 channels, serverhttpsport for the administrator) and to set up the good path to every components (don't forget the path to the html admin files, between admin or admin2 - the second being more dynamic -).

Database configuration

Then, you create a database entry in the model you choose (currently supported is Oracle, PostGreSQL, MySQL and H2 Database). This database entry should be referenced in the XML config files. Below is presented the schema of the database that will be created.



You need to get all the necessary jar (see the dependencies of the project) to allow you to launch the server. An example of shell script including all jar is presented in the src/main/example directory.

Then you launch the ServerInitDatabase (see the details in ServerInitDatabase).

Mainly, one has to create the database (`-initdb`), and to provide various elements to fill the database (`-loadBusiness`, `-loadAlias`, `-loadRoles`, `-dir`, `-auth`, `-limit`) or using the upgrade procedure if the database already exists from a previous version (from 2.4.17) using `-upgradeDb` option.

Note that multiple Waarp R66 Servers may shared the exactly same database. Waarp R66 is compatible with such a model. Note however that Rules are strictly shared, which means that you will need to take care of the absolute path of EXEC commands such that they are valid on all Waarp R66 servers where it should be executed.

Logging

Example of logback.xml file that provides both logging to a file (R66Server.log) and to stdout, with an historic on 30 items in zip format, with a limit of 1 per day or 10MB each.

```
<configuration>
  <appender name="FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/GG/R66/log/R66Server.log</file>
    <append>true</append>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>/GG/R66/log/R66Server.%d{yyyy-MM-dd}.
%i.log.zip</fileNamePattern>
      <maxHistory>30</maxHistory>
      <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
          <maxFileSize>10MB</maxFileSize>
      </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>
        %date{dd/MM/yyyy/HH: mm: ss.SSS} %level [%logger] [%thread] %msg%n
      </pattern>
    </encoder>
  </appender>
  <appender name="STDOUT"
    class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <Pattern>%date{dd/MM/yyyy/HH: mm: ss.SSS} %level [%logger] [%thread] %msg
%n</Pattern>
    </encoder>
  </appender>

  <root>
    <level value="warn" />
    <appender-ref ref="FILE" />
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

And use the **`-Dlogback.configurationFile=J:/GG/R66/conf/logback.xml`** as argument to the java command (before the class to launch, roughly were you put the `-Xms -Xmx` arguments).

Now you can launch the Waarp R66 Server.

XML configuration

Every command require an XML configuration file as first argument. This file contains some information that are really relative to the host. Those informations are described shortly in the following pages.

For all those configuration files, there are 2 kind of helpers, that can be found in xample directory under src/main/xample or in Waarp R66 configuration example zip file under xample directory:

- OpenR66-XXX.xsd : XML schema that could be used with any particular XML Schema compatible editor, and in particular the Waarp Xml Editor
- jaxe-OpenR66-XXX.xml : an equivalent of the XML schema that could be used with Jaxe editor (http://jaxe.sourceforge.net)

Waarp R66 Server Configuration File

`config`

- `comment`: string Optional

`identity`

- `hostid`: nonEmptyString

  Host ID in NON SSL mode
- `sslhostid`: nonEmptyString Optional

  Host ID in SSL mode
- `cryptokey`: Des-File

  Des CryptoKey File containing the key in Des mode for R66 passwords
- `authentfile`: XML-File Optional

  Authentication File containing Authentications for partners

`server`

- `serveradmin`: nonEmptyString

  Username for Administrator access
- `serverpasswd`: nonEmptyString or `serverpasswdfile`: nonEmptyString

  Password for Administrator access using Crypto Key (directly or through a ggp file)
- `usenossl`: booleanType default="True"

  True (Default) if R66 will allow no SSL mode connection
- `usessl`: booleanType default="False"

  True if R66 will allow SSL mode connection
- `usehttpcomp`: booleanType default="False"

  True if Administrator (HTTPS) will allow HTTP compression
- `uselocalexec`: booleanType default="False"

  By default, use the System.exec() but can lead to limitation in performances (JDK limitation). The usage of the Waarp LocalExec Daemin tends to reach better performances through execution delegation
- `lexecaddr`: address default="127.0.0.1" Optional

  Address of LocalExec Daemon
- `lexecport`: nonNulInteger default="9999" Optional

  Port of LocalExec Daemon
- `httpadmin`: directoryType

  Home Directory for HTTPS file support for Administrator
- `admkeypath`: JKS-File

JKS HTTP KeyStore for Administrator Access in HTTPS Mode (containing Server Certificate)

- `admkeystorepass`: nonEmptyString

  Password for the HTTP KeyStore

- `admkeypass`: nonEmptyString

  Password for the HTTP Server Certificate within the HTTP KeyStore

- `checkaddress`: booleanType default="False" Optional

  True if R66 will check remote IP address while accepting a new connection

- `checkclientaddress`: booleanType default="False" Optional

  True if R66 will check remote IP address also for remote Client

- `multiplemonitors`: nonNullInteger default="1"

  Set the number of servers that act in the same group as a single instance of OpenR66 File Transfer Monitor

`network`

- `serverport`: nonNullInteger default="6666"

  Port used in NON SSL mode

- `serversslport`: nonNullInteger Optional default="6667"

  Port used in SSL mode

- `serverhttpport`: nonNullInteger default="8066"

  Port used for monitoring in HTTP mode

- `serverhttpsport`: nonNullInteger default="8067"

  Port used for Administrator access in HTTPS mode

`ssl` Optional

- `keypath`: JKS-File

  JKS KeyStore for R66 Access in SSL Mode (containing Server Certificate)

- `keystorepass`: nonEmptyString

  Password for the KeyStore

- `keypass`: nonEmptyString

  Password for the Server Certificate within the KeyStore

- `trustkeypath`: JKS-File

  JKS Trust KeyStore for R66 Access in SSL Mode with Client Authentication (containing Client Certificate)

- `trustkeystorepass`: nonEmptyString

  Password for the Trust KeyStore

- `trustuseclientauthenticate`: booleanType default="False"

True if R66 will only allow client through SSL authentication

directory

- `serverhome`: directoryType

  Home Directory for the OpenR66 Server (relative path to this Home)
- `in`: nonEmptyString Optional default="IN"

  Default Receive Directory
- `out`: nonEmptyString Optional default="OUT"

  Default Send Directory
- `arch`: nonEmptyString Optional default="ARCH"

  Default Archive Directory
- `work`: nonEmptyString Optional default="WORK"

  Default Working Directory
- `conf`: nonEmptyString Optional default="CONF"

  Configuration Directory

limit

- `serverthread`: nonNullInteger default="8" Optional

  Number of Threads on Server side (=Number of Cores)
- `clientthread`: nonNullInteger default="80" Optional

  Number of Threads on Client side (=10xServer)
- `memorylimit`: nonNullInteger default="4000000000" Optional

  Memory Limit for R66 Server Java Process
- `sessionlimit`: nonNegInteger default="8388608" Optional

  HBandwidth for one session (64Mb)
- `globallimit`: nonNegInteger default="67108864" Optional

  Global Bandwidth (512Mb)
- `delaylimit`: nonNullInteger default="10000" Optional

  Delay between 2 checks of Bandwidth (10s). The less this value, the better the bandwidth limitation is done. However take care to not give too low value
- `runlimit`: nonNullInteger default="10000" Optional

  Limite by batch of active transfers (10000)
- `delaycommand`: nonNullInteger default="5000" Optional

  Delay between 2 execution of the Commander (5s)
- `delayretry`: nonNullInteger default="30000" Optional

  Delay between 2 attemps in case of error (30s)
- `timeoutcon`: nonNullInteger default="30000" Optional

Delay before a Time Out occurs (30s)

- `blocksize`: nonNulInteger default="65536" Optional

  Block size (64Ko). A value between 8 KB to 16 MB is recommanded

- `gaprestart`: nonNulInteger default="30" Optional

  Gap to use when restarting a transfer as gap x blocksize

- `usenio`: booleanType default="False" Optional

  Usage of NIO support for the files. According to the JDK, it can enhance the performances

- `usecpulimit`: booleanType default="False" Optional

  Usage of CPU Limitation when new request starts

- `usejdkcpulimit`: booleanType default="False" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `cpulimit`: decimalType default="0.0" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `connlimit`: nonNegInteger default="0" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `digest`: integerNotNegative default=0 (to setup a different Digest than MD5 globally)

- `usefastmd5`: booleanType default="True"

  Usage of FastMD5 library. The library greatly improves the performances for the MD5 computations

- `fastmd5`: SODLL-File Optional

  Library JNI Filepath. If the path is empty, the pure Java version will be used

- `checkversion`: booleanType Optional default="False"

  If True, it will enable extended protocol (>= 2.3) to enable some extra information getting back at end of transfers

- `globaldigest`: : booleanType Optional default="True"

  If True, it will enable a global digest (MD5, SHA1, ...) per file per transfer

`db`

- `dbdriver`: address

  4 types of database are currently supported: oracle, mysql, postgresql, h2

- `dbserver`: normString

  Connection to the database in JDBC mode (jdbc: type://[host: port]....). Use the database documentation to find the correct syntax for the JDBC connection

- `dbuser`: address

  Database User
- `dbpasswd`: nonEmptyString

  Database User's Password
- `taskrunnernodb`: booleanType Optional default="False"

  When server with no DB, do R66 will use XML files as permanent information on Transfer Tasks

`business`

- `businessid`: nonEmptyString

  the host ids (1 by 1) that will be allow to use Business Request (>= 2.3)
- `roles`
  - `role`

  If specified for one host, this will override database roles for ALL hosts (from version 2.4.9). By default, local server should be added as role = FULLADMIN.
    - `roleid`: nonEmptyString

      The host ids (1 by 1) that will override database roles
    - `roleset`: nonEmptyString with separators as blank or '|'

      The role assign to this host between NOACCESS, READONLY, TRANSFER, RULE, HOST, LIMIT, SYSTEM, LOGCONTROL, PARTNER(READONLY, TRANSFER), CONFIGADMIN(PARTNER, RULE, HOST), FULLADMIN(CONFIGADMIN, LIMIT, SYSTEM, LOGCONTROL)

`aliases`

- `alias`

  This will allow alias usage for host ids (from version 2.4.12).
  - `realid`: nonEmptyString

    The real host id that will have aliases (locally)
  - `aliasid`: nonEmptyString with separators as blank or '|'

    The set of aliases assign to this host

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns: x0="http://www.w3.org/2001/XMLSchema">
    <comment>Example of config file: change its as your need.</comment>
    <identity>
        <hostid>hosta</hostid>
        <sslhostid>hostas</sslhostid>
        <cryptokey>D:\GG\R66\certs\test-key.des</cryptokey>
        <authentfile>D:\GG\R66\conf\OpenR66-authent-A.xml</authentfile>
    </identity>
    <server>
```

```
        <serveradmin>monadmin</serveradmin>
        <serverpasswd>c5f4876737cf351a</serverpasswd>
        <usenossl>True</usenossl>
        <usessl>False</usessl>
        <usehttpcomp>False</usehttpcomp>
        <uselocalexec>False</uselocalexec>
        <httpadmin>D:\NEWSOURCE\WaarpR66\src\main\admin</httpadmin>
        <admkeypath>D:\GG\R66\certs\testsslnocert.jks</admkeypath>
        <admkeystorepass>testsslnocert</admkeystorepass>
        <admkeypass>testalias</admkeypass>
        <checkaddress>False</checkaddress>
        <checkclientaddress>False</checkclientaddress>
    </server>
    <network>
        <serverport>6666</serverport>
        <serversslport>6667</serversslport>
        <serverhttpport>8066</serverhttpport>
        <serverhttpsport>8067</serverhttpsport>
    </network>
    <ssl>
        <keypath>D:\GG\R66\certs\testsslnocert.jks</keypath>
        <keystorepass>testsslnocert</keystorepass>
        <keypass>testalias</keypass>
        <trustkeypath>D:\GG\R66\certs\testcert.jks</trustkeypath>
        <trustkeystorepass>testcert</trustkeystorepass>
        <trustuseclientauthenticate>True</trustuseclientauthenticate>
    </ssl>
    <directory>
        <serverhome>d:/GG/R66</serverhome>
        <in>in</in>
        <out>out</out>
        <arch>arch</arch>
        <work>work</work>
        <conf>conf</conf>
    </directory>
    <limit>
        <usefastmd5>True</usefastmd5>
        <fastmd5>D:\NEWJARS\gglib\win32\MD5.dll</fastmd5>
        <delayretry>10000</delayretry>
        <timeoutcon>10000</timeoutcon>
        <serverthread>8</serverthread>
        <clientthread>80</clientthread>
        <memorylimit>4000000000</memorylimit>
        <sessionlimit>8388608</sessionlimit>
        <globallimit>67108864</globallimit>
        <delaylimit>10000</delaylimit>
        <runlimit>10000</runlimit>
        <delaycommand>5000</delaycommand>
        <blocksize>65536</blocksize>
        <gaprestart>30</gaprestart>
        <usenio>False</usenio>
        <usecpulimit>False</usecpulimit>
        <usejdkcpulimit>False</usejdkcpulimit>
        <cpulimit>0.0E1</cpulimit>
        <connlimit>0</connlimit>
    </limit>
    <db>
        <dbdriver>h2</dbdriver>
        <dbserver>jdbc: h2:
D:/GG/R66/data/openr66;IFEXISTS=TRUE;MODE=Oracle;AUTO_SERVER=TRUE</dbserver>
        <dbuser>openr66</dbuser>
        <dbpasswd>openr66</dbpasswd>
    </db>
</config>
```

Waarp R66 Client Configuration File

`config`

- `comment`: string Optional

`identity`

- `hostid`: nonEmptyString

  Host ID in NON SSL mode

- sslhostid: nonEmptyString Optional

  Host ID in SSL mode

- `cryptokey`: Des-File

  Des CryptoKey File containing the key in Des mode for R66 passwords

- `authentfile`: XML-File Optional

  Authentication File containing Authentications for partners

`client` Optional

- `taskrunnernodb`: booleanType Optional default="False"

  When client with no DB, do R66 will use XML files as permanent information on Transfer Tasks

`ssl` Optional

- `keypath`: JKS-File

  JKS KeyStore for R66 Access in SSL Mode (containing Server Certificate)

- `keystorepass`: nonEmptyString

  Password for the KeyStore

- `keypass`: nonEmptyString

  Password for the Server Certificate within the KeyStore

- `trustkeypath`: JKS-File

  JKS Trust KeyStore for R66 Access in SSL Mode with Client Authentication (containing Client Certificate)

- `trustkeystorepass`: nonEmptyString

  Password for the Trust KeyStore

- `trustuseclientauthenticate`: booleanType default="False"

  True if R66 will only allow client through SSL authentication

`directory`

- `serverhome`: directoryType

  Home Directory for the OpenR66 Server (relative path to this Home)

- `in`: nonEmptyString Optional default="IN"

  Default Receive Directory

- `out`: nonEmptyString Optional default="OUT"

  Default Send Directory
- `arch`: nonEmptyString Optional default="ARCH"

  Default Archive Directory
- `work`: nonEmptyString Optional default="WORK"

  Default Working Directory
- `conf`: nonEmptyString Optional default="CONF"

  Configuration Directory

`limit`

- `serverthread`: nonNulInteger default="8" Optional

  Number of Threads on Server side (=Number of Cores)
- `clientthread`: nonNulInteger default="80" Optional

  Number of Threads on Client side (=10xServer)
- `memorylimit`: nonNulInteger default="4000000000" Optional

  Memory Limit for R66 Server Java Process
- `sessionlimit`: nonNegInteger default="8388608" Optional

  HBandwidth for one session (64Mb)
- `globallimit`: nonNegInteger default="67108864" Optional

  Global Bandwidth (512Mb)
- `delaylimit`: nonNulInteger default="10000" Optional

  Delay between 2 checks of Bandwidth (10s). The less this value, the better the bandwidth limitation is done. However take care to not give too low value
- `runlimit`: nonNulInteger default="10000" Optional

  Limit by batch of active transfers (10000)
- `delaycommand`: nonNulInteger default="5000" Optional

  Delay between 2 execution of the Commander (5s)
- `delayretry`: nonNulInteger default="30000" Optional

  Delay between 2 attemps in case of error (30s)
- `timeoutcon`: nonNulInteger default="30000" Optional

  Delay before a Time Out occurs (30s)
- `blocksize`: nonNulInteger default="65536" Optional

  Block size (64Ko). A value between 8 KB to 16 MB is recommanded
- `gaprestart`: nonNulInteger default="30" Optional

  Gap to use when restarting a transfer as gap x blocksize
- `usenio`: booleanType default="False" Optional

Usage of NIO support for the files. According to the JDK, it can enhance the performances

- `usecpulimit`: booleanType default="False" Optional

  Usage of CPU Limitation when new request starts

- `usejdkcpulimit`: booleanType default="False" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `cpulimit`: decimalType default="0.0" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `connlimit`: nonNegInteger default="0" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `digest`: integerNotNegative default=0 (to setup a different Digest than MD5 globally)

- `usefastmd5`: booleanType default="True"

  Usage of FastMD5 library. The library greatly improves the performances for the MD5 computations

- `fastmd5`: SODLL-File Optional

  Library JNI Filepath. If the path is empty, the pure Java version will be used

- `checkversion`: booleanType Optional default="False"

  True to allow the extended protocol on end of request (>= 2.3)

- `globaldigest`: : booleanType Optional default="True"

  If True, it will enable a global digest (MD5, SHA1, ...) per file per transfer

db Optional

- `dbdriver`: address

  4 types of database are currently supported: oracle, mysql, postgresql, h2

- `dbserver`: normString

  Connection to the database in JDBC mode (jdbc: type://[host: port]....). Use the database documentation to find the correct syntax for the JDBC connection

- `dbuser`: address

  Database User

- `dbpasswd`: nonEmptyString

  Database User's Password

- `taskrunnernodb`: booleanType Optional default="False"

  When client with no DB, do R66 will use XML files as permanent information on Transfer Tasks

business

- `businessid`: nonEmptyString

  the host ids (1 by 1) that will be allow to use Business Request (>= 2.3)

aliases

- `alias`

  This will allow alias usage for host ids (from version 2.4.12).

  - `realid`: nonEmptyString

    The real host id that will have aliases (locally)

  - `aliasid`: nonEmptyString with separators as blank or '|'

    The set of aliases assign to this host

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns: x0="http://www.w3.org/2001/XMLSchema">
    <comment>Example of config file: change its as your need.</comment>
    <identity>
        <hostid>hosta</hostid>
        <sslhostid>hostas</sslhostid>
        <cryptokey>D:\GG\R66\certs\test-key.des</cryptokey>
        <authentfile>D:\GG\R66\conf\OpenR66-authent-A.xml</authentfile>
    </identity>
    <client>
    </client>
    <ssl>
        <keypath>D:\GG\R66\certs\testsslnocert.jks</keypath>
        <keystorepass>testsslnocert</keystorepass>
        <keypass>testalias</keypass>
        <trustkeypath>D:\GG\R66\certs\testcert.jks</trustkeypath>
        <trustkeystorepass>testcert</trustkeystorepass>
        <trustuseclientauthenticate>False</trustuseclientauthenticate>
    </ssl>
    <directory>
        <serverhome>d:/GG/R66</serverhome>
        <in>in</in>
        <out>out</out>
        <arch>arch</arch>
        <work>work</work>
        <conf>conf</conf>
    </directory>
    <limit>
        <usefastmd5>True</usefastmd5>
        <fastmd5>D:\NEWJARS\gglib\win32\MD5.dll</fastmd5>
        <delayretry>10000</delayretry>
        <timeoutcon>10000</timeoutcon>
    </limit>
    <db>
        <dbdriver>h2</dbdriver>
        <dbserver>jdbc: h2:
D:/GG/R66/data/openr66;IFEXISTS=TRUE;MODE=Oracle;AUTO_SERVER=TRUE</dbserver>
        <dbuser>openr66</dbuser>
        <dbpasswd>openr66</dbpasswd>
    </db>
</config>
```

Waarp R66 Client Without Database Configuration File

`config`

- `comment`: string Optional

`identity`

- `hostid`: nonEmptyString

  Host ID in NON SSL mode
- `sslhostid`: nonEmptyString Optional

  Host ID in SSL mode
- `cryptokey`: Des-File

  Des CryptoKey File containing the key in Des mode for R66 passwords
- `authentfile`: XML-File

  Authentication File containing Authentications for partners

`client` Optional

- `taskrunnernodb`: booleanType Optional default="False"

  When client with no DB, do R66 will use XML files as permanent information on Transfer Tasks

`ssl` Optional

- `keypath`: JKS-File

  JKS KeyStore for R66 Access in SSL Mode (containing Server Certificate)
- `keystorepass`: nonEmptyString

  Password for the KeyStore
- `keypass`: nonEmptyString

  Password for the Server Certificate within the KeyStore
- `trustkeypath`: JKS-File

  JKS Trust KeyStore for R66 Access in SSL Mode with Client Authentication (containing Client Certificate)
- `trustkeystorepass`: nonEmptyString

  Password for the Trust KeyStore
- `trustuseclientauthenticate`: booleanType default="False"

  True if R66 will only allow client through SSL authentication

`directory`

- `serverhome`: directoryType

  Home Directory for the OpenR66 Server (relative path to this Home)
- `in`: nonEmptyString Optional default="IN"

  Default Receive Directory

- `out`: nonEmptyString Optional default="OUT"

  Default Send Directory
- `arch`: nonEmptyString Optional default="ARCH"

  Default Archive Directory
- `work`: nonEmptyString Optional default="WORK"

  Default Working Directory
- `conf`: nonEmptyString Optional default="CONF"

  Configuration Directory

`limit`

- `serverthread`: nonNulInteger default="8" Optional

  Number of Threads on Server side (=Number of Cores)
- `clientthread`: nonNulInteger default="80" Optional

  Number of Threads on Client side (=10xServer)
- `memorylimit`: nonNulInteger default="4000000000" Optional

  Memory Limit for R66 Server Java Process
- `sessionlimit`: nonNegInteger default="8388608" Optional

  HBandwidth for one session (64Mb)
- `globallimit`: nonNegInteger default="67108864" Optional

  Global Bandwidth (512Mb)
- `delaylimit`: nonNulInteger default="10000" Optional

  Delay between 2 checks of Bandwidth (10s). The less this value, the better the bandwidth limitation is done. However take care to not give too low value
- `runlimit`: nonNulInteger default="10000" Optional

  Limite by batch of active transfers (10000)
- `delaycommand`: nonNulInteger default="5000" Optional

  Delay between 2 execution of the Commander (5s)
- `delayretry`: nonNulInteger default="30000" Optional

  Delay between 2 attemps in case of error (30s)
- `timeoutcon`: nonNulInteger default="30000" Optional

  Delay before a Time Out occurs (30s)
- `blocksize`: nonNulInteger default="65536" Optional

  Block size (64Ko). A value between 8 KB to 16 MB is recommanded
- `gaprestart`: nonNulInteger default="30" Optional

  Gap to use when restarting a transfer as gap x blocksize
- `usenio`: booleanType default="False" Optional

Usage of NIO support for the files. According to the JDK, it can enhance the performances

- `usecpulimit`: booleanType default="False" Optional

  Usage of CPU Limitation when new request starts

- `usejdkcpulimit`: booleanType default="False" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `cpulimit`: decimalType default="0.0" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `connlimit`: nonNegInteger default="0" Optional

  Usage of CPU limitation based on Native JDK support, else (False) on Java Sysmon library

- `digest`: integerNotNegative default=0 (to setup a different Digest than MD5 globally)
- `usefastmd5`: booleanType default="True"

  Usage of FastMD5 library. The library greatly improves the performances for the MD5 computations

- `fastmd5`: SODLL-File Optional

  Library JNI Filepath. If the path is empty, the pure Java version will be used

- `taskrunnernodb`: booleanType Optional default="False"

  When client with no DB, do R66 will use XML files as permanent information on Transfer Tasks

- `checkversion`: booleanType Optional default="False"

  True to allow the extended protocol on end of request (>= 2.3)

- `globaldigest`: : booleanType Optional default="True"

  If True, it will enable a global digest (MD5, SHA1, ...) per file per transfer

db Optional

- `taskrunnernodb`: booleanType Optional default="False"

  When client with no DB, do R66 will use XML files as permanent information on Transfer Tasks

`business`

- `businessid`: nonEmptyString

  the host ids (1 by 1) that will be allow to use Business Request (>= 2.3)

`aliases`

- `alias`

  This will allow alias usage for host ids (from version 2.4.12).

- `realid`: nonEmptyString
  The real host id that will have aliases (locally)
- `aliasid`: nonEmptyString with separators as blank or '|'
  The set of aliases assign to this host

Waarp R66 Client for Submit Only Configuration File

`config`

- `comment`: string Optional

`identity`

- `hostid`: nonEmptyString

  Host ID in NON SSL mode
- `sslhostid`: nonEmptyString Optional

  Host ID in SSL mode
- `cryptokey`: Des-File

  Des CryptoKey File containing the key in Des mode for R66 passwords

`directory`

- `serverhome`: directoryType

  Home Directory for the OpenR66 Server (relative path to this Home)
- `in`: nonEmptyString Optional default="IN"

  Default Receive Directory
- `out`: nonEmptyString Optional default="OUT"

  Default Send Directory
- `arch`: nonEmptyString Optional default="ARCH"

  Default Archive Directory
- `work`: nonEmptyString Optional default="WORK"

  Default Working Directory
- `conf`: nonEmptyString Optional default="CONF"

  Configuration Directory

`limit`

- `blocksize`: nonNulInteger default="65536" Optional

  Block size (64Ko). A value between 8 KB to 16 MB is recommanded

`db`

- `dbdriver`: address

  4 types of database are currently supported: oracle, mysql, postgresql, h2
- `dbserver`: normString

  Connection to the database in JDBC mode (jdbc: type://[host: port]....). Use the database documentation to find the correct syntax for the JDBC connection
- `dbuser`: address

  Database User
- `dbpasswd`: nonEmptyString

Database User's Password

`aliases`

- `alias`

  This will allow alias usage for host ids (from version 2.4.12).

  - `realid`: nonEmptyString

    The real host id that will have aliases (locally)
  - `aliasid`: nonEmptyString with separators as blank or '|'

    The set of aliases assign to this host

Waarp R66 Limit Configuration File

config

- `comment`: string Optional

identity

- `hostid`: nonEmptyString

  Host ID in NON SSL mode

limit

- `sessionlimit`: nonNegInteger default="8388608" Optional

  Bandwidth for one session (64Mb)

- `globallimit`: nonNegInteger default="67108864" Optional

  Global Bandwidth (512Mb)

- `delaylimit`: nonNulInteger default="10000" Optional

  Delay between 2 checks of Bandwidth (10s). The less this value, the better the bandwidth limitation is done. However take care to not give too low value

- `runlimit`: nonNulInteger default="10000" Optional

  Limit by batch of active transfers (10000)

- `delaycommand`: nonNulInteger default="5000" Optional

  Delay between 2 execution of the Commander (5s)

- `delayretry`: nonNulInteger default="30000" Optional

  Delay between 2 attemps in case of error (30s)

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns: x0="http://www.w3.org/2001/XMLSchema">
    <comment>Example of config file: change its as your need.</comment>
    <identity>
        <hostid>hosta</hostid>
    </identity>
    <limit>
        <sessionlimit>0</sessionlimit>
        <globallimit>0</globallimit>
        <delaylimit>10000</delaylimit>
        <runlimit>10000</runlimit>
        <delaycommand>5000</delaycommand>
        <delayretry>30000</delayretry>
    </limit>
</config>
```

Waarp R66 Rule Configuration File

`rule`

- `comment`: string Optional
- `idrule`: nonEmptyString

  Rule ID
- `hostids`

  List of Host Ids allowed to use this rule. No Host Id means all allowed.
  - `hostid`: nonEmptyString unbounded

    Host ID allowed to use this rule
- `mode`: nonNulInteger

  1=SEND 2=RECV 3=SEND+MD5 4=RECV+MD5
- `recvpath`: nonEmptyString Optional default="IN"

  Default Receive Directory
- `sendpath`: nonEmptyString Optional default="OUT"

  Default Send Directory
- `archivepath`: nonEmptyString Optional default="ARCH"

  Default Archive Directory
- `workpath`: nonEmptyString Optional default="WORK"

  Default Working Directory
- `rpretasks`

  List of tasks -if any- to execute before transfer on receiver side
- `rposttasks`

  List of tasks -if any- to execute after transfer on receiver side
- `rerrortasks`

  List of tasks -if any- to execute after an error on receiver side
- `spretasks`

  List of tasks -if any- to execute before transfer on sender side
- `sposttasks`

  List of tasks -if any- to execute after transfer on sender side
- `serrortasks`

  List of tasks -if any- to execute after an error on sender side

Where List of tasks is

- `tasks`
  - `task` Optional unbounded

- `type`: nonEmptyString

  Type of Task: `LOG, MOVE, MOVERENAME, COPY, COPYRENAME, EXEC, EXECMOVE, EXECOUTPUT, EXECJAVA, TRANSFER, VALIDFILEPATH, DELETE, LINKRENAME, RESCHEDULE, TAR, ZIP, TRANSCODE, SNMP`

- `path`: nonEmptyString

  Argument -often a path- applied to the task where substitution can occur like #TRUEFULLPATH#, #FILESIZE#, #RULE#, #DATE#, #TRANSFERID#, ...

- `delay`: nonNegInteger

  Maximum delay for execution of the task in ms

Example:

```
<rule>
 <idrule>rule2</idrule>
 <hostids>
  <hostid>hosta</hostid>
  <hostid>hostb</hostid>
 </hostids>
 <mode>2</mode>
 <recvpath></recvpath>
 <sendpath></sendpath>
 <archivepath></archivepath>
 <workpath></workpath>
 <rpretasks>
  <tasks>
   <task>
    <type>LOG</type>
    <path>mon info</path>
    <delay>0</delay>
    <rank>0</rank>
   </task>
   <task>
    <type>LOG</type>
    <path>une autre info</path>
    <delay>0</delay>
    <rank>1</rank>
   </task>
  </tasks>
 </rpretasks>
 <rposttasks>
  <tasks>
   <task>
    <type>LOG</type>
    <path>test</path>
    <delay>0</delay>
    <rank>0</rank>
   </task>
   <taskno>
    <type>EXECRENAME</type>
    <path>D:/GG/R66/conf/montest.bat #TRUEFULLPATH#
D:\GG\FTP\fredo\a\#TRANSFERID#_#ORIGINALFILENAME# #TRUEFILENAME#
#ORIGINALFILENAME# #DATE# #HOUR# #REMOTEHOST# #LOCALHOST# #TRANSFERID#
#RANKTRANSFER# #BLOCKSIZE#</path>
    <delay>20000</delay>
    <rank>0</rank>
   </taskno>
   <taskno>
    <type>COPYRENAME</type>

<path>D:/GG/FTP/fredo/a/#DATE#_#HOUR#_#TRANSFERID#_#REMOTEHOST#_#LOCALHOST#_#ORI
GINALFILENAME#_#TRUEFILENAME#_%s_%s</path>
    <delay>0</delay>
    <rank>0</rank>
   </taskno>
  </tasks>
 </rposttasks>
 <rerrortasks>
  <tasks>
   <task>
    <type>LOG</type>
    <path>erreur</path>
    <delay>1</delay>
    <rank>0</rank>
   </task>
  </tasks>
 </rerrortasks>
 <serrortasks>
  <tasks>
   <task>
    <type>LOG</type>
    <path>erreur</path>
    <delay>1</delay>
    <rank>0</rank>
   </task>
  </tasks>
 </serrortasks>
</rule>
```

Waarp R66 Authentication Configuration File

`authent`

- `comment`: string Optional
- `entry` Multiple(1: n)

  Used to initialize remote Hosts table at setup or with client with no database support
  - `hostid`: nonEmptyString

    Host ID of remote Host
  - `address`: address (DNS or IP)

    Address of remote host (IP or DNS entry)
  - `port`: nonNullInteger

    Port associated with the Address of the remote Host
  - `isssl`: booleanType default="False"

    True if this Address Entry is for SSL mode
  - `admin`: booleanType Optional default="False"

    True if this Address Entry allows Admin access through R66 Protocol
  - `isclient`: booleanType Optional default="False"

    True if this Address Entry is for a Client
  - `keyfile`: GGP-File choice 1

    GoldenGate Password File containing the password for this host
  - `key`: nonEmptyString choice 2

    GoldenGate Password for this host

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<authent xmlns: x0="http://www.w3.org/2001/XMLSchema">
    <comment>example for ServerA</comment>
    <entry>
        <hostid>hosta</hostid>
        <address>127.0.0.1</address>
        <port>6666</port>
        <isssl>False</isssl>
        <keyfile>D:\GG\R66\certs\test-passwd.ggp</keyfile>
    </entry>
    <entry>
        <hostid>hostas</hostid>
        <address>127.0.0.1</address>
        <port>6667</port>
        <isssl>True</isssl>
        <admin>True</admin>
        <keyfile>D:\GG\R66\certs\test-passwd.ggp</keyfile>
    </entry>
    <entry>
        <hostid>hostb</hostid>
        <address>127.0.0.1</address>
        <port>6676</port>
        <isssl>False</isssl>
        <isclient>False</isclient>
        <keyfile>D:\GG\R66\certs\test-passwd2.ggp</keyfile>
    </entry>
    <entry>
        <hostid>hostbs</hostid>
        <address>127.0.0.1</address>
        <port>6677</port>
        <isssl>True</isssl>
        <admin>True</admin>
        <key>a5847a6ebb2eb5230554eb160326e7b1f53a193d9c6ee1b0</key>
    </entry>
    <entry>
        <hostid>test</hostid>
        <address>127.0.0.1</address>
        <port>6670</port>
        <isssl>False</isssl>
        <isclient>True</isclient>
        <keyfile>D:\GG\R66\certs\test-passwd3.ggp</keyfile>
    </entry>
    <entry>
        <hostid>tests</hostid>
        <address>127.0.0.1</address>
        <port>6670</port>
        <isssl>True</isssl>
        <admin>True</admin>
        <isclient>True</isclient>
        <keyfile>D:\GG\R66\certs\test-passwd3.ggp</keyfile>
    </entry>
</authent>
```

Tasks in rules

We focus here on the several tasks that are possible to run before a transfer starts (pre action), after a transfer is finished correctly (post action) or after an error occurs (either in pre or post action or during transfer: error action).

Those actions are defined in one rule. Each rule contains 2 parts:

1. Sender actions: A host is a Sender if it is the requester on a SEND rule or if it is the requested on a RECV rule.
2. Receiver actions: A host is a Sender if it is the requester on a RECV rule or if it is the requested on a SEND rule.

Each action could be on pre, post or error step, each step can have several actions.

It is defined with a unified form of XML:

```
    <tasks>
      <task>
         <type>NAME</type>
         <path>path</path>
         <delay>x</delay>
      </task>
      <task>
         <type>NAME</type>
         <path>path</path>
         <delay>x</delay>
      </task>
  </tasks>
```

- Type is the type of task to execute (see below the supported types)
- Path is a fixed argument for the task to execute. On this argument, string replacements are done when the following patterns are found:
  - `#TRUEFULLPATH#` : Current full path of current FILENAME
  - #`TRUEFILENAME`# : Current FILENAME (basename) (change in retrieval part)
  - #`ORIGINALFULLPATH`# : Original full path FILENAME (before changing in retrieval part)
  - #`ORIGINALFILENAME`# : Original FILENAME (basename) (before changing in retrieval part)
  - #`FILESIZE`# : File size if it exists

  - #INPATH# : In (Receive) path
  - #OUTPATH# : Out (Send) path
  - #WORKPATH# : Working (while receiving) path
  - #ARCHPATH# : Archive path (for export Log)
  - #HOMEPATH# : Home path (to enable for instance relative path commands)
  - #RULE# : Rule used during transfer
  - #DATE# : Current Date in yyyyMMdd format
  - #HOUR# : Current Hour in HHmmss format
  - #REMOTEHOST# : Remote host id (if not the initiator of the call)
  - #REMOTEHOSTIP# : Remote host IP (if not the initiator of the call)
  - #LOCALHOST# : Local host Id
  - #LOCALHOSTIP# : Local host IP
  - #TRANSFERID# : Transfer Id
  - #REQUESTERHOST# : Requester host Id
  - #REQUESTEDHOST# : Requested host Id
  - #FULLTRANSFERID# : Full Transfer Id as `TRANSFERID_REQUESTERHOST_REQUESTEDHOST`
  - #RANKTRANSFER# : Current or final RANK of block
  - #BLOCKSIZE# : Block size used
  - #ERRORMSG# : The current error message or NoError if no error occurs until this call
  - #ERRORCODE# : The current error code or '-' (Unknown) if no error occurs until this call
  - #ERRORSTRCODE# : The current error code message or "Unknown" if no error occurs until this call
  - #NOWAIT# : Used by Exec type task to specify that the command will be executed in asynchronous mode, without waiting any result from it
  - #LOCALEXEC# : Used by Exec type task to specify that the command will be executed not locally (within the JVM) but outside using a LocalExec Daemon (specified in the global configuration)
- Delay is generally the delay (if any) for execution before the execution becomes out of time.
- Additionnaly, a task will use also the argument from the transfer itself (Transfer Information). It uses the `String.format(info from rule, info from transfer.split by " ")`. This enables a very adaptive argument passing to the various tasks.

Each action is of one of the following items:

LOG

This task logs or writes to an external file some info:

- if delay is 0, no echo at all will be done
- if delay is 1, will echo some information in the normal log
- if delay is 2, will echo some information in the file (last deduced argument will be the full path for the file output)
- if delay is 3, will echo both in the normal log and in the file (last deduced argument will be the full path for the file output)
- If first word of the log is one of debug, info, warn or error, this word will be used as the log level

Example:

```
<task>
      <type>LOG</type>
      <path>warn information /path/logfile</path>
      <delay>2</delay>
</task>
```

This will log a "Warn" log as "warn information" to the file /path/logfile.

If delay is 0, nothing is done. If delay is 1, only normal log (warning log) is made. If delay is 3, then both logs from 1 and 2 are made.

MOVE

Move the file to the path designed by Path and Transfer Information arguments without renaming the filename (same basename).

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

Delay is ignored.

The file is marked as moved.

Example:

```
<task>
   <type>MOVE</type>
   <path>/newpath/</path>
```

```
</task>
```

This will move the file (not copying it) to the new directory /newpath/. The current file is now the moved file.

MOVERENAME

Move the file to the path designed by Path and Transfer Information arguments.

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

Delay is ignored.

The file is marked as moved.

Example:

```
<task>
   <type>MOVERENAME</type>
   <path>/newpath/newfilename</path>
</task>
```

This will move the file (not copying it) to the new directory /newpath/ with the new name as /newpath/newfilename. The current file is now the moved file.

COPY

Copy the file to the path designed by Path argument without renaming the filename (same basename). The path obtained must be an absolute path (not a relative path).

Delay and Transfer Information are ignored.

The file is not marked as moved.

Example:

```
<task>
   <type>COPY</type>
   <path>/newpath/</path>
</task>
```

This will copy the current file to /newpath/ as /newpath/currentfilename.

The current file stays the previous one (not changed).

COPYRENAME

Copy the file to the path designed by Path and Transfer Information arguments.

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

Delay is ignored.

The file is not marked as moved.

Example:

```
<task>
   <type>COPYRENAME</type>
   <path>/newpath/newfilename_%s_#TRANSFERID#</path>
</task>
```

Taking into consideration file transfer information to be "myinfoFromTransfer", this will copy the file as new file /newpath/newfilename_myinfoFromTransfer_transferid where transferid will be replaced by the unique id (as 123456789).

The current file is not changed and stays the same.

VALIDFILEPATH

Test if the current file is under one of the paths based on the Path and Transfer Information arguments.

The paths arguments are obtained from Path transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`).

The result should be as: "path1 path2 ..." where each path is separated by blank character.

If Delay is not 0, a log is printed out.

The file is not marked as moved.

Example:

```
<task>
   <type>VALIDFILEPATH</type>
   <path>/path1/ /path2/</path>
   <delay>1</delay>
</task>
```

This will check if the current file is under one of the directories specified, here /path1 or /path2. And it will log out the result of this check.

DELETE

This task deletes the current file.

The current file is no more valid.

No arguments are taken into account.

Example:

```
<task>
   <type>DELETE</type>
</task>
```

This command will simply delete the current file. Therefore no more file will be currently hold by the transfer task from this point.

LINKRENAME

Create a link of the current file and make the file pointing to it.

The link first tries to be a hard link, then a soft link, and if it is really not possible (not supported by the filesystem), it does a copy and rename task.

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

Delay is ignored.

The file is not marked as moved.

Example:

```
<task>
   <type>LINKRENAME</type>
   <path>/newpath/filenamelink</path>
</task>
```

This will try to link the file (if not possible, it performs a copy) to the new directory /newpath/ with name filenamelink.

RENAME

This task rename the current file without really touching the real file, which means that the target is changed, while no other action is taken:

- After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (String.format(Path,Info)). The path obtained must be an absolute path (not a relative path).
- Delay is ignored.
- The file is marked as moved.

Example:

```
<task>
   <type>RENAME</type>
   <path>/newpath/newfilename</path>
</task>
```

This will move the current file to the new path specified, but it does not touch the previous current file.

This command allows to change dynamically the file to send/recv or on post operation, without touching the real file.

EXEC

Execute an external command given by Path and Transfer Information arguments.

The Delay is the maximum amount of time in milliseconds before the task should be considered as over time and so in error.

The command path is obtained from Path transformed according to above dynamic replacements, and after a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

The file is not marked as moved.

The external command is supposed to behave as the following for its exiting value:

- exit 0, for a correct execution
- exit 1, for a warned execution (but however correct)
- other exit values for a failed execution

Example:

```
  <task>
   <type>EXEC</type>
     <path>/path/command arguments #TRANSFERID# #TRUEFULLPATH# %s</path>
   <delay>10000</delay>
  </task>
```

Taking into account that File transfer information could be "transferInformation", this will execute the command /path/command with the following arguments: "arguments transferId /path/currentFilename transferInformation".

EXECMOVE

Execute an external command given by Path and Transfer Information arguments.

The Delay is the maximum amount of time in milliseconds before the task should be considered as over time and so in error.

The command path is obtained from Path transformed according to above dynamic replacements, and after a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

The last line returned by the external command is interpreted as the new full absolute file path. The external command is responsible to really move the previous file to the new one.

The file is marked as moved.

The external command is supposed to to behave as the following for its exiting value:

- exit 0, for a correct execution
- exit 1, for a warned execution (but however correct)
- other exit values for a failed execution

Example:

```
 <task>
   <type>EXECMOVE</type>
```

```
    <path>/path/command arguments #TRANSFERID# #TRUEFULLPATH# %s</path>
  <delay>10000</delay>
</task>
```

Taking into account that File transfer information could be "transferInformation", this will execute the command /path/command with the following arguments: "arguments transferId /path/currentFilename transferInformation". The last line returned by the execution will be the new file path (absolute) of the current file. Therefore the current file is changed to this new file path.

EXECOUTPUT

Execute an external command given by Path and Transfer Information arguments.

The Delay is the maximum amount of time in milliseconds before the task should be considered as over time and so in error.

The command path is obtained from Path transformed according to above dynamic replacements, and after a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The path obtained must be an absolute path (not a relative path).

All lines returned by the external command (normal output) is interpreted as the possible error message in case of error.

If no error (0 or 1 as exit value), the output is ignored (no file move is done).

The file is not marked as moved, except in case of error (and if NEWFILENAME: is used as a prefix to the filename).

The external command is supposed to to behave as the following for its exiting value:

- exit 0, for a correct execution
- exit 1, for a warned execution (but however correct)
- other exit values for a failed execution, for which the output (stdout) lines are used as error message, bring back to the remote host as #ERRORMSG# and #ERRORCODE# / #ERRORSTRCODE#, and NEWFINALNAME: is used to find the new filename (if any)

Example:

```
<task>
  <type>EXECOUTPUT</type>
   <path>/path/command arguments #TRANSFERID# #TRUEFULLPATH# %s</path>
  <delay>10000</delay>
```

```
</task>
```

Taking into account that File transfer information could be "transferInformation", this will execute the command /path/command with the following arguments: "arguments transferId /path/currentFilename transferInformation".

If the execution is in error, the output will be bring back to the remote host through "#ERRORMSG# and "#ERRORCODE#".

Moreover, in case of error, if the last line contains the prefix NEWFILENAME:, then the current file is move (logically) to this new one.

EXECJAVA

Execute an external Java class given by Path and Transfer Information arguments.

The Delay is the maximum amount of time in milliseconds before the task should be considered as over time and so in error.

The class name (which must implement `R66Runnable`) is obtained from Path transformed according to above dynamic replacements, and after a String Format where Transfer Information is used as input (`String.format(Path,Info)`). The first argument is this full classname. The allocation must be of the form new MyClass(), so an empty constructor.

The file is not marked as moved.

Example:

```
<task>
   <type>EXECJAVA</type>
   <path>java.class.name #TRANSFERID# #TRUEFULLPATH#</path>
   <delay>10000</delay>
</task>
```

This will execute the class named java.class.name with the following arguments: "arguments transferId /path/currentFilename".

TRANSFER

Submit a new transfer based on the Path and Transfer Information arguments.

The transfer arguments are obtained from Path transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`).

The result should be as r66send command except "-info" must be the last field:

```
"-file filepath -to requestedHost -rule rule [-md5] [-start yyyyMMddHHmmss or -delay (delay or +delay)] -info transferInformation"
```

where each field is separated by blank character. Last field (`transferInformation`) may contain however blank character.

Delay is ignored.

The file is not marked as moved.

Example:

```
          <task>
             <type>TRANSFER</type>
               <path>-file #TRUEFULLPATH# -to remotehost -rule ruletouse -info transfer Information</path>
          </task>
```

This will create a new transfer request, using the current file (#TRUEFULLPATH#), to send (or receive, depending on the rule way ruletouse) to (from) the remote host, using "transfer Information" as argument to the transfer.

RESCHEDULE

Reschedule Transfer task to a time delayed by the specified number of milliseconds, if the error code is one of the specified codes and the optional intervals of date are compatible with the new time schedule

Result of arguments will be as following options (the two first are mandatory):

- `"-delay ms"` where ms is the added number of ms on current time before retry on schedule
- `"-case errorCode,errorCode,..."` where errorCode is one of the following error of the current transfer (either literal or code in 1 character): `ConnectionImpossible(C), ServerOverloaded(l), BadAuthent(A), ExternalOp(E), TransferError(T), MD5Error(M), Disconnection(D), RemoteShutdown(r), FinalOp(F), Unimplemented(U), Shutdown(S), RemoteError(R), Internal(I), StoppedTransfer(H), CanceledTransfer(K), Warning(W), Unknown(-), QueryAlreadyFinished(Q), QueryStillRunning(s), NotKnownHost(N), QueryRemotelyUnknown(u), FileNotFound(f), CommandNotFound(c), PassThroughMode(p)`
- `"-between start;end"` and/or `"-notbetween start;end"` (multiple times are allowed, start or end can be not set) and where start and stop are in the following format:

`Yn:Mn:Dn:Hn:mn:Sn` where n is a number for each time specification, each specification is optional, as Y=Year, M=Month, D=Day, H=Hour, m=minute, s=second.
Format can be X+n, X-n, X=n or Xn where X+-n means adding/subtracting n to current date value, while X=n or Xn means setting exact value
If one time specification is not set, it is based on the current date.

- If `"-notbetween"` is specified, the planned date must not be in the area.
- If `"-between"` is specified, the planned date must be found in any such specified areas (could be in any of the occurrence). If not specified, it only depends on "-notbetween".
- If none is specified, the planned date is always valid.
  - `"-count limit"` will be the limit of retry. The value limit is taken from the internal "information on transfer".

Each time this function is called, the limit value will be replaced as newlimit = limit - 1 in the "info of transfer".

To ensure correctness, the value must be in the "info of transfer" since this value will be changed statically in the "info of transfer". However, a value must be setup in the rule in order to reset the value when the count reach 0.

So in the rule, `"-count resetlimit"` must be present, where resetlimit will be the new value set when the limit reach 0. If it is missing, the condition is not applied.

Note that if a previous called to a reschedule was done for this attempt and was successful, the following calls will be ignored.

Important note: any subsequent task will be ignored and not executed once the reschedule is accepted. On the contrary, if the reschedule is not accepted, the following tasks will be executed normally.

- In case start > end, end will be +1 day
- In case start and end < current planned date, both will have +1 day.

Example:

```
"-delay 3600000 -case ConnectionImpossible,ServerOverloaded,Shutdown -notbetween H7:m0:S0;H19:m0:S0 -notbetween H1:m0:S0;H=3:m0:S0 -count 1"
```

means retry in case of error during initialization of connection in 1 hour if not between 7AM to 7PM and not between 1AM to 3AM and with a limit of 3 retries (retry will be reset to 1 in case of 3 attempts).

TAR

Create a TAR from the argument as source and destination or UNTAR files from a TAR file.

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`).

Delay of 1 = UNTAR PATH="sourceFile targetDirectory"

Delay of 2 = TAR PATH="targetFile sourceDirectory"

Delay of 3 = TAR PATH="targetFile sourceFile1 sourceFile2..."

The current file is not touched.

Example:

```
<task>
   <type>TAR</type>
   <path>/path/sourcetarfile /path/targetdirectory/</path>
   <delay>1</delay>
</task>
```

This will launch a "untar" command of tar file /path/sourcetarfile into directory /path/targetdirectory.

ZIP

Create a ZIP from the argument as source and destination or UNZIP files from a ZIP file.

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`).

Delay of 1 = UNZIP PATH="sourceFile targetDirectory"

Delay of 2 = ZIP PATH="targetFile sourceDirectory"

Delay of 3 = ZIP PATH="targetFile sourceFile1 sourceFile2..."

The current file is not touched.

Example:

```
<task>
   <type>ZIP</type>
   <path>/path/targetzipfile /path/sourcedirectory/</path>
   <delay>2</delay>
</task>
```

This will launch a "zip" command to zip into file /path/targetzipfile all contents from directory /path/sourcedirectory.

TRANSCODE

Allow to transcode a file from a Charset to another one.

After Path is transformed according to above dynamic replacements, it is then used as a String Format where Transfer Information is used as input (`String.format(Path,Info)`).

- `"-from fromCharset"`
- `"-to toCharset"`
- `"-newfile filename"` optional argument ; if not used, will be current filename.extension ; if used, extension is ignored
- `"-extension extension"` optional argument ; if not used, will be filename.transcode

fromCharset and toCharset are string representations of the official charsets in Java.

A convenient method (from Waarp Common) allows to list in html (-html), csv (-csv) or text format (-text) all the supported Charsets from your JVM. To use it, run the following command:

```
java -cp WaarpCommon-1.2.7.jar org.waarp.common.transcode.CharsetsUtil [-csv | -html | -text ]
```

It could also be used as a test of transcode outside R66:

```
java -cp WaarpCommon-1.2.7.jar org.waarp.common.transcode.CharsetsUtil -from fromFilename fromCharset -to toFilename toCharset
```

The current file is not touched and is not marked as moved.

Example:

```
<task>
   <type>TRANSCODE</type>
```

```
    <path>-from fromCharset -to toCharset -newfile /path/file</path>
  </task>
```

This will encode the current file using fromCharset as source charset code and using toCharset as target charset code, and the result will be placed into the file /path/file.

SNMP

This task SNMP trap or info according to snmp configuration with the info field:

- if delay is 0, only a warning trap/info is sent with the field info and the Transfer ID
- if delay is 1, a trap/info with full transfer information plus the field info is sent

Example:

```
<task>
   <type>SNMP</type>
   <path>information</path>
   <delay>0</delay>
</task>
```

This will send a snmp trap/info as "information" plus TransferID.

If the delay was 1, the same is sent with full Transfer information.

FTP

This task FTP allows to have a synchronous file transfer using FTP in a task. It uses the following parameters:

```
"-file filepath
-to requestedHost
-port port
-user user
-pwd pwd
[-account account]
[-mode active/passive]
[-ssl no/implicit/explicit]
[-cwd remotepath]
[-digest (crc,md5,sha1)]
[-pre extraCommand1 with ',' as separator of arguments]
-command command from (get,put,append)
[-post extraCommand2 with ',' as separator of arguments]"
```

The order of commands will be:

1. connection to requestHost on port (if ssl native => using native ssl link)
2. `User user`
3. `PASS pwd`

4. if account => `ACCT account`
5. if -ssl & auth => `AUTH, PBSZ 0, PROT P`
6. if passive => `PASV`
7. `CWD remotepath`; if error => `MKD remotepath` then `CWD remotepath` (and ignoring any error)
8. if pre => extraCommand1 with ',' replaced by ' ' (note: do not use standard commands from FTP like `ACCT,PASS,REIN,USER,APPE,STOR,STOU,RETR,RMD,RNFR,RNTO,ABOR,CWD,CDUP,MODE,PASV,PORT,STRU,TYPE,MDTM,MLSD,MLST,SIZE,AUTH`)
9. `BINARY` (binary format)
10. Transfer operation:
    1. if get => `RETR filepath.basename` ;
    2. if put => `STOR filepath` ;
    3. if append => `APPE filepath.basename`
11. if digest & get/put/append & remote site compatible with XCRC,XMD5,XSHA1 => `FEAT` (parsing if found corresponding XCRC,XMD5,XSHA1) ; then `XCRC/XMD5/XSHA1 filepath.basename` ; then locally comparing this XCRC/XMD5/XSHA1 with the local file
12. if post => extraCommand2 with ',' replaced by ' ' (note: do not use standard commands from FTP like `ACCT,PASS,REIN,USER,APPE,STOR,STOU,RETR,RMD,RNFR,RNTO,ABOR,CWD,CDUP,MODE,PASV,PORT,STRU,TYPE,MDTM,MLSD,MLST,SIZE,AUTH`)
13. `QUIT`

The current file is not touched and is not marked as moved.

Example:

```
        <task>
                <type>FTP</type>
                <path>-file /path/file -to remotehost -port port -user username -pwd password -command put</path>
        </task>
```

This will send (put) the file /path/file to the ftp server remotehost on port port using for credential username and password.

Example:

```
  <tasks>
   <task>
      <type>MOVE</type>
      <path>/pathout/</path>
      <comment>move the file to /pathout/#TRUEFILENAME#</comment>
```

```
        <delay>0</delay>
     </task>
     <task>
        <type>EXEC</type>
           <path>#HOMEPATH#/pathexec/monscript #TRUEFULLPATH# #ORIGINALFILENAME#
#FILESIZE# #RULE# %s %d #REMOTEHOST#</path>
            <comment>information passed by transfer is "a_string_without_blank
a_number" and replaced respectively in %s and %d</comment>
        <delay>30000</delay>
        <comment>maximum 30 seconds to execute this script</comment>
     </task>
   </tasks>
```

Waarp R66 Options

Limit CPU / Connexion

With these options (`usecpulimit`), Waarp R66 Server can limit the new requests according to a threshold on CPU global usage and on a maximum number of concurrent requests. When one of these limits is exceeded, the request is refused and postponed for a random number proportional to 30 seconds (`optiontimeoutcon`). After 3 retries, the request is cancelled.

These tests are done both on requester and requested side.

- `cpulimit`: Value for CPU is in float between 0 and 1 (% of CPU) where 0 or 1 means no limit.
- `usejdkcpulimit`: CPU is computed either with native support from JRE (but not all JRE support this) or either from Java Sysmon library.
- `connlimit`: Value for connexion is starting from 0 where 0 means no limit.

Check of IP on servers and clients

By default, real IP address used by the remote host is not compared against the IP (or the resolution given from DNS) stored with the remote HostId. If for security reasons this is required, you can enable this check to be done. When activated in `checkaddress`:

- For a server, the test will always be done.
- For a client, it depends if you enable it through the specific option `checkclientaddress`, then 2 cases occur:
  - If the client has an address of "0.0.0.0", then no test is done (this special address is to be used when a lot of remote clients are to be used without high security challenge).
  - If not, then the check is done as for a server.

Host as Client

In version 2.0, a new property is attached to the Host definition: `isclient`. This property stands to recognize remote client to prevent a server to try to initiate a request to this client. Since this client is not a server, it is not listening to incoming requests and therefore cannot be the target of an incoming request.

The special address "0.0.0.0" can also be used to specify a client but it extends it to not try to check its IP address. It is useful in the situation where you have a lot of clients and you don't want to declare them all in the Host table. All those clients will have to share the ID and the password (and the SSL key in case of strong authentication).

Cryptographic support

In version 2.0, many improvements were done on cryptographic side.

1. SSL: You can now have simple SSL support or even using string authentication of clients using the trustuseclientauthenticate option.
2. Password: all passwords are crypted using a private DES Key (private to each server but can be the same if you want). The passwords are crypted both on files, database. This key is locally referenced by `cryptokey` option. You have to use the Waarp Password tool (see Waap Password Tool) to crypt your passwords (at least for the administrator password) or to use the administration web interface.
3. On the network, a single way crypto key is used, common to all OpenR66 hosts when they need to exchange passwords.

Store Task saving on XML file for Thin Client

The special option `taskrunnernodb` allows you to have a persistent view of the transfer (the task) for Thin Client without database by using a XML file. It allows to restart a stopped or cancelled transfer smoothly without starting a new transfer from the beginning.

Control on restart transfer

Although an interrupted request restarts and each received packet should have been previously validated, the protocol include a backward move on the packet rank in order to ensure the quality of the transfer without doubt. When a transfer is restarted, OpenR66 first checks the receiver rank and takes it as the reference minus a gap. Then it checks the existing file (reception side) and check again against this rank minus a gap. You can control this feature by specifying the block size (`blocksize`) and the gap of rank (`gaprestart`), where the retransmitted byte size will be: `block size x rank`.

Usage of Waarp LocalExec Daemon

In order to improve the efficiency of external commands execution by preventing the fork of the Waarp R66 JVM process (costly regarding its memory), there is an optional support to execute those commands through a Waarp LocalExec Daemon (see in Waarp Local Exec). To use this support, you have to define the following:

- `uselocalexec` (default="False") to enable the use of LocalExec support
- `lexecaddr` (optional) which could contains the address (default="127.0.0.1")
- `lexecport` (optional) which could contains the port to be used (default="9999")

Usage of FastMD5 support

In order to improve the efficiency of the computation of MD5 on blocks during file transfer, 3 methods are provided.

1. **usefastmd5=False** => Will use internal JDK support of MD5 computation
2. usefastmd5=True but fastmd5 not declared or empty => Will use Java MD5 implementation more efficient than JDK version
3. usefastmd5=True and fastmd5=path to SO file or DLL (according to the systems) => Will use JNI C MD5 implementation suppose to be more efficient than version 2.

However note that the results could depends according to the systems and the JDK. Most of the time, Version 3 is the fastest, but sometime Version 2 is better than Version 3, and always better than Version 1. The efficiency of Version 3 greatly depends on the compilation of the C module. See Waarp Digest in Waarp Commons

It appears with recent versions of JVM that JDK in server mode is really efficient, almost equivalent to C JNI version. So one might use by default usefastmd5=False.

Note also that a new option can specify which kind of digest you want to use (it is for now a global option, not a local option): digest where values mean: MD5=0, MD2=1, SHA1=2, SHA256=3, SHA384=4, SHA512=5, CRC32=6, ADLER32=7.

Usage of the same database between several R66 servers

In case a database is shared among several R66 servers (with different names, so not in Multiple Monitors support option), the following tables will be totally shared:

- Host table: all partners definition, including itself will be shared among all servers. It implies also that the Key used to crypt/uncrypt the password are the same for all servers sharing the database.
- Rules table: all rule definitions will be shared. The difference of real action could be done either on the recv/send part of the tasks, but also using local variables (see the R66 Task Options) or local scripts

The following tables, even if shared, will have different entries for each server:

- Configuration table: the bandwidth limitation will be independent for each server
- Runner table: each transfer will be owned by one server only. Even if 2 servers are partners for the very same transfer, there will be 2 lines in the database, one for each server (requester and requested).
- MultipleMonitor table: this table is of no use in case of no multiple monitor usage ; in case of multiple monitor usage with several clusters on the same database, each cluster will act as a single host (so sharing or not sharing accordingly) and one line per cluster will be setup in this table.

Usage of Multiple Monitors support

In order to improve reliability of the OpenR66 File Transfer Monitors and the scalability, we propose a new option that allows to spread the load behind a Load Balancer in TCP mode (as HA-Proxy) and a shared storage (as a simple NAS).

1. `multiplemonitors`=1 => No multiple monitors will be supported (single instance)
2. `multiplemonitors`=n => n servers will be used as a single instance to spread the load and increase the high availability

Note that some specific attentions are needed such as to share the IN, OUT and WORK storages such that any servers can act on those files and any other storages that must be shared from the beginning of the transfer (pre-task) to the end of the transfer (post-task), and as to configure correctly the Load Balancer in TCP mode such as to spread the load and keep the connection once opened between 2 partners.

The principle is as follow:

- Put in place a TCP load balancer that allows to maintain a TCP connection with one server behind and that allows to spread the new connection attempts on the pool of servers available. The algorithm to spread the load could be for instance: the less connections opened at that time. A detection on open port could be enough to test the availability of the service. In more complex configuration, one could also implement a Java method that will do a “message” call to the proposed target server in order to test its availability.
- The IP/Port of the load balancer service will be the IP/port of the R66 service, behind it, you will have a set of R66 servers with their own IP/port couples internally (on which the load balancer spreads the load).
- Note that if the LB is “transparent”, meaning the IP from the client is not changed from the real server behind, the IP check could be possible on that pool of servers. Reversely, if the IP shown to the real R66 server is the one from the LB, the IP check will not be possible. However note that if the LB is transparent, it does not prevent that the client might still see the LB IP, and not the real R66 server's IP, and therefore preventing the IP check on client side. So a particular attention is needed if one wants to enable IP checking while using a LB in front of a pool of R66 servers.
- All R66 servers behind the LB will share the exact same name (ID), both for non SSL and SSL, and will share also the same database. They will have to share also the IN, OUT and WORK directories, and probably any other resources needed for the pre, post and error tasks (through a NAS for instance).
- All R66 servers will have to specify the same multiplemonitor option with the number of servers in the pool.

In theory, this enables the following HA capabilities:

- A load balance of all transfers among several clusters (horizontal scalability).
- A restart on disconnection, even on a crash of the original R66 server, since the new connection will go through the LB algorithm.

Note however that obtaining a HA R66 service does not required absolutely to have this option enabled. Indeed, one could check regularly through a monitoring tool that the service is still responding (using e message for instance), and if not to stop/restart the R66 service accordingly. Since the restart of a request is merely related to the “timeout” time, a check roughly repeated at that interval should enable a “clean” HA availability without having the complexity of a LB configuration.

One could also mixed the two solutions, in order to restart one unresponsive server in the HA pool.

Usage of Thrift support

The “`usethrift`” option (specifying a port > 0) allows to enable the Thrift server support in one R66 server. Currently only Synchronous Binary thrift protocol is allowed, so a client should use “Tsocket” for its Ttransport and “TbinaryProtocol” for its Tprotocol.

One example in Java is given in org.waarp.openr66.protocol.test.TestThriftClientExample to show how to interact with the Thrift R66 service.

This option should enables more capabilities to R66 to be embedded in existing applications, in a more large cover than just Java.

The current methods available are:

- `transferRequestQuery`: allows to initiate a submitted transfer from the related R66 server to another one. Note that the request could be asynchronous (immediately returns once the request is submitted) or synchronous (returns only once the request is done, whatever in error or in success, but it does not take into account future reschedule if any as it will return the status once the current try is over).
- `InfoTransferQuery`: allows to request some information on one particular transfer request
- `isStillRunning`: allows to quickly have the information on one particular transfer request running status or not
- `infoListQuery`: allows to get the information on file list on the local R66 server

Note that the Thrift service, for security reason, is only opened on 127.0.0.1 address since no authentication is made, as it stands for a local service.

Error task on Init step transfer

In the early stage of the transfer, some tests are made to validate that the request is valid. Among those tests, there are:

- unknown rule
- incompatible rule setup between the 2 partners (both requiring they act as the sender for instance)
- file not found or not readable
- request already started or already totally finished

In those case, previously, no error tasks will run. Now, we decide to enable error tasks to be run, in particular to enable the “reschedule” task. However, if one is willing to not have such error tasks running (globally for all rules) in such condition (before any pre task is executed), it will have to pass the following option to the R66 server java command:

```
-Dopenr66.executebeforetransferred=0
```

Windows Service support

With R66, it is possible to instantiate R66 as a Windows service through Apache Commons Daemon.

In the source of R66, you will find in the directory org.waarp.openr66.service the script service.bat. This script has to be updated to reflect your configuration.

```
@echo off

rem -- DO NOT CHANGE THIS ! OR YOU REALLY KNOW WHAT YOU ARE DOING ;)

rem -- Organization:
rem -- EXEC_PATH is root (pid will be there)
rem -- EXEC_PATH\..\logs\ will be the log place
rem -- EXEC_PATH\windows\ is where prunsrv.exe is placed
rem -- DAEMON_ROOT is where all you jars are (even commons-daemon)
rem -- DAEMON_NAME will be the service name
rem -- SERVICE_DESCRIPTION will be the service description
rem -- MAIN_DAEMON_CLASS will be the start/stop class used

rem -- Root path where the executables are
set EXEC_PATH=C:\Waarp\Run

rem -- Change this by the path where all jars are
set DAEMON_ROOT=C:\Waarp\Classpath

rem -- Service description
set SERVICE_DESCRIPTION="Waarp R66 Server"

rem -- Service name
set SERVICE_NAME=WaarpR66

rem -- Service CLASSPATH
set SERVICE_CLASSPATH=%DAEMON_ROOT%\myjar.jar

rem -- Service main class
set MAIN_SERVICE_CLASS=org.waarp.openr66.service.R66ServiceLauncher

rem -- Path for log files
set LOG_PATH=%EXEC_PATH%\..\logs
```

```
rem -- STDERR log file: IMPORTANT SINCE LOG will be there according to logback.xml
set ERR_LOG_FILE=%LOG_PATH%\stderr.txt

rem -- STDOUT log file: IMPORTANT SINCE LOG will be there according to logback.xml
set OUT_LOG_FILE=%LOG_PATH%\stdout.txt

rem -- Startup mode (manual or auto)
set SERVICE_STARTUP=auto

rem -- JVM option (auto or full path to jvm.dll, if possible pointing to server version)
rem example: C:\Program Files\Java\jdk1.7.0_05\jre\bin\server\jvm.dll
set JVMMODE=--Jvm=auto



rem -- Java memory options
set JAVAxMS=64m
set JAVAxMX=512m

rem -- Logback configuration file: ATTENTION recommendation is to configure output to STDOUT or STDERR
set LOGBACK_CONF=%EXEC_PATH%\..\conf\logback.xml

rem -- R66 configuration file
set R66_CONF=%EXEC_PATH%\..\conf\config-serverA2-2.xml

rem -- prunsrv.exe location
set PRUNSRVEXEC=%EXEC_PATH%\windows\prunsrv.exe

rem -- Loglevel of Daemon between debug, info, warn, error
set LOGLEVEL=info
```

Usage of No Database for Server

For special project where no database is needed, but then loosing the ability to store all transfer status and associated capacity (such as automatic retry if not specified in the rules), the server code now supports to not have any database connections. However it is still possible to store the trace of the transfers within XML files, such that one can automatize some actions in regard to the status of each transfer through file analysis. Note however that some functionality like the monitoring will be limited. The option `taskrunnernodb` will allow to use XML files support if set to true. If set to false, no information will be retained out of memory of the process.

Usage of ExecJava class

In order to facilitate the integration in application modules, OpenR66 now supports the ability to run specific Java Class through 3 ways. Note that this functionality is only valid starting in version 2.3.

- One is through pre or post or error tasks using the EXECJAVA keyword, following by the full class name which must implement the R66Runnable interface.
- Another one is through specific R66Business command, which will also execute an R66Runnable implementation, through for instance the AbstractExecJavaTask abstract class that could be extended.
- Finally, there is the possibility to associate a Business Class (see `R66BusinessInterface`) through a Business Factory (see `R66BusinessFactoryInterface`) to each transfer that will run several methods in the various steps that could occur:
  - `void checkAtStartup(R66Session session)`: launched at the very startup of the transfer and before the pre commands
  - `void checkAfterPreCommand(R66Session session)`: launched after the pre commands and before the transfer starts
  - `void checkAfterTransfer(R66Session session)`: launched after the transfer is finished and before the post commands
  - `void checkAfterPost(R66Session session)`: launched after the post commands and before the end of the request
  - `void checkAtError(R66Session session)`: launched once an error occurs
  - `void checkAtChangeFilename(R66Session session)`: launched if the filename is changed during the commands (pre or post)
  - `void releaseResources()`: launched at the very end, to release any internal resources that should be released
  - `String getInfo()` and `void setInfo(String info)`: launched by programmatic (business code) to enable to set a special info (as String) and to retrieve it at any time.
- Note that to allow a host to call a Business Request, it has to be added in the configuration file as

  `<business><businessid>hostname</businessid>...</business>`.

  If not set, the host will not be allow. On EXECJAVA, the security is first that the rule is only local to the host, and second the rule has the possibility to limit the allowed hosts to be partner of it.

Transcode support

With R66, it is possible to transcode one file from one charset to another charset using the TRANSCODE task. This one takes 2 arguments: `-from fromCharset` and `-to toCharset` where fromCharset and toCharset are string representations of the official charsets in Java. Optional arguments `-newfile newfilename` and `-extension extension` could be added. If `-newfile` is specified, the target filename will be the one specified. If it is not specified, the target filename will be the source filename with as extension either the one specified, or if not specified, the extension ".transcode".

For EBCDIC charsets, see:

- http://www-01.ibm.com/software/globalization/ccsid/ccsid_registered.html
- http://publib.boulder.ibm.com/infocenter/pcomhelp/v5r9/topic/com.ibm.pcomm.doc/reference/html/hcp_reference.htm
- In particular (but not limited to), one could consider the following charsets (from or to):
    - France: IBM297 or IBM01147
    - Italy: IBM280 or IBM01144
    - UK: IBM285 or IBM01146
    - International (Switzerland, Belgium): IBM500 or IBM01148
    - Austria/Germany: IBM273 or IBM01141
    - Spain and Latin America: IBM284 or IBM01145
    - Portugal, Brazil, USA, Canada, Netherlands: IBM037 or IBM01140
    - Central and Eastern Europe: IBM870
    - Cyrillic: x-IBM1025 (x-IBM1381?)
    - Turkey: IBM1026
    - Cyrillic Ukraine: x-IBM1123
    - Denmark, Norway: IBM277 or IBM01142
    - Finland or Sweden: IBM278 or IBM01143
    - Greece: x-IBM875 or x-IBM1124

A convenient method (from Waarp Common) allows to list in html (-html), csv (-csv) or text format (-text) all the supported Charsets from your JVM. To use it, run the following command:

```
java -cp WaarpCommon-1.2.7.jar
org.waarp.common.transcode.IdentifyCharsetsAvailable [-csv|-html|-text]
```

It could also be used as a test of transcode outside R66:

```
java  -cp  WaarpCommon-1.2.7.jar  org.waarp.common.transcode.CharsetsUtil
-from fromFilename fromCharset -to toFilename toCharset
```

Note that in MVS environment, it might be necessary to use **`-Dfile.encoding=UTF8`** to fix some issues with passwords.

An extract as example of the output from a JDK 1.7.05 Windows from Oracle using -html option:

Name	CanEncode	IANA Registered	Aliases
IBM01140	true	true	• ccsid01140 • cp01140 • 1140 • cp1140
IBM01141	true	true	• cp1141 • ccsid01141 • cp01141 • 1141
IBM01142	true	true	• cp01142 • cp1142 • 1142 • ccsid01142
IBM01143	true	true	• cp01143 • 1143 • ccsid01143 • cp1143
IBM01144	true	true	• cp01144 • cp1144 • ccsid01144 • 1144
IBM01145	true	true	• cp1145 • cp01145 • ccsid01145 • 1145
IBM01146	true	true	• ccsid01146 • cp01146 • cp1146 • 1146
IBM01147	true	true	• ccsid01147

			• cp1147 • 1147 • cp01147
IBM01148	true	true	• cp1148 • ccsid01148 • 1148 • cp01148
IBM01149	true	true	• cp1149 • cp01149 • ccsid01149 • 1149
IBM285	true	true	• ibm285 • ebcdic-cp-gb • cpibm285 • cp285 • csIBM285 • ebcdic-gb • 285 • ibm-285
IBM297	true	true	• cp297 • ibm297 • 297 • cpibm297 • ebcdic-cp-fr • ibm-297 • csIBM297

FTP Client support

With R66, it is possible to forward or receive a file by FTP. In passive mode (an external FTP client connects to the server to initiate a server), one can use the Waarp Gateway FTP. In active mode (the server will connect to a remote FTP server to initiate a transfer), one can use the integrated FTP Client (based on FTP4J) as a Task after or before a file transfer. This client is compatible with FTP, FTPS and FTPSE. In R66, the task is named FTP. See in Waarp Commons the package Waarp Ftp Client.

Proxy/Reverse Proxy support

With R66, it is possible install a proxy/reverse proxy in a DMZ in order to ensure high level of security. This Proxy/RP will forward any request to the target defined. No database is needed for this R66 Proxy. See in Waarp Commons the package Waarp Proxy R66.

Global Digest support

With R66, it is now possible (from 2.4.10) to have only one Digest per file transfer. Previously it was optional and only by packet. While this is still possible, it is also optional to have one global digest computed efficiently for the full file transfer. This option is activated by default but could be deactivated by using the `<limit><globaldigest>` entry set to false or 0.

Self Request

With R66, it is now fully supported (from 2.4.10) to send a request of file transfer to itself, whatever using a direct file transfer or a submitted file transfer.

Enhanced capability to handle filename with "blank" characters

Previously, filename transmitted should not have any "blank" character since they could introduce some issues. In order to allow such characters in the filename, a change that could lead to backward incompatibility was made from version 2.4.13. This change now uses the ': ' as separator (considering this character is not allowed in most filename implementations). The code keeps the possibility to still accept blank character as separator (as previously to version <= 2.4.12) and is therefore backward compatible. However, if one wants to stay with the old way, one can force the R66 server to use the old blank way by specifying the following property on java command:

```
-Dopenr66.usespaceseparator=1
```

Possibility to block/unblock new requests

When someone wants to stop his Waarp server, he/she might want to wait first that all requests are over and finished. To do that, it is possible now to ask the Waarp server to block all new requests while letting the existing one to continue. This operation is reversible and he/she can unblock as well. This can be achieved either through the web administration interface, or through the ServerShutdown command using the extra '-`block`' or '-`unblock`' option.

Focus on RESCHEDULE Task

The RESCHEDULE task has the following capabilities:

- If one RESCHEDULE task is executed (wherever it is) and validated (the transfer is really rescheduled), all following tasks are ignored and the execution flow stops

- The task will rely on 2 informations: the rule default information (specifically for the maximum restart available) and the transfer information which will include the current retry counter.

The options are:

- "`-delay ms`" where ms is the added number of ms on current time before retry on schedule
- "`-case errorCode,errorCode,...`" where errorCode is one of the following error of the current transfer (either literal or code in 1 character):
  - `ConnectionImpossible(C), ServerOverloaded(l), BadAuthent(A), ExternalOp(E), TransferError(T), MD5Error(M), Disconnection(D), RemoteShutdown(r), FinalOp(F), Unimplemented(U), Shutdown(S), RemoteError(R), Internal(I), StoppedTransfer(H), CanceledTransfer(K), Warning(W), Unknown(-), QueryAlreadyFinished(Q), QueryStillRunning(s), NotKnownHost(N), QueryRemotelyUnknown(u), FileNotFound(f), CommandNotFound(c), PassThroughMode(p)`
- "`-between start;end`" and/or "`-notbetween start;end`" (multiple times are allowed, start or end can be not set) and where start and stop are in the following format:
  - `Yn:Mn:Dn:Hn:mn:Sn` where n is a number for each time specification, each specification is optional, as Y=Year, M=Month, D=Day, H=Hour, m=minute, s=second.
  - Format can be X+n, X-n, X=n or Xn where X+-n means adding/subtracting n to current date value, while X=n or Xn means setting exact value
  - If one time specification is not set, it is based on the current date.
  - If "`-notbetween`" is specified, the planned date must not be in the area.
  - If "`-between`" is specified, the planned date must be found in any such specified areas (could be in any of the occurrence). If not specified, it only depends on "`-notbetween`".
  - If none is specified, the planned date is always valid.
  - In case start > end, end will be +1 day
  - In case start and end < current planned date, both will have +1 day.
- "`-count limit`" will be the limit of retry. The value limit is taken from the "info on transfer".
  - Each time this function is called, the limit value will be replaced as newlimit = limit - 1 in the "info of transfer".
  - To ensure correctness, the value must be in the "info of transfer" since this value will be changed statically in the "info of transfer". However, a value must be setup in the rule in order to reset the value when the count reach 0.

  - So in the rule, "`-count resetlimit`" must be present, where resetlimit will be the new value set when the limit reach 0. If it is missing, the condition is not applied.

Note: When the task is definitively in error (counter to 0), the counter in the transfer information is reset to the counter default limit set in the rule and the transfer is stopped. Then it is up to other way to relaunch the transfer to restart the transfer.

Among the various possibilities to relaunch a transfer once it is in error:

- Through the RESCHEDULE task, filtered by the error code and according to constraints and limits
- Through the Admin GUI (HTTPS): Transfer -> Restart
- Through command line: RequestTransfer or SubmitTransfer
- Through API using Thrift: using transferRequestQuery method
- Through the database directly by changing the status of UPDATEDINFO to the value TOSUBMIT (3)

Example of RESCHEDULE usage:

With the following rule:

- `"-delay 3600000 -case ConnectionImpossible,ServerOverloaded,Shutdown -notbetween H7: m0: S0;H19: m0: S0 -notbetween H1: m0: S0;H=3: m0: S0 -count 1"`

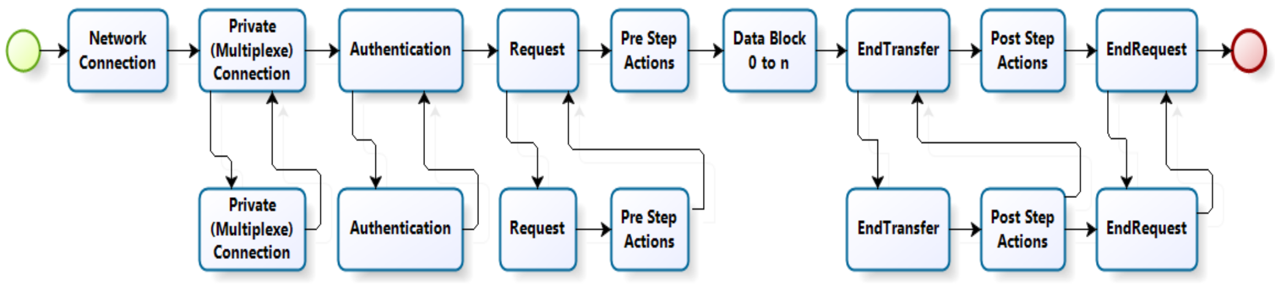
The reschedule task means retry in case of error during initialization of connection in 1 hour if not between 7AM to 7PM and not between 1AM to 3AM and with a limit of 3 retries (retry will be reset to 1 in case of 3 attempts).

Waarp R66 Internals

R66 Protocol

We describe here the logic of the protocol. It has been designed to be efficient, secured and to fullfill specific requirements of MFT and integration in IT.

A simplified picture is shown here:



First we describe the different value that can be found.

- A Request (DbTaskRunner) can have several UpdatedInfo status:
    - *UNKNOWN* : no particular information on it.
    - *NOTUPDATED* : in used by other database object when they are taken into account.
    - *INTERRUPTED* : a request is interrupted but can be rescheduled.
    - *TOSUBMIT* : a request is proposed to be submitted by the Commander (or other database object are supposed to be taken into account).
    - *INERROR* : a request is in error and can not be submitted by the Commander until its status is changed explicitly.
    - *RUNNING* : a request is currently running.
    - *DONE* : a request is over and fully done.

- A Request can have several Step values:
    - *NOTASK* : the request has never started.
    - *PRETASK* : the request is currently in Pre transfer step.
    - *TRANSFERTASK* : the request is currently in transfer step.
    - *POSTTASK* : the request is currently in Post transfer step (valid transfer only).
    - *ALLDONETASK* : the request is fully finished (UpdatedInfo is in DONE status too).
    - *ERRORTASK* : the request is currently in the Error step while an error occurs (either in *PRE*, *TRANSFER* or *POST* step).

- A Request has two Step values:
  - *GlobalStep* : the current request step value
  - *GlobalLastStep* : this is the last valid request step value. When *GlobalStep* is in *ERROR* step, *GlobalLastStep* says in which step it was before entering in error. This information is used to enable restart of the Request from this valid last step.

- Each *Step* values and *UpdatedInfo* has an *ErrorCode* detailed information:
  - *InitOk* : stands for correct initialization of the Request (startup and authentication)
  - *PreProcessingOk*, *TransferOk*, *PostProcessingOk* : stand for correct ending of the specified step
  - *CompleteOk* : stands for all action are correct (*ALLDONE* for *step* and *DONE* for *UpdatedInfo*)
  - *Running* : stands for current Step is in Running status.
  - *StoppedTransfer*, *CanceledTransfer* : stand for a Request where an action stopped or canceled the given Request. A Stopped Request can be restart from the current status. A Canceled Request starts from the beginning of the current step. For instance for the Transfer step, Stopped will imply that restart is from the current valid transferred block, while Cancel will imply to restart from the very beginning of the transfer (first block).
  - *QueryAlreadyFinished* : stands for special code where the Request is in fact remotely already finished and so can be finished locally.
  - Other codes specifies different kinds of error (*NotKnownHost*, *Shutdown*, *RemoteError*, ...):
    - *ConnectionImpossible* : connection is impossible (local or remote)
    - *BadAuthent* : authentication is in error (local or remote)
    - *ExternalOp* : external operation is in error (pre, post or error actions)
    - *TransferError* : transfer is in error (for instance, uncorrect block)
    - *MD5Error* : transfer is in error due to a bad MD5 checking (in MD5 mode)
    - *Disconnection* : the network was disconnected
    - *RemoteShutdown* : the remote host is in shutdown
    - *Shutdown* : the local host is in shutdown
    - *FinalOp* : a final operation (not an external operation) is in error (for instance the finalization of the received file)
    - *Unimplemented* : the requested command is unimplemented (for instance, a task in a rule is unknown)
    - *RemoteError* : a remote error was received and stops the transfer
    - *Internal* : an internal error arrives

- *Warning* : an external execution (pre or post operation) is executed well but with a warning (code 1)
- *Unknown* : an unknown error arrives
- *QueryStillRunning* : a required transfer during a restart is still in transfer
- *NotKnownHost* : the remote host is unknown
- *QueryRemotelyUnknown* : a required transfer during a restart or other commands should exist but is not identified on the remote host (for instance after a purge of log)
- *FileNotFound* : the file is not found
- *CommandNotFound* : the Command is not found
- *PassThroughMode* : the requested transfer (restart) is in PassThrough mode and is not compatible with the required action

A request of transfer follows a sequential logic:

Startup

- Requester:
  1. The requester checks first if the given partner name is an alias, and if so, replaces it by the real host id.
  2. The request is registered in the database with a TOSUBMIT status.
  3. A Request that was in a INTERRUPTED status is changed in a TOSUBMIT status.
  4. The Commander get some requests with the TOSUBMIT status and makes them as RUNNING status.
  5. The Commander submits those requests as separates ClientRunners.
  6. The ClientRunner first checks that the given Request is not a "Self Requested" request, meaning that only requester host can execute a ClientRunner, except if this request was in the POSTTASK step so that Requested Host can finalize the request.
  7. The ClientRunner gets the remote requested Host address and tries to open the connection. If a network conection with the given requested host is already opened, this network connection is reused by the new ClientRunner.
  8. The connection can use SSL support (different port than non SSL). This is an option of transfer. This option is selected while selecting the Host ID for SSL support for the remote Requested Host. This option is CPU and Memory consuming.
- Requester and Requested:
  1. Once the network connection is found, a private connection (in memory connection) is opened to enable the multiplexing of this request with other requests on the same network connection. This private connection is attached to a new LocalChannelReference which references the Request, the session, the remote and private connections. A valid LocalChannelReference contains two

private connection Ids, one for the local private connection, and one for the remote private connection.

1. A Startup Message is sent to the local private connection to initiate it.
2. The same Startup Message is sent on the remote private side (when connection occurs) to initiate the relation between them and to instantiate the same LocalChannelReference on the remote host.

Authentication

- Requester:
  1. Once the LocalChannelReference is OK, the Requester host sends an Authent Message in order to authenticate this host.
- Requested:
  1. The Requested host sends back its own Authent Message too.

Request

- Requester:
  1. Once authenticated, the Requester Host sends the Request Message.
- Requested:
  1. The Requested Host check if the authentication and the request are compatibles, check some specific options on the request itself (start, restart, …), the status of the file if it is the sender, then it runs its Pre Task step
     - The Request is in PreProcessingOK as status.

Pre-Tasks

- Requested:
  1. If the Pre Task step is ok, it sends back the validated Request Message to the Requester Host.
     - The Request is in InitOk as detailed information on requester side.
- Requester:
  1. The Requester Host check if the authentication and the request are compatibles, check some specific options on the request itself (start, restart, …), the status of the file if it is the sender.
     - The Request is in InitOk.
  2. The validated Request is now running the Pre Task step.
     - Once finished, the Request is in PreProcessingOK status.

Data Transfer

- Sender:
  1. Now the transmission can start. From now on, this is the sender that leads the communication, no more the requester (it could be the same host however, depending on the way of transfer). The sender (which could be either the

Requested or the Requester host) launch its own RetrieveRunner. This RetrieveRunner sends to the other host all DataBlock Messages.
2. Each DataBlock Message can include a Hash control (MD5 or SHA or other) of the packet in it (option of transfer). This option is not mandatory and is CPU consuming.
3. Once all DataBlock are sent, the RetrieveRunner sends an EndTransfer Message to the receiver host.

- Receiver:
  1. For each DataBlock, the receiver appends this new block, checking the order of the block, and optionnaly the hash control.
  2. Once all blocks are received, the EndTransfer Message indicates the end of transmission from the Sender.

Post-Actions

- Receiver:
  1. The receiver host executes the Post actions.
     - The Request is in PostProcesseingOk status.
  2. The receiver host sends back the validated EndTransfer Message.
     - The Request is in TransferOk status on sender side.
- Sender:
  1. The Request is now on Finalize way. Sender host executes the Post actions.
     - The Request is in PostProcesseingOk status.

End Request

- Sender:
  1. Once the PostProcessing is over, the RetrieveRunner (Sender) sends to the remote host a EndRequest Message.
- Receiver:
  1. The remote host sends back the validated EndRequest Message.
- Receiver and Sender:
  1. The Request is now totally finished and its status is CompleteOk ALLDONE.

At each step, an error can occurs and will stop the request, setting its UpdatedInfo to INERROR or INTERRUPTED status. The GlobalStep could be in ERROR status if the ERROR step action is run.

File Status

A file change of status along the transfer according to the receiver or sender of the file itself. Note that the requester host can be either the receiver or sender, according to the used rule.

If the Host is the sender:

- At startup, before any pre actions is taken, the file is logically instantiated:
  - If the transfer is in SendThrough mode, no test is done. The file is logically instantiate as is.
  - If not, in first try, the file is searched under the default send directory (OUT) in case of relative path.
  - If the file is an absolute path, it is searched according to this path.
  - If the file cannot be found or is not readable (except in SendThrough mode), the transfer stops in error for no file found.
- After the pre actions, the file is tested again on its existence and access before the real send:
  - If the transfer is in SendThrough mode, no test is done. The file is logically instantiate as is.
  - If not, in first try, the file is searched under the default send directory (OUT) in case of relative path.
  - If the file is an absolute path, it is searched according to this path.
  - If the file cannot be found or is not readable (except in SendThrough mode), the transfer stops in error for no file found.
  - If the file was moved (MOVE or EXECMOVE), the receiver is informed of the changed source filename.
- Once the transfer is over, no action is taken on the file except those implied by any post actions.
- In case of error before pre actions, the restart will take the first step of checking before pre actions.
- In case of error after or while the pre actions, the restart will take into account the name as stored (eventually modified by previously successfully executed pre-actions, which will not be re-executed).

If the Host is the receiver:

- At startup, before any pre actions is taken, the file is logically instantiated:
  - The file is logically instantiate as is.
- After the pre actions, the file is modified:
  - If the transfer starts from zero (usual situation), a temporary unique file is created in the working directory (using the transfer id, a unique number and the

extension ".r66"). The file is tested in write. The logical file points to this physical temporary file.

  - If the transfer does not start from zero (a restart occurs), the previous file name is reused. The file is again tested in write. The logical file points to this physical temporary file.
  - If the transfer is in RecvThrough mode, no test is done on the file. It is just instantiated logically on the basis of a temporary unique file virtually created in the working directory.
  - If the file is not writeable (except for RecvThrough mode), the transfer stops in error.

- It is not a good idea to change the name of the file during the Pre actions for the receiver. However, it is possible to do so, for instance in order to move this working file into a different directory than the working directory specified in the rule.
- Once the transfer is over, the file is renamed without its temporary extension ".r66" and is moved into the receive directory (IN). The Post actions can change the name of the file (MOVE, MOVERENAME, ...). No test is done once the file transfer is over, even after a move.
- In case of error before pre actions, the restart will take the first step of checking before pre actions.
- In case of error after or while the pre actions, the restart will take into account the name as stored (eventually modified by previously successfully executed pre-actions, which will not be re-executed).

Transfer Restarting

A transfer can stop for several reasons:

- The network connection was interrupted.
- One of both hosts is shutting down.
- An error occurs (one of the above).
- An administrator stops or cancels the transfer.

It is possible to restart a transfer once interrupted.

It is generally recommended that the requester host takes the decision of restarting the transfer. However, both hosts can try to restart the transfer.

There is several ways to do so:

- Using the administrator interface Cancel/Restart web page
- Using the RequestTransfer command line

Commands

Waarp R66 is mainly a set of commands,from the server, to the clients, through utilities.

In the following, commands are in fact Class in Java. To access them from shell, follow the Command line helper.

org.waarp.openr66.server package:

R66Server

It launches the Waarp R66 Server using as unique argument the full XML configuration file. It loads the rest of the configuration (or update its configuration) from the database specified in the XML configuration file.

By default, three services are opened:

- Non SSL file transfer monitor
- SSL file transfer monitor
- HTTP file transfer observer

By default, 5 subdirectories (under the main home) are defined:

- Config directory where configurations files are stored
- Archive directory where archive transfer can be put (using post actions) or where transfer logs are placed
- Input directory where received files are stored (except if a post action moves it)
- Output directory where to be sent files are stored (or linked)
- Working directory where currently received files are temporarily placed

Utilities:

ServerInitDatabase

The first step in installation of an Waarp R66 Server or heavy client is to construct the XML configuration files and to initiate the database that support file transfers and configurations. This function is an helper for database initialisation.

- The first argument is the JDBC database connection XML file.

From there any argument are optional. Any existing data will be replaced. No data are deleted if not replaced.

- '-initdb' : It will try to create the database tables and to initiate the sequence (unique Id by host of runners from runner table).
- '-loadBusiness businessConfig' : It will load the business configuration into the database HostConfig table.
- '-loadAlias aliasConfig' : It will load the aliases configuration into the database HostConfig table.

- '-loadRoles rolesConfig' : It will load the roles configuration into the database HostConfig table.
- '-dir directory' : this argument is the directory from where to load rules into the database using the XML rule files present in this directory (rules table).
- '-auth hostConfig' : this argument is to load the host authentication XML file into the database (hosts table).
- '-limit limitConfig' : this argument is to load the bandwidth XML file into the database (configuration table).
- '-upgradeDb': this argument allow to try to upgrade the database schema (from version 2.4.17).

From that point, the Waarp R66 server (R66Server) can be started.

ServerExportConfiguration

One of the ideas of Waarp R66 is the ability to centralized some information asynchronously into a centralized database. This centralization can be in two ways:

- Import new rules, host authentication or bandwidth limitation: this can be done using the ServerInitDatabase after some file transfers from a post operation which could execute this utility.
- Export rules, hosts authentication and actual runners (done or not): this is the goal of this utility. It takes two arguments:
    - The client configuration XML file
    - The directory where to export files in XML format

It will export Rules, all present Runners, Host Authentications.

ConfigExport

This tool enables to ask the server remotely to export Rules and Hosts configuration into one file each.

It could be used before a retrieve of the configuration in a central repository.

Options are:

- The first argument is the client XML configuration file
- -hosts : export all Hosts configuration
- -rules : export all Rules configuration
- -business : export all Business configuration
- -alias : export all Alias configuration
- -role : export all Roles configuration
- -host host : try to run this command over host (if allowed)

ConfigImport

This tool enables to ask the server remotely to import Rules and Hosts configuration from one file each.

It could be used before a upload the configuration from a central repository after transmission.

Options are:

- The first argument is the client XML configuration file
- -hosts : import all Hosts configuration
- -purgehosts : purge all hosts configuration before importing new configuration
- -rules : import all Rules configuration
- -purgerules : purge all rules configuration before importing new configuration
- -business file (if compatible)
- -alias file (if compatible)
- -roles file (if compatible)
- -purgebusiness (if compatible)
- -purgealias (if compatible)
- -purgeroles (if compatible)
- -hostid fileTransferId (if compatible, from a previous file transfer)
- -ruleid fileTransferId (if compatible, from a previous file transfer)
- -businessid fileTransferId (if compatible, from a previous file transfer)
- -aliasid fileTransferId (if compatible, from a previous file transfer)
- -roleid fileTransferId (if compatible, from a previous file transfer)
- -host host (optional)

ServerShutdown

The Waarp R66 Server can be shutdown mainly in two ways:

- Sending a -SIGTERM signal (or under Unix a -SIGUSR1 signal) to the JVM process hosting the OpenR66 Server.
- **The prefered way** is to use the HTTPS Administrator interface.
- **Another prefered way** is to use this utility which sends a shutdown request through network but using a key shared physically (adminkey). This utility takes the same XML configuration file than the server (where the admin key is refered but not stored in any database) and uses the SSL service.
- If one wants to used the non SSL service, use the option -nossl
- This command allows also to "block" or "unblock" new requests while leaving currently running requests to finish normally. To block, add the argument '-`block`', to unblock add the argument '-`unblock`'.

LogExport

If one wants to centralized in asynchronous way the transfer logs, this utility is made for it. Its purpose is to export logs into the archive directory and to eventually purge them from database. It sends a local request to the server, which really export the logs into a file. No file are transfered, they are just export to a file in the global archive directory. To use it, two main ways can be achieved:

1. The local server runs this tool refering the OpenR66 Server using the client configuration XML file and then running a log request.
2. Directly runs a transfer request where the rule execute at pre processing this utility (again either in receive or send mode according to the initiate server).

- The First argument is the Client Configuration (eventually without database access).
- The rest of arguments can be:
    - '-clean' option: Change all UpdatedInfo to Done where GlobalLastTask is ALLDONETASK and status is CompleteOk (sometimes some runners can be done but UpdatedInfo could be erroneous - no impact but clean function -).
    - '-purge' option: This option removes all ALLDONETASK from those that will be exported.
    - '-start' option: This option specifies the low limit to select runners from start runner time. If not specified, there is no low limit.
    - '-stop' option: This option specifies the upper limit to select runners from start runner time. If not specified, there is no upper limit.

LogExtendedExport

Same as LogExport but with more options:

- The First argument is the Client Configuration (eventually without database access).
- The rest of arguments can be:
    - '-clean' option: Change all UpdatedInfo to Done where GlobalLastTask is ALLDONETASK and status is CompleteOk (sometimes some runners can be done but UpdatedInfo could be erroneous - no impact but clean function -).
    - '-purge' option: This option removes all ALLDONETASK from those that will be exported.
    - '-start' option: This option specifies the low limit to select runners from start runner time. If not specified, there is no low limit.
    - '-stop' option: This option specifies the upper limit to select runners from start runner time. If not specified, there is no upper limit.
    - -startid id: This option specifies the low limit for Transfer Id
    - -stopid id: This option specifies the upper limit for Transfer Id
    - -rule rule: This option specifies that logs will only concern this rule
    - -request host: This option specifies that logs will only concern this host
    - -pending: This option specifies that logs will only concern pending request

    - -transfer: This option specifies that logs will only concern in transfer request
    - -done: This option specifies that logs will only concern done (finished) request
    - -error: This option specifies that logs will only concern in error request
    - -host host: This options specifies a specific server to contact

ChangeBandwidthLimits

This tool enables to change dynamicaly the bandwidth limitations (only in memory, not in database). To use it, it should be used locally with the admin account.

- The First argument is the Client Configuration (eventually without database access).
- The rest of arguments can be:
    - '-wglob' option: Write Global limitation in Bytes by Second (minimum 1024 so 1KBs).
    - '-rglob' option: Read Global limitation in Bytes by Second (minimum 1024 so 1KBs).
    - '-wsess' option: Write Session limitation in Bytes by Second (minimum 1024 so 1KBs).
    - '-rsess' option: Read Session limitation in Bytes by Second (minimum 1024 so 1KBs).

org.waarp.openr66.client package:

SubmitTransfer

To transfer a file, two main methods exist. This one is a submission request, so an asynchronous operation since once the request is submitted, the client returns without waiting for the end of the operation.

It takes the following argument:

- The client XML configuration file as first argument, the one including database access
- At least 3 other arguments are necessary:
    - '-to' option: specifies the remote Host Id (either the Id for SSL or not).
    - '-file' option: specifies the file to transfer (either in receive or send mode).
    - '-rule' option: specifies the rule to apply (which specifies the transfer mode, the pre, post or error operations, ...).
- Or at least 2 other arguments are necessary:
    - '-to' option: specifies the remote Host Id (either the Id for SSL or not).
    - '-id' option: specifies the Id of a previous transfer (stopped or in error).
- Other options are:

    - '-info' option: specifies the optional information that is send at the same time with the transfer request (extra information that could be needed by the remote host).
    - '-md5' option: specifies that each block transfer will be checked with a MD5 key. If the rule used is already in MD5 mode, this option will change nothing.
    - '-block' option: specifies the block size (default is 64 KB).
    - '-nolog' option: specifies that this transfer will not be logged (only on requester side).
    - '-start' "time start" as yyyyMMddHHmmss (override previous -delay options)
    - '-delay' "+delay in ms" as delay in ms from current time(override previous -start options)
    - '-delay' "delay in ms" as time in ms (override previous -start options)

MultipleSubmitTransfer

This function is the same than SubmitTransfer except that multiple hosts and multiples files could be specifed using a comma (',') as separator to -file and -to arguments.

- -client : If the Rule is a RECV method, passing the -client additionnal option allows the function to contact the remote servers in case wildcards are used (?*~).

DirectTransfer

This is the second method to transfer a file. This method is direct, so as a synchronous operation. This time the client will do the real work (transfering the file). The options are exactly the same than with SubmitTransfer. However two cases exist:

1. Heavy client: The client XML configuration file includes the database access. All transfer operations will be logged (eventually deleted at the end from the client side if the '-nolog' option is set). This option is useful for "production" clients in a data center.
2. Light client: The client XML configuration file does not included the database access. All transfer operations will not be logged at all (at client side). This option is useful for "light client" like personal computers where transfering files to or from a data center is a necessity.

However, both clients can only be the initiator of the transfer (receive or send), since no service is running once the client is over.

MultipleDirectTransfer

This function is the same than DirectTransfer except that multiple hosts and multiples files could be specifed using a comma (',') as separator to -file and -to arguments.

SendThroughClient

This method is not a full implemented method. It is a way to route a file transfer from one protocol in Java to OpenR66. For instance, if a protocol like HTTP upload or FTP upload allows to get a file upload by packet, you can then route this file transfer through Waarp R66 to a final Waarp R66 Server using the protocol but without writing an intermediary file (directly write the bytes from the upload to the Waarp R66 SendThroughClient interface). A simple example is shown in TestSendThroughClient.

So this method needs some minor developments to be implemented.

RecvThroughClient

This method is not a full implemented method. It is a way to route a file transfer to one protocol in Java to OpenR66. For instance, if a protocol like HTTP or FTP download allows to get a file downloaded by packet, you can then route directly the file transfer from Waarp R66 from a remote Waarp R66 Server using the protocol directly but without writing an intermediary file (directly write the bytes from the download from the Waarp R66 RecvThroughHandler interface). A simple example is shown in TestRecvThroughClient.

So this method needs some minor developments to be implemented.

ProgressBarTransfer

This method is not a full implemented method. It is a way to implement a file transfer within a Graphical User Interface or to get information on progression during the transfer. Such an example is presented in the R66GUI.

So this method needs some minor developments to be implemented

RequestTransfer

This utility is used to get information for a specific runner or to have an action on this runner.

The arguments are the following:

- The first argument is the client XML configuration file including the database access.
- '-id' option: this is the Runner Id.
- '-to' or '-from' option (exclusive): this specifies the way of the request transfer. '-to' specifies that the original request was initiated by the current running host to the remote specified host. '-from' specifies that the original request was initiated by the remote specifed host.
- Optional arguments (exclusive): without any of those arguments, the request only returns the current information of the runner.
  - '-cancel' option: the runner will be canceled. Any file currently in writing will be deleted.
  - '-stop' option: the runner will be stopped (but not canceled).
  - '-restart' option: the runner will be restart (if stopped). Optionnally the following options can be applied in addition:
    - '-start' "time start" as yyyyMMddHHmmss (override previous -delay options)
    - '-delay' "+delay in ms" as delay in ms from current time(override previous -start options)
    - '-delay' "delay in ms" as time in ms (override previous -start options)

RequestInformation

This utility is used to get information for a file, a directory, with or without wildcard characters ('*' and '?").

The arguments are the following:

- The first argument is the client XML configuration file including the database access.
- '-to' option: specifies the requested host.
- '-rule' option: the rule
- '-file' the optional file for which to get info (may contain wildcard characters)

Optional arguments (exclusive): without any of those arguments, the request only returns the existence of the file or directory.

- '-exist' to test the existence.
- '-detail' to get the detail on file.
- '-list' to get the list of files.
- '-mlsx' to get the list and details of files

•Message

This utility is used to send a short message to another server (like a ping in Waarp R66 protocol)

The arguments are the following:

- The first argument is the client XML configuration file
- '-to' option: specifies the requested host.
- '-msg' option: the msg to send

Command line helper

In the zip distribution, there is an example of installation including a bin directory and a ENV_R66 file that contains some commands to be used in a shell on a Unix server. It should be easy to adapt those files to fit your needs.

```
SERVER SIDE
r66server
# start the Waarp R66 server
# no option

r66signal
# shutdown locally the server
# [ PID ] optional PID of the server process

r66shutd
# shutdown by network the server
# [-nossl] [-block | -unblock]

r66limit
# change limits of bandwidth
# "[ -wglob x ] [ -rglob w ] [ -wsess x ] [ -rsess x ]"

r66init
# init database from argument
# [ -initdb ] [ -loadBusiness businessConfiguration ] [ -loadRoles roleConfiguration ] [ -loadAlias aliasConfig ] [ -dir rulesDirectory ] [ -limit xmlFileLimit ] [ -auth xmlFileAuthent ] [ -upgradeDb ]

r66export
# export the log
# [ -purge ]|[ -clean ] [ -start timestamp ] [ -stop timestamp ]

r66cnfexp
# export configuration
# directory

r66confexp
# export configuration as arguments
# [-hosts] [-rules] [-business ] [-alias] [-roles] [-host host]

r66confimp
# import configuration as arguments
```

# [-hosts host-configuration-file] [-purgehosts] [-rules rule-configuration-file] [-purgerules] [-business file] [-purgebusiness] [-alias file] [-purgealias] [-roles file] [-purgeroles] [-hostid file transfer id] [-ruleid file transfer id] [-businessid file transfer id] [-aliasid file transfer id] [-roleid file transfer id] [-host host]

r66export

# export the log

# [-purge] [-clean] [-start timestamp] [-stop timestamp] where timestamp are in yyyyMMddHHmmssSSS format eventually truncated and with possible ':- ' as separators

r66logexport

# export the log (extended)

# [-host host] [-purge] [-clean] [-startid id] [-stopid id] [-rule rule] [-request host] [-pending] [-transfer] [-done] [-error] [-start timestamp] [-stop timestamp] where timestamp are in yyyyMMddHHmmssSSS format eventually truncated and with possible ':- ' as separators

r66limit

# change limits of bandwidth

# "[-wglob x] [-rglob w] [-wsess x] [-rsess x]"

CLIENT SIDE

r66info

# get information on remote files or directory

"-<u>to</u> host -<u>rule</u> rule [-<u>file</u> file] [-<u>exist</u> | -<u>detail</u> | -<u>list</u> | -mlsx]

r66mesg

# test the connectivity

-<u>to</u> host -msg "message"

r66req

# get information on transfers

-<u>id</u> transferId [-<u>to</u> hostId | -<u>from</u> hostId] [-<u>cancel</u> | -<u>stop</u> | -<u>restart</u> [-start yyyMMddHHmmss | -delay [+]timeInMs]]

r66syncsend

# synchronous transfer

-<u>to</u> hostId -<u>file</u> filepath -<u>rule</u> ruleId [-<u>md5</u>] [-<u>block</u> size] [-nolog] [-<u>info</u> "information"]

r66send

# asynchronous transfer

# -to hostId -file filepath -rule ruleId [-md5] [-block size] [-nolog] [-info "information"]

r66multisend

# R66 Multiple Submit

# (-to hostId,hostID -file filepath,filepath -rule ruleId) | (-to hostId -id transferId) [-md5] [-block size] [-nolog] [-start yyyyMMddHHmmssSSS | -delay +durationInMilliseconds | -delay preciseTimeInMilliseconds] [-info "information"]

r66multisyncsend

# multiple synchronous transfer

# (-to hostId,hostid -file filepath,filepath -rule ruleId) | (-to hostId -id transferId) [-md5] [-block size] [-nolog] [-start yyyyMMddHHmmssSSS | -delay +durationInMilliseconds | -delay preciseTimeInMilliseconds] [-info "information"]

Dependencies

This project depends on the following libraries:

- NETTY (excellent NIO framework) from version 3.5 (NIO support).
- Apache Commons IO from 2.3 (special file functions like wildcard support).
- Apache Commons Codec from 1.6 (Base64 support).
- Apache Commons Compress from 1.4 (TAR and ZIP support).
- Apache Commons Exec from 1.1 (external Exec support).
- DOM4J from 1.6.1 and JAXEN from 1.1.1 (XML support), only used by the current implementation of the standard Ftp Server. It could be replaced very easily.
- optional: LOGBACK from 1.0.5, can be replaced by any logger facilities (default is Java native logger). Only used by the current implementation of the standard Ftp Server. It could be replaced very easily.
- optional: JavaSysMon from 0.3.3 (from Waarp Common) to enable the CPU throtling. If the JRE supports native way, this is optional. If the CPU throtling is not required, it is optional.
- A Database support: right now tested were H2 database engine ; partially tested are Oracle, PostGreSQL and MySQL support. Any new database could easily be implemented (only 3 functions are to be implemented).
- From Waarp project, two modules are necessary: Waarp Common (files support) and Waarp Digest (MD5 and other digests support) ; on most of the projects, Waarp Gateway Kernel is also needed (HTTP support)
- From Waarp project: optionally Waarp LocalExec (Local Execution Daemon support) and Waarp SNMP (SNMP support) which implies to have also SNMP4J jars
- For FTP Client, Waarp uses FTP4J project.
- Jackson JSON library from version 2.2

Wiki part

R66Authentication

The authentication occurs at 4 levels:

1. Host/Key

   Mandatory, if a server is not known (no host id in the list, wrong key associated), no request will be allowed.

2. SSL

   If the SSL mode is used, the trust level of SSL could be managed:

   - the SSL Server key is validated by the client (SSL default mode)
   - the SSL Client key could be also validated by the server (trustuseclientauthenticate option, set to False by default, therefore not asking the client to identify itself with its own SSL key).

   So the SSL allows a simple side or dual sides authentication.

3. IP

   - checkaddress=True means that all IP addresses for Servers will be checked against the real IP on the network connection
   - checkclientaddress=True means that all IP addresses for Clients will be checked against the real IP on the network connection

   Those IP are tested to check the consistency between the declaration and the reality.

   - Warning: If a non transparent proxy is present, the IP of source is replaced by the one from the proxy.
   - Warning: For clients, except if one would like to have a very high level of security, it is in general not recommended to check the IP for the Client in order to have a simple configuration (for instance, one entry for many clients in a group)
   - Warning: The authentications are stored in the database. Therefore, if several servers share the same database, they will share also those identifications.

4. In case of Shared Database

   As the administrator option will probably be active for all Servers (since this field is necessary for themselved to be their own administrator, in order for instance to be able to shutdown from command line), there is an option that allows to specify a superset of roles locally, through local files, with fine grain role support. This way of doing is highly recommended in the case of database shared among several R66 servers, in order to give more flexibility and still security in the roles.

Special notice for Alias: since aliases are just replaced at the very beginning of the start of any operation, there is no need to have those aliases defined in the Host database table, since they will rely on the real host definition.

Centralization of Logs

Several ways exist to centralize transfer status.

- Using the SNMP behavior, using the same SNMP central point, with the related level of SNMP trap to be sent by the R66 servers (could be from nothing to All events, with intermediate levels as Start/Stop, Alert level, Important Warning level, ALl warning events)
- Using the task Export logs function (note that currently Import logs is not developped, since the behaviour of this central repository is business dependent) and then file transfer rule to push (or pull) file transfer logs
- Using its own tasks ans scripts to centralize status through pre, post and error tasks
- Using the same database for all R66 servers, but then leading to possible issue of latency (database access), even if R66 monitors are not so linked to database access

R66 Cluster or HA

Cluster configuration and High availability consideration

The cluster principle for Waarp R66 is as the following:

- Having a load balancer, based on TCP, allowing to maintain an open connection to stay open, and to balance new connection with a pool of R66 servers, using probably the less connection as elected
- The IP/port of the load balancer service is the R66 service address associated with the pool of servers
- If the load balancer is transparent, the IP of the client is visible from the R66 servers in the pool. If the load balancer is not transparent, the IP of the client is replaced by the IP of the load balancer itself, therefore the R66 servers in the poll will not be able to check the IP addresses of the partners.
- All servers in the pool will have the exact same IDs (SSL and not SSL), sharing the same database, the same filesystem sub part (work, out at least, probably also in, arch and conf). The configuration must set the multiplemonitor to the number of total available servers in the pool (at most available).

This should enable high availability on the following:

- a balance between several R66 servers (thus leading to less risk of "high throughput side effects")
- a high availability in the sens that if one server is down in the pool, the others will ensure the continuity of the service
- a restart on disconnection (after a crash or stop of one server in the pool) using standard restart procedure of transfer, going to another server but as they all share at least the database, work and out directory, the transfer can restart easily

However, a well enough service level could be sustained using simple monitoring on R66 server, forcing its stop/restart according to the results, giving almost the same result than cluster mode.

Indeed, when a server will retry a connection, it will respect a minimal delay between 2 checks, therefore the delay for the monitoring and restart is around this check delay or a factor of 2 of this delay. For instance, if the timeout/delay is setup to 30s, then the restart should occur in less than 60s to be almost equal to high availability.

Finally, regarding the database access, it is important to note that R66 monitors have a certain level of tolerance of unavailability of the database. As long as the transfer is not finishing, the database updates (which log the current status of the transfer) can be ignored. Each time, if the database connection is lost, the server will retry to open new connections when needed. However, when the transfer is finishing (in error or correct), the R66 server MUST save the status, and therefore, if it cannot, it sends back to its partner an internal erropr and will keep this transfer status as it was the las time it was saved. The next time this transfer will be restarted, it will restart form the point saved in the database.

R66 in DMZ

For DMZ configuration, several options are available with Waarp.

Waarp Gateway FTP

By installing a Waarp Gateway FTP in DMZ, you can have FTP transfers from outside in connection with a Waarp R66 server installed in the DMZ too. The Gateway FTP will allow sending or receiving of file through FTP to/from outside, while the Waarp R66 will allow sending or receiving to/from internal side. The links will be made through rules, bot in Gateway FTP and in Waarp R66.

In that case, the R66 DMZ server does not need necesseraly to be accessible from outside natively, since it is accessed through FTP service.

- Gateway FTP
  - Each put will be followed by a send file request from R66 DMZ server to an internal R66 partner
  - Each recv will be prefixed by a recv file request from R66 DMZ server to an internal R66 partner, then followed by the FTP recv file transfer
- Waarp R66
  - Each send file request received by the R66 DMZ server from an internal R66 partner will be followed by a put request in FTP (within R66 as a task) to a remote FTP server
  - Each recv file request received by the R66 DMZ server from an internal R66 partner will be prefixed by a recv request in FTP (within R66 as a task) to a remote FTP server, then followed by the R66 internal transfer

Waarp R66 in forward mode

By installing a DMZ R66 server, through the rules, it could act as a forward request will full checking.

The interest is to have full checking at once for all type of transfers, without having to directly connect to internal R66 servers from outside. The drawback is that this DMZ R66 server has a full configuration (using database and all host authentications), which could lead to some issues in very high level protected area.

- Waarp R66
  - Each send file request received by the R66 DMZ server from an internal R66 partner will be followed by a send request in R66 (within R66 as a task) to a remote R66 server
  - Each recv file request received by the R66 DMZ server from an internal R66 partner will be prefixed by a recv request in R66 (within R66 as a task) to a remote R66 server, then followed by the R66 internal transfer

Waarp Proxy R66

By installing a Proxy R66 server, it will forward in both ways requests directly to external or internal R66 servers.

The interest is to have a minimalist R66 server in DMZ, with no configuration that could be a source of attack. The drawback is that no control is made within this Proxy R66 server, meaning that the packet are just transmistted as is to the internal or external R66 partner. However, if some attacks as deny of service are made, this will be probably the first level of catch, then enhancing the security level of the R66 solution.

The configuration is made by pair, meaning that each listening interface (address, port, ssl mode) is linked to one and only one proxified interface (address, port, ssl mode). Therefore, let say that on internal side we have a R66 server named A, on external side a R66 server named B, the configuration will be as follow:

- Listening B' in DMZ through address/port/SSL mode (probably none) accessible from inside, linked to B
- Listening A' in DMZ through address/port/SSL mode (probably yes) accessible from outside, linked to A

Therefore, in A, the configuration to access to B is made through address/port/SSL mode defined in B', while the remote partner B will access to A through address/port/SSL mode defined in A'.

R66 Embedded

Several options are available to have an embedded version of the R66 monitor.

Natively in Java

The first option is to integrate the R66 monitor natively using client integration. Several examples exist in the distribution to show how to do that.

The second option is to integrate an applicative module with R66 monitor, such that it will call directly application Java modules during task executions.

Outside Java

The first option is of course to use scripts to call R66 command line commands.

The second option is to use the proposed Thrift interface which facilitates the integration of the client side in several languages (see Thrift for more details).

Among the supported languages in Thrift, we have:

- C++
- C#
- Cocoa
- D
- Delphi
- Erlang
- Haskell
- Java
- OCaml
- Perl
- PHP
- Python
- Ruby
- Smalltalk

R66 with Other Protocols

In order to integrate new protocols within Waarp Gateway and Waarp R66, several options exist.

Native integration

For FTP, the protocol is already integrated, both in Server (Gateway only) and Client (Gateway and R66) modes. This integration is done through additional tasks done before or after a file transfer in the native protocol.

We note active for initiated file transfer request, as for initiating the underlying network connection, while passive stands for the opposite, as for receiving an incoming network connection.

- Waarp Gateway
  - Native protocol = FTP in server mode (passive)
  - Extended protocol = R66 (active) or FTP (active) in pre command (RETRieve way) or post command (STOre way)
- Waarp R66
  - Native protocol = R66 in server mode (passive and active)
  - Extended procotol = FTP (active) in pre, post or error commands

Task through integration

The idea is to use external implementation of other protocols through tasks executing external commands.

This implementation enables for instance to forward a file transfer into another protocol as in post task. This post task could be:

- EXEC like tasks that execute an external command (native OS command)
- EXECJAVA tasks that execute a Java task using a specific class implementing the necessary protocol

Using this way enables for instance to migrate from one protocol to R66 protocol smoothly by keeping one server in the old mode corresponding with one R66 server, both acting as a relay (or gateway) between the 2 protocols, as the Waarp Gateway FTP does for FTP.

Example: other protocol named XXX

- Send XXX to R66
  1. A standard XXX client sends a file to your XXX server
  2. At the end of the reception in your XXX server, the transfer is transformed into a R66 file transfer using command line
  3. If the post execution is blocking in your XXX server, you can get back the status on the file transfer in R66 protocol (using for instance a blocking mode) to inform back the XXX client of the overall result

- Recv XXX through R66
    1. A standard XXX client asks to receive a file from your XXX server
    2. A blocking transfer is done with a R66 blocking command
    3. Once the receive is done in R66, the XXX server respond to the XXX client with the file transfer
- Send R66 to XXX
    1. A R66 partner sends a file to your R66 server
    2. At the end of the reception in your R66 server, a post task forward this transfer through your XXX client
    3. Once the file transfer in XXX protocol is done, the script launched by R66 get the status of this transfer for R66 monitor
- Recv R66 through XXX
    1. A R66 partner asks to receive a file from your R66 server
    2. A pre-task launches a receive file transfer operation from your XXX client
    3. Once the receive is done in XXX protocol, your R66 server respond to the R66 partner with the file transfer and continue with the R66 protocol

Platform compatibility

Waarp R66 needs at least a JRE 1.6. Until now, here is the list of the tested platforms (but not limited to):

- Windows (32 and 64 bits)
- Linux (32 and 64 bits, debian (Ubuntu for instance) or redhat (centOS for instance) based)
- AIX
- Z/OS (not directly tested by Waarp team, but by a final user, with success)

If you are aware of specific platforms where this software was successfully tested, we would be happy to grow the list.

Waarp R66 Administration & Monitoring

Waarp R66 Server comes with an administrator in HTTPS mode with authentication and a supervision in HTTP mode without authentication and a SNMP module.

Http Monitoring

The supervision enables only to show information on transfers (active or not) and is dynamically reloaded every 10 seconds. The access of the supervision is at

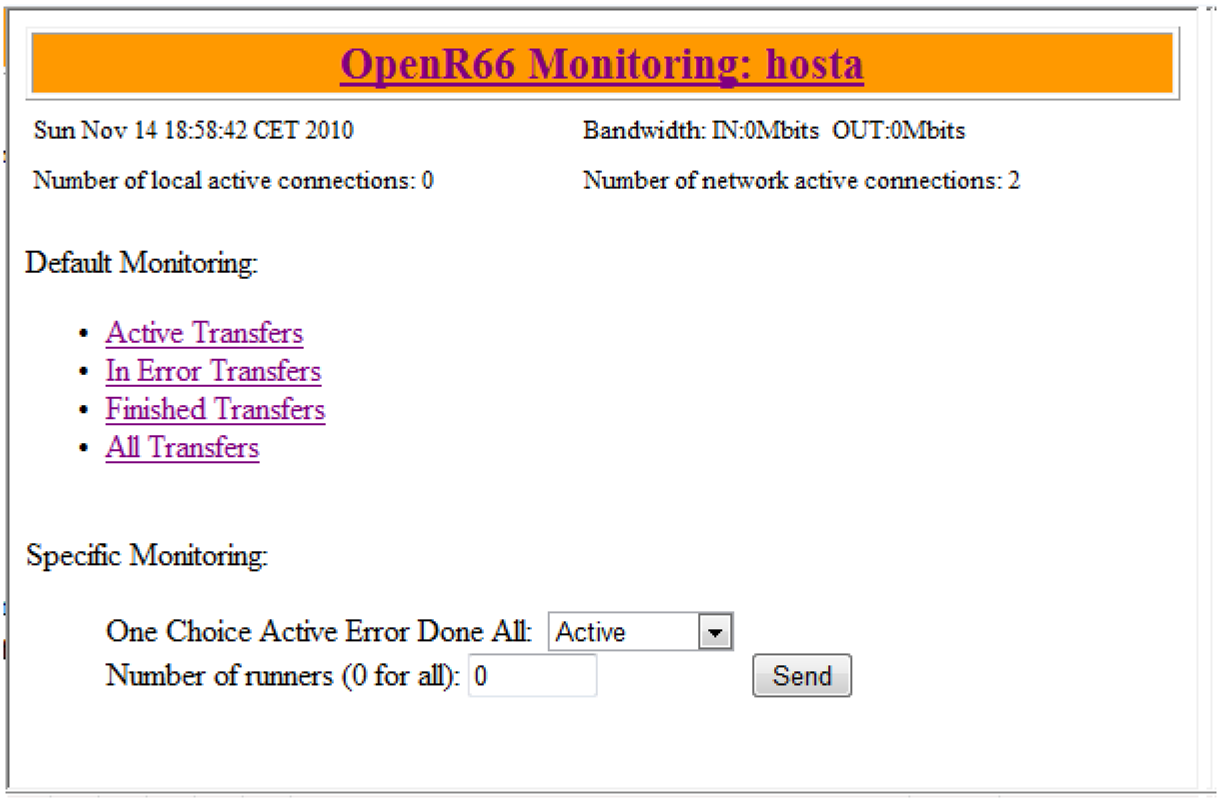
`http://host_address:8066/`

(8066 port can be changed in the configuration file).

A specific URL `http://host_address:8066/status` enables access to simple status where it returns a 200 OK code if everything is OK, else it returns a small page with some info and a status as Internal Server Error (500) denoting that some errors were found.

Another specific URL `http://host_address:8066/statusxml` enables access to detailed status where it returns an xml containing various informations as followed.

Note: `statusxml?NB=xxx&DETAIL=1` means how far in the past as xxx seconds this status should look for and is the given information presenting a detailed information - DETAIL=1 - or only short information - DETAIL clause absent -.


OpenR66 Monitoring: hosta

Sun Nov 14 18:58:42 CET 2010

Bandwidth: IN:0Mbits OUT:0Mbits

Number of local active connections: 0

Number of network active connections: 2

Default Monitoring:

- Active Transfers
- In Error Transfers
- Finished Transfers
- All Transfers

Specific Monitoring:

One Choice Active Error Done All: Active

Number of runners (0 for all): 0

Example of XML output:

```
<STATUS>
<HostID>hosta</HostID><Date>Mon Feb 21 18: 57: 26 CET 2011</Date><LastRun>Mon
Feb 21 18: 57: 26 CET 2011</LastRun>
<SecondsRunning>180</SecondsRunning><NetworkConnections>2</NetworkConnections><N
bThreads>21</NbThreads>
<InBandwidth>0</InBandwidth><OutBandwidth>0</OutBandwidth>

<OVERALL>
<AllTransfer>78</AllTransfer><Unknown>0</Unknown><NotUpdated>0</NotUpdated><Inte
rrupted>0</Interrupted>
<ToSubmit>0</ToSubmit><Error>1</Error><Running>0</Running><Done>77</Done>
</OVERALL>

<STEPS>
<Notask>1</Notask><Pretask>0</Pretask><Transfer>0</Transfer><Posttask>0</Posttas
k>
<AllDone>77</AllDone><Error>0</Error>
</STEPS>

<RUNNINGSTEPS>
<AllRunning>0</AllRunning><Running>0</Running><InitOk>0</InitOk>
<PreProcessingOk>0</PreProcessingOk><TransferOk>0</TransferOk><PostProcessingOk>
0</PostProcessingOk>
<CompleteOk>0</CompleteOk>
</RUNNINGSTEPS>

<ERRORTYPES>
<ConnectionImpossible>0</ConnectionImpossible><ServerOverloaded>0</ServerOverloa
ded>
<BadAuthent>0</BadAuthent><ExternalOp>0</ExternalOp><TransferError>0</TransferEr
ror>
<MD5Error>0</MD5Error><Disconnection>0</Disconnection><FinalOp>0</FinalOp>
<Unimplemented>0</Unimplemented><Internal>0</Internal><Warning>0</Warning>
<QueryAlreadyFinished>0</QueryAlreadyFinished><QueryStillRunning>0</QueryStillRu
nning>
<KnownHost>0</KnownHost><RemotelyUnknown>0</RemotelyUnknown>
<CommandNotFound>0</CommandNotFound><PassThroughMode>0</PassThroughMode>
<RemoteShutdown>0</RemoteShutdown><Shutdown>0</Shutdown><RemoteError>0</RemoteEr
ror>
<Stopped>0</Stopped><Canceled>0</Canceled><FileNotFound>0</FileNotFound>
<Unknown>1</Unknown>
</ERRORTYPES>
</STATUS>
```

SNMP monitoring

A SNMP agent is also available in OpenR66 from version 2.1.1. It includes the possibility to poll values from a SNMP manager or to push value as notification to a SNMP manager using trap or inform model.

HTTPS Administrator

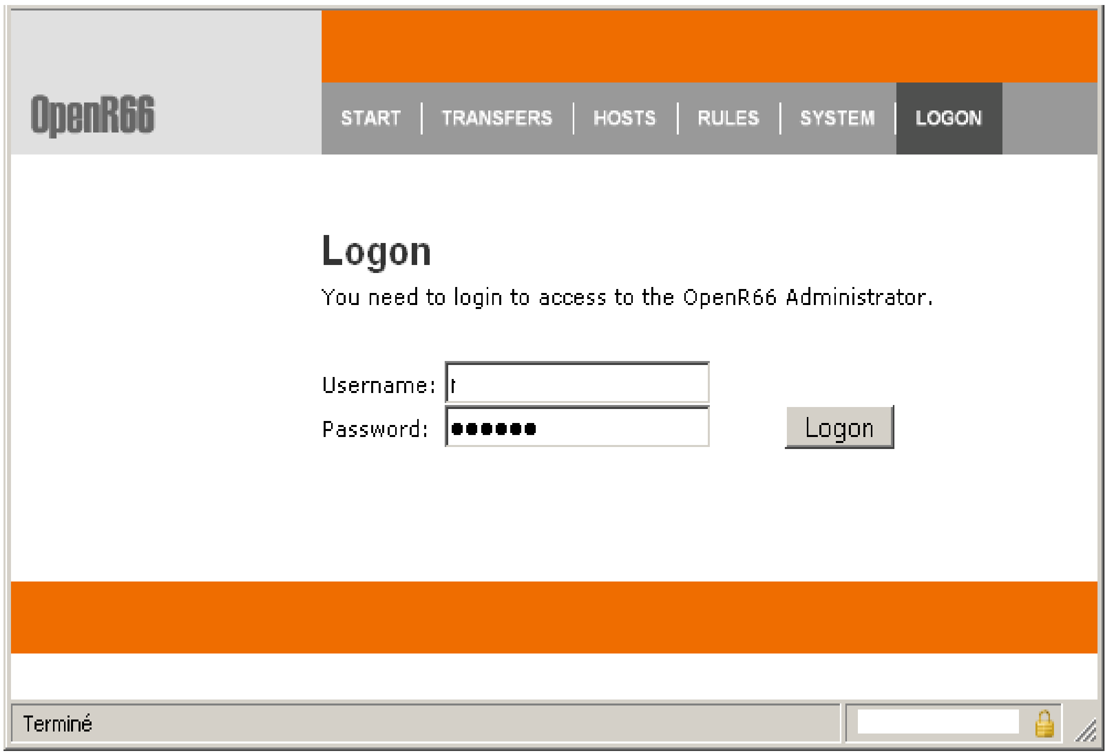
The administrator allows to take action on transfers (stop, restart, ...), export logs, handle hosts and rules, and some specific internal functions. It is accessed at

`https://host_address:8067/`

(8067 port can be changed in the configuration file). Note the 's' in https. You will need to accept the security concern from your browser (specially if the SSL key is generated by yourself).

Here are some pictures from the administrator:

The Logon screen



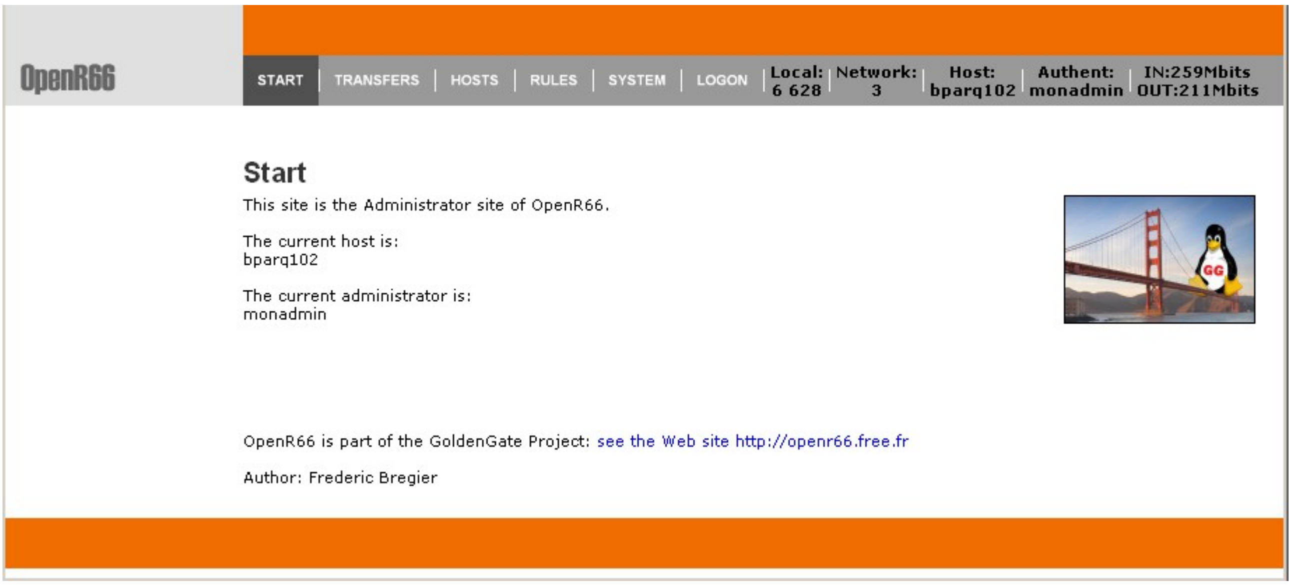
Logon

You need to login to access to the OpenR66 Administrator.

Username:

Password: ●●●●●●

The Start screen



Local: 6 628 | Network: 3 | Host: bparq102 | Authent: monadmin | IN:259Mbits OUT:211Mbits

Start

This site is the Administrator site of OpenR66.

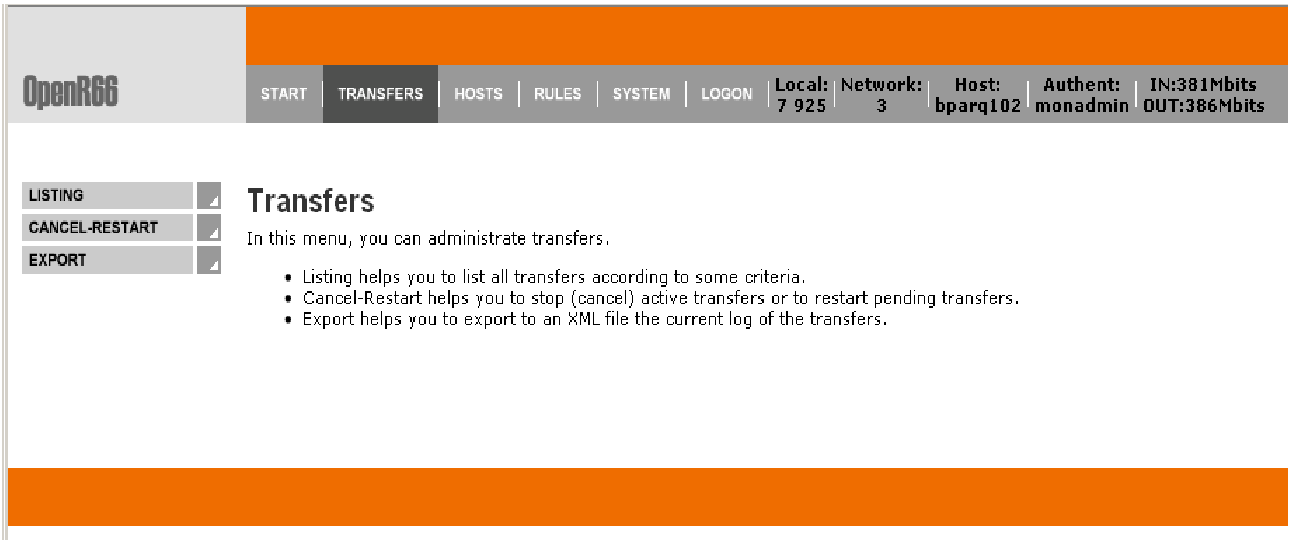
The current host is:
bparq102

The current administrator is:
monadmin

OpenR66 is part of the GoldenGate Project: see the Web site http://openr66.free.fr

Author: Frederic Bregier

The Transfers main screen

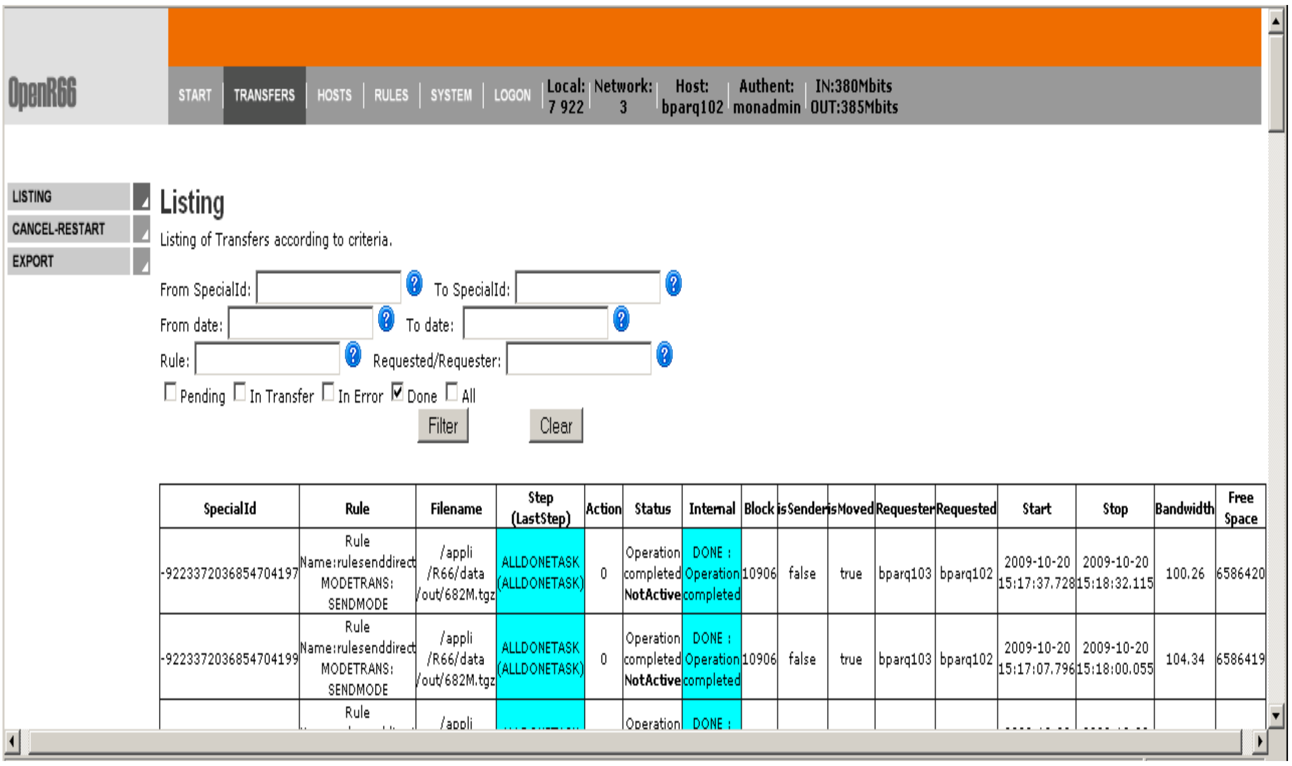


Transfers

In this menu, you can administrate transfers.

- Listing helps you to list all transfers according to some criteria.
- Cancel-Restart helps you to stop (cancel) active transfers or to restart pending transfers.
- Export helps you to export to an XML file the current log of the transfers.

The Transfers Listing screen



Listing

Listing of Transfers according to criteria.

From SpecialId: To SpecialId:

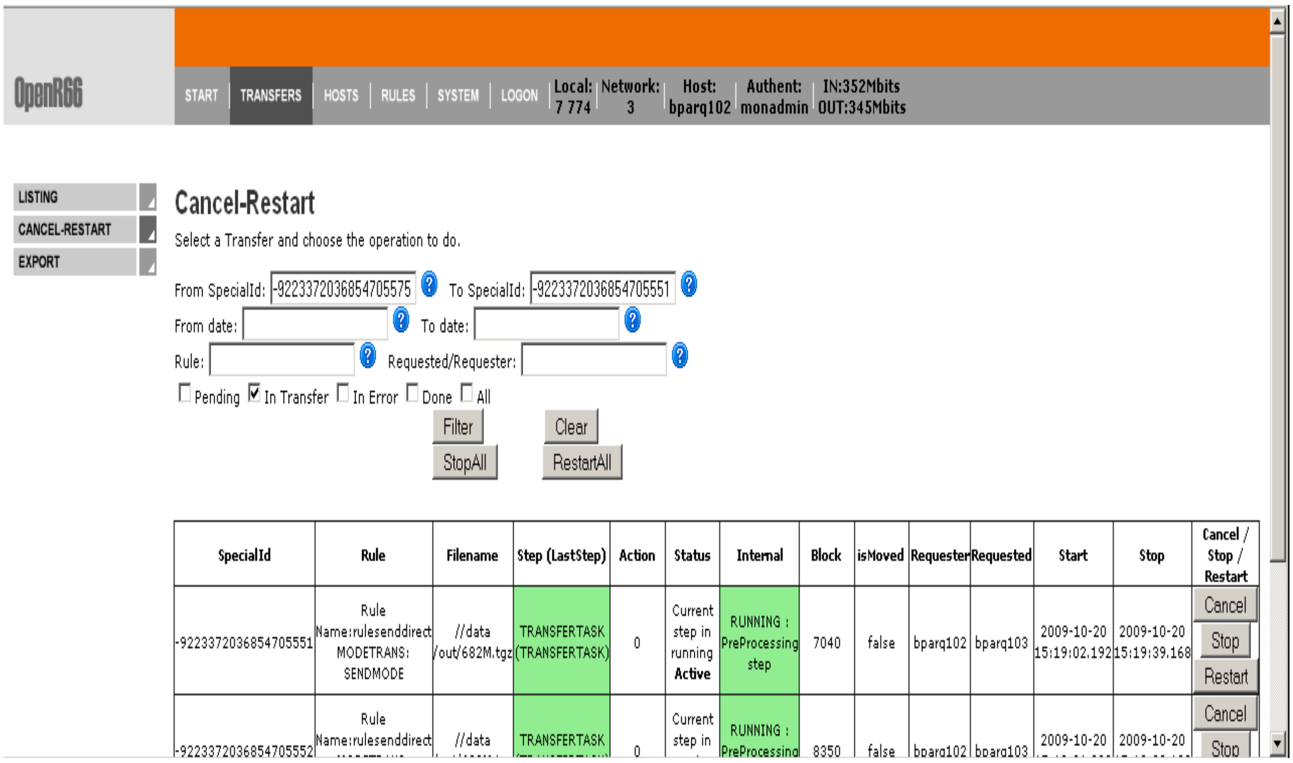
From date: To date:

Rule: Requested/Requester:

☐ Pending ☐ In Transfer ☐ In Error ☑ Done ☐ All

SpecialId	Rule	Filename	Step (LastStep)	Action	Status	Internal	Block	isSender	isMoved	Requester	Requested	Start	Stop	Bandwidth	Free Space
-9223372036854704197	Rule Name:rulesenddirect MODETRANS: SENDMODE	/appli /R66/data /out/682M.tgz	ALLDONETASK (ALLDONETASK)	0	Operation completed **NotActive**	DONE : Operation completed	10906	false	true	bparq103	bparq102	2009-10-20 15:17:37.728	2009-10-20 15:18:32.115	100.26	6586420
-9223372036854704199	Rule Name:rulesenddirect MODETRANS: SENDMODE	/appli /R66/data /out/682M.tgz	ALLDONETASK (ALLDONETASK)	0	Operation completed **NotActive**	DONE : Operation completed	10906	false	true	bparq103	bparq102	2009-10-20 15:17:07.796	2009-10-20 15:18:00.055	104.34	6586419
	Rule	/appli			Operation	DONE :									

The Cancel-Restart Transfer screen



Local: 7 774 | Network: 3 | Host: bparq102 | Authent: monadmin | IN:352Mbits OUT:345Mbits

Cancel-Restart

Select a Transfer and choose the operation to do.

From SpecialId: -9223372036854705575 To SpecialId: -9223372036854705551

From date: To date:

Rule: Requested/Requester:

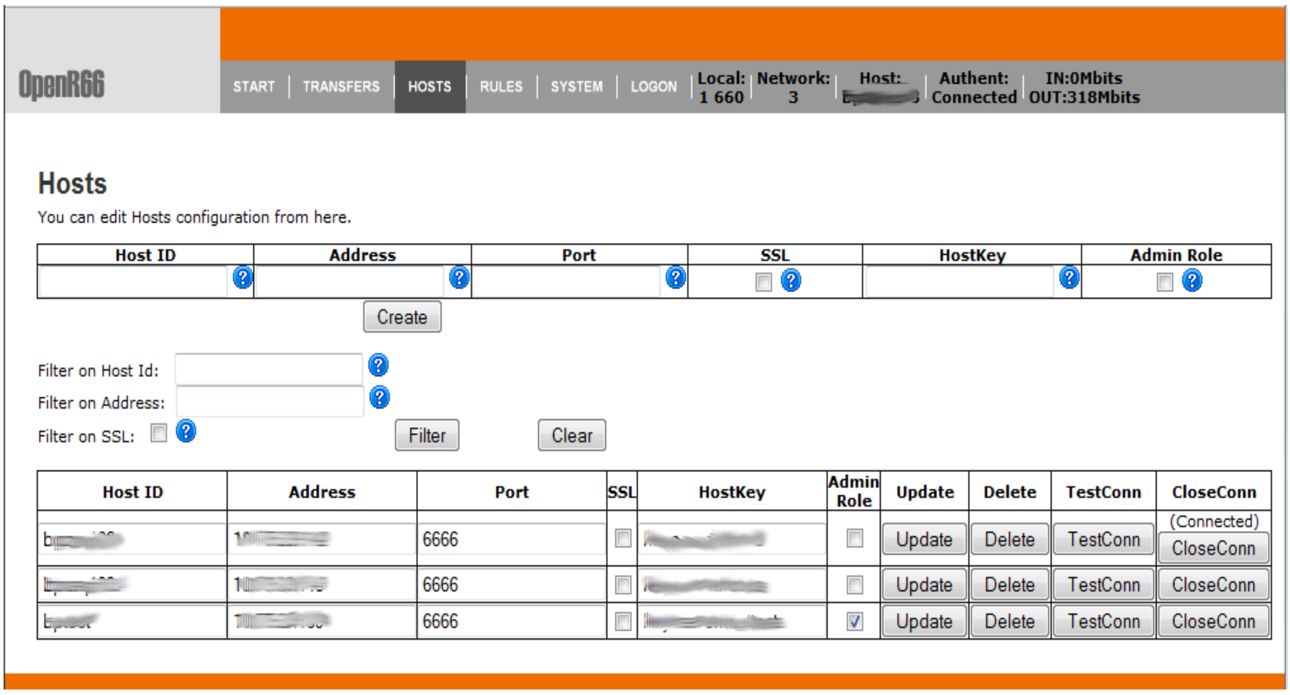
☐ Pending ☑ In Transfer ☐ In Error ☐ Done ☐ All

Filter Clear

StopAll RestartAll

SpecialId	Rule	Filename	Step (LastStep)	Action	Status	Internal	Block	isMoved	Requester	Requested	Start	Stop	Cancel / Stop / Restart
-9223372036854705551	Rule Name:rulesenddirect MODETRANS: SENDMODE	//data /out/682M.tgz	TRANSFERTASK (TRANSFERTASK)	0	Current step in running **Active**	RUNNING : PreProcessing step	7040	false	bparq102	bparq103	2009-10-20 15:19:02.192	2009-10-20 15:19:39.168	Cancel Stop Restart
-9223372036854705552	Rule Name:rulesenddirect	//data	TRANSFERTASK	0	Current step in	RUNNING : PreProcessing	8350	false	bparq102	bparq103	2009-10-20	2009-10-20	Cancel Stop

The Hosts Screen



Local: 1 660 | Network: 3 | Host: [redacted] | Authent: Connected | IN:0Mbits OUT:318Mbits

Hosts

You can edit Hosts configuration from here.

Host ID	Address	Port	SSL	HostKey	Admin Role
			☐		☐

Create

Filter on Host Id:

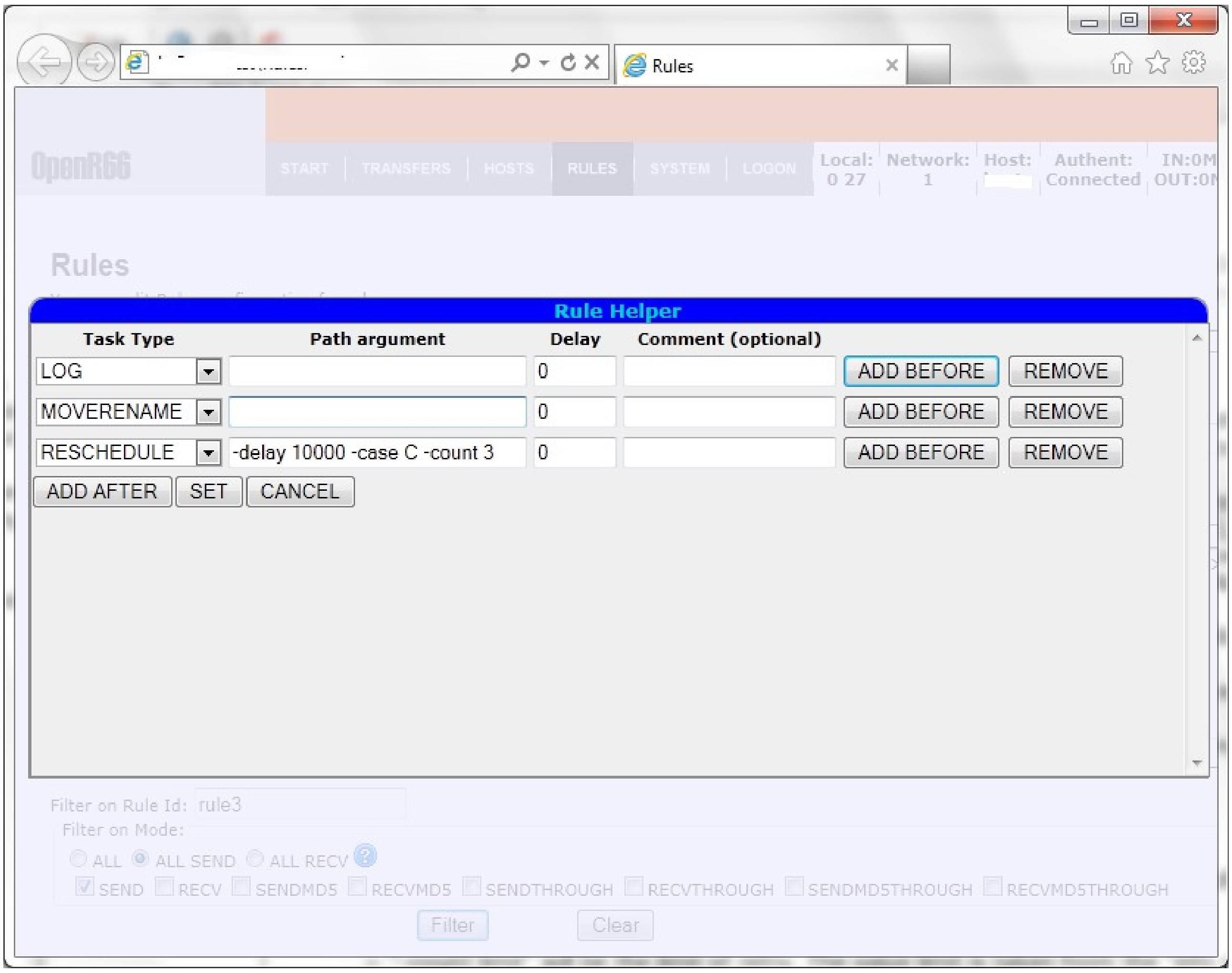
Filter on Address:

Filter on SSL: ☐

Filter Clear

Host ID	Address	Port	SSL	HostKey	Admin Role	Update	Delete	TestConn	CloseConn
[redacted]	[redacted]	6666	☐	[redacted]	☐	Update	Delete	TestConn	(Connected) CloseConn
[redacted]	[redacted]	6666	☐	[redacted]	☐	Update	Delete	TestConn	CloseConn
[redacted]	[redacted]	6666	☐	[redacted]	☑	Update	Delete	TestConn	CloseConn

The Rules screen



Rule Helper

Task Type	Path argument	Delay	Comment (optional)		
LOG		0		ADD BEFORE	REMOVE
MOVERENAME		0		ADD BEFORE	REMOVE
RESCHEDULE	-delay 10000 -case C -count 3	0		ADD BEFORE	REMOVE

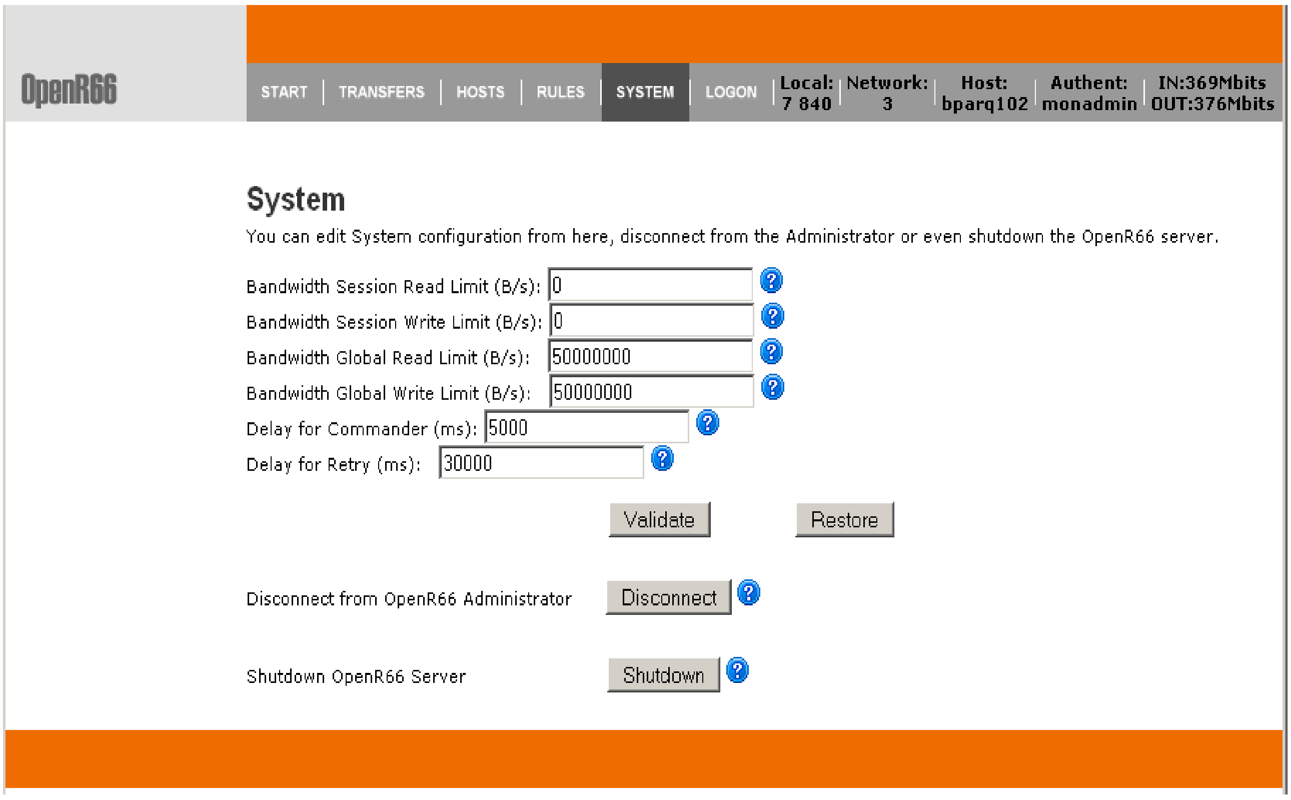
Filter on Rule Id: rule3

Filter on Mode:

ALL [✓ ALL SEND] ALL RECV

[✓ SEND] RECV SENDMD5 RECVMD5 SENDTHROUGH RECVTHROUGH SENDMD5THROUGH RECVMD5THROUGH

The System screen



Local:	Network:	Host:	Authent:	IN:369Mbits
7 840	3	bparq102	monadmin	OUT:376Mbits

System

You can edit System configuration from here, disconnect from the Administrator or even shutdown the OpenR66 server.

Bandwidth Session Read Limit (B/s):	0
Bandwidth Session Write Limit (B/s):	0
Bandwidth Global Read Limit (B/s):	50000000
Bandwidth Global Write Limit (B/s):	50000000
Delay for Commander (ms):	5000
Delay for Retry (ms):	30000

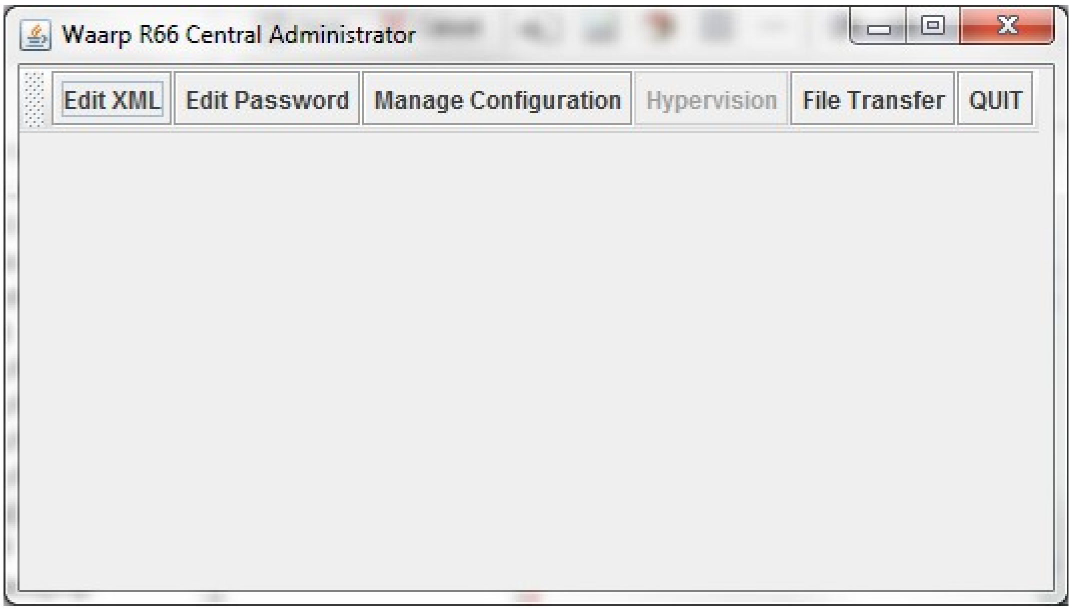
Disconnect from OpenR66 Administrator

Shutdown OpenR66 Server

A new option allow to "block" or "unblock" new requests, such that someone can wait that all current requests are over to shutdown his/shis Waarp server without interrupting requests.

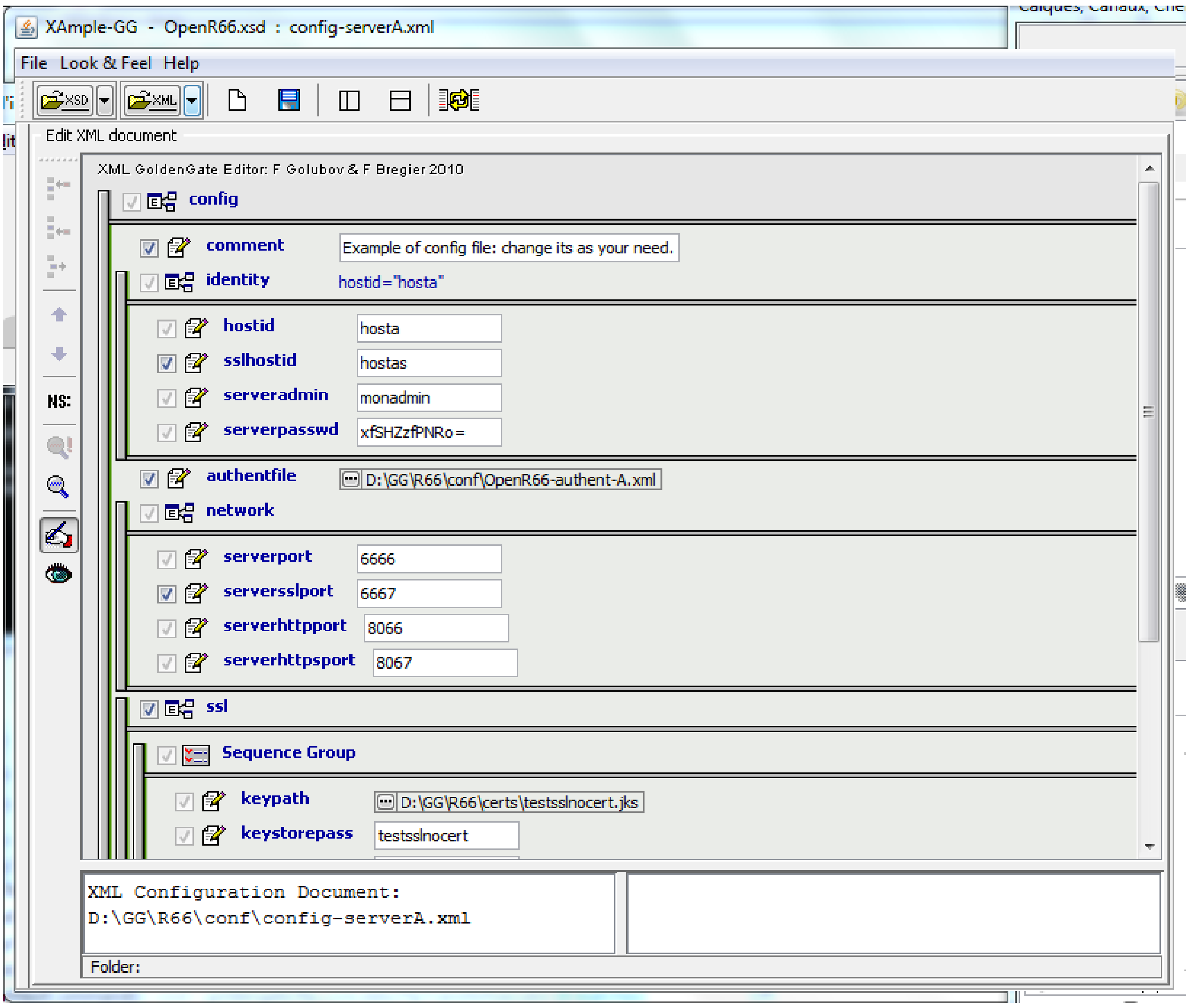
Waarp Central Administrator

The Waarp Administrator assembles several tools from Waarp sub projects into one:



To be able to use it, you must first create a Client configuration XML file, then add this "client" as a "FullAdmin" in each of the R66 server configuration you would like to act on.

- Waarp Xml Editor : in order to edit XML configuration file

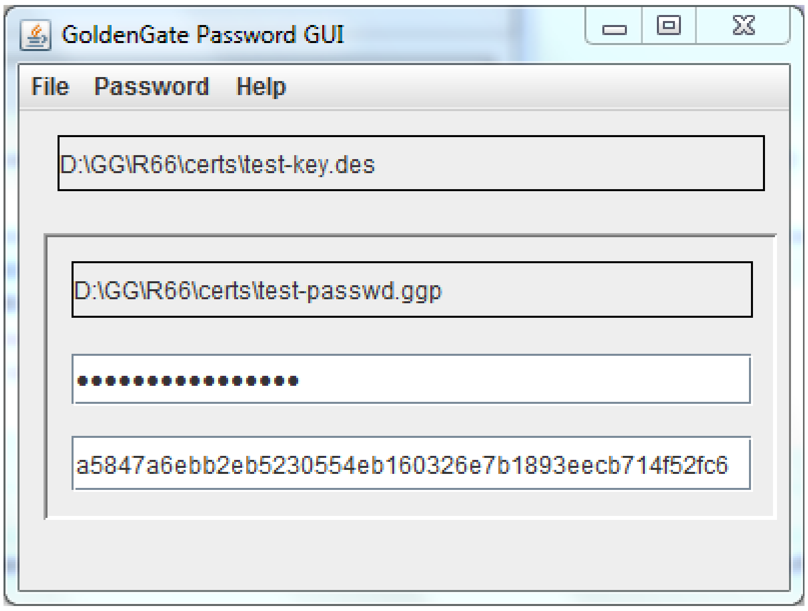

XAmple-GG - OpenR66.xsd : config-serverA.xml
File Look & Feel Help
Edit XML document
XML GoldenGate Editor: F Golubov & F Bregier 2010

config
comment: Example of config file: change its as your need.
identity: hostid="hosta"
hostid: hosta
sslhostid: hostas
serveradmin: monadmin
serverpasswd: xfSHZzfPNRo=
authentfile: D:\GG\R66\conf\OpenR66-authent-A.xml
network
serverport: 6666
serversslport: 6667
serverhttpport: 8066
serverhttpsport: 8067
ssl
Sequence Group
keypath: D:\GG\R66\certs\testsslnocert.jks
keystorepass: testsslnocert

```
XML Configuration Document:
D:\GG\R66\conf\config-serverA.xml
```

Folder:

- Waarp Password : in order to be able to create des key file and ggp password files



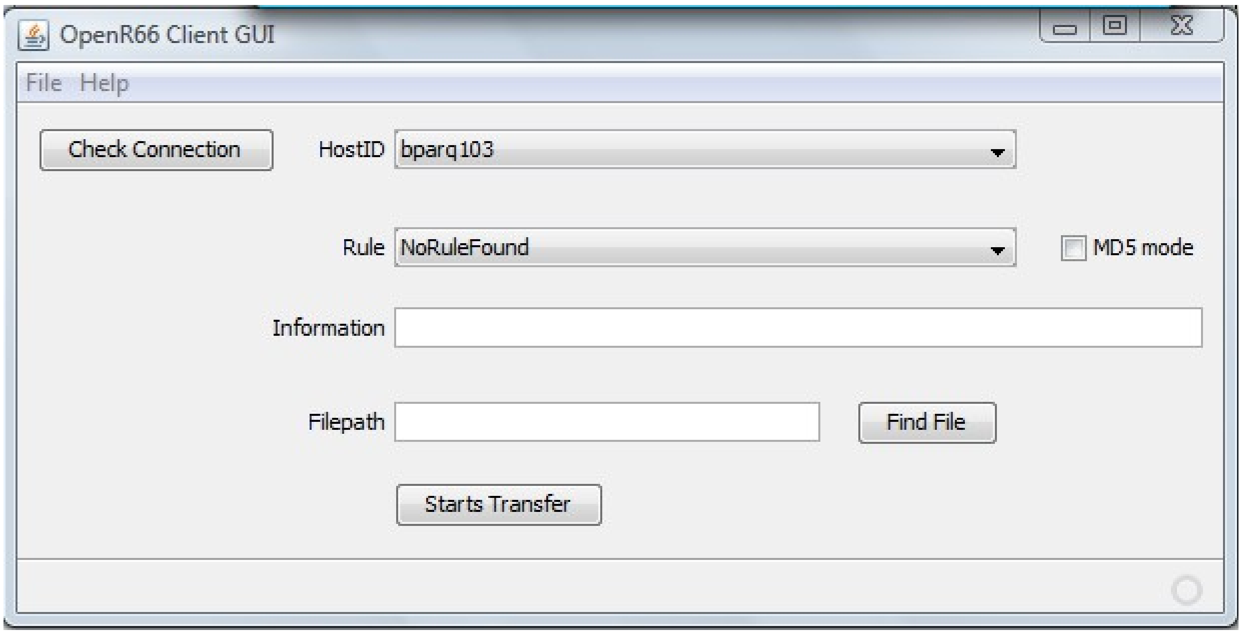
D:\GG\R66\certs\test-key.des

D:\GG\R66\certs\test-passwd.ggp

●●●●●●●●●●●●●●●●●

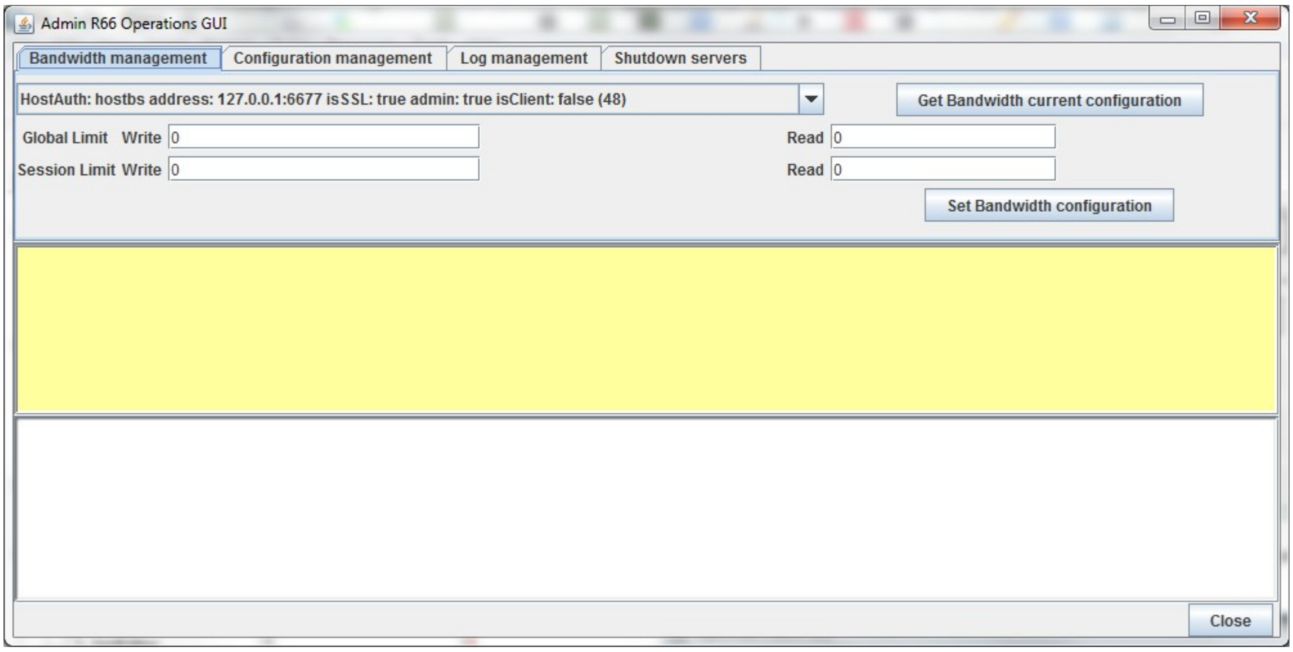
a5847a6ebb2eb5230554eb160326e7b1893eecb714f52fc6

- Waarp R66 Gui : in order to be able to directly comunicate with R66 servers from the Central Administrator GUI (for testing or for real usage)



It allows also some other functionalities, bringing native Server functions within this Central Administrator GUI in remote mode:

- Get or Set Bandwicth limitation

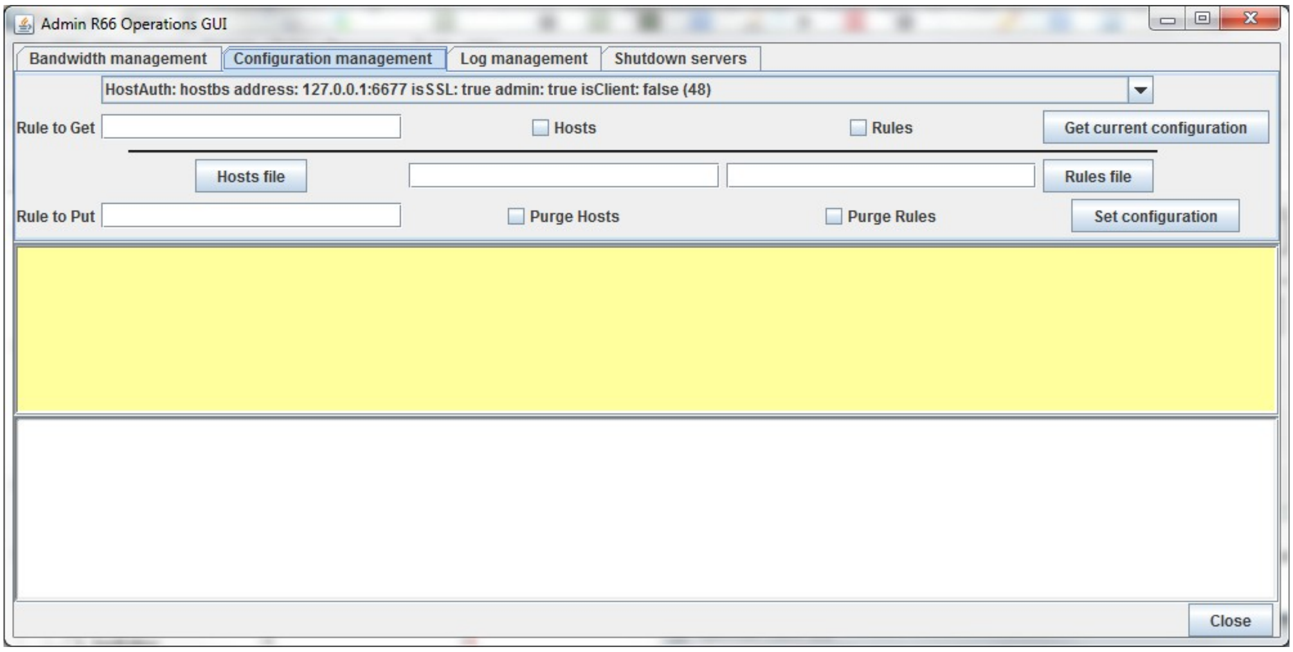


HostAuth: hostbs address: 127.0.0.1:6677 isSSL: true admin: true isClient: false (48)

Global Limit Write 0 Read 0

Session Limit Write 0 Read 0

- Export or Import Host / Rules configuration

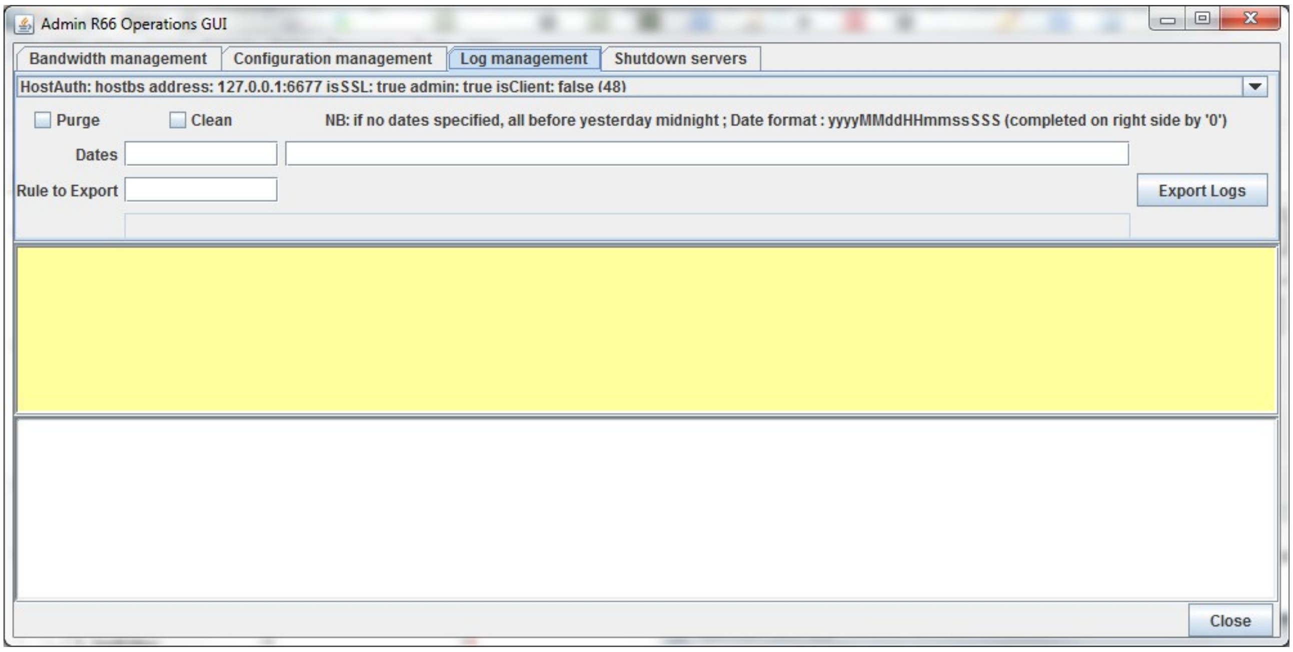


HostAuth: hostbs address: 127.0.0.1:6677 isSSL: true admin: true isClient: false (48)

Rule to Get ☐ Hosts ☐ Rules

Rule to Put ☐ Purge Hosts ☐ Purge Rules

- Log export



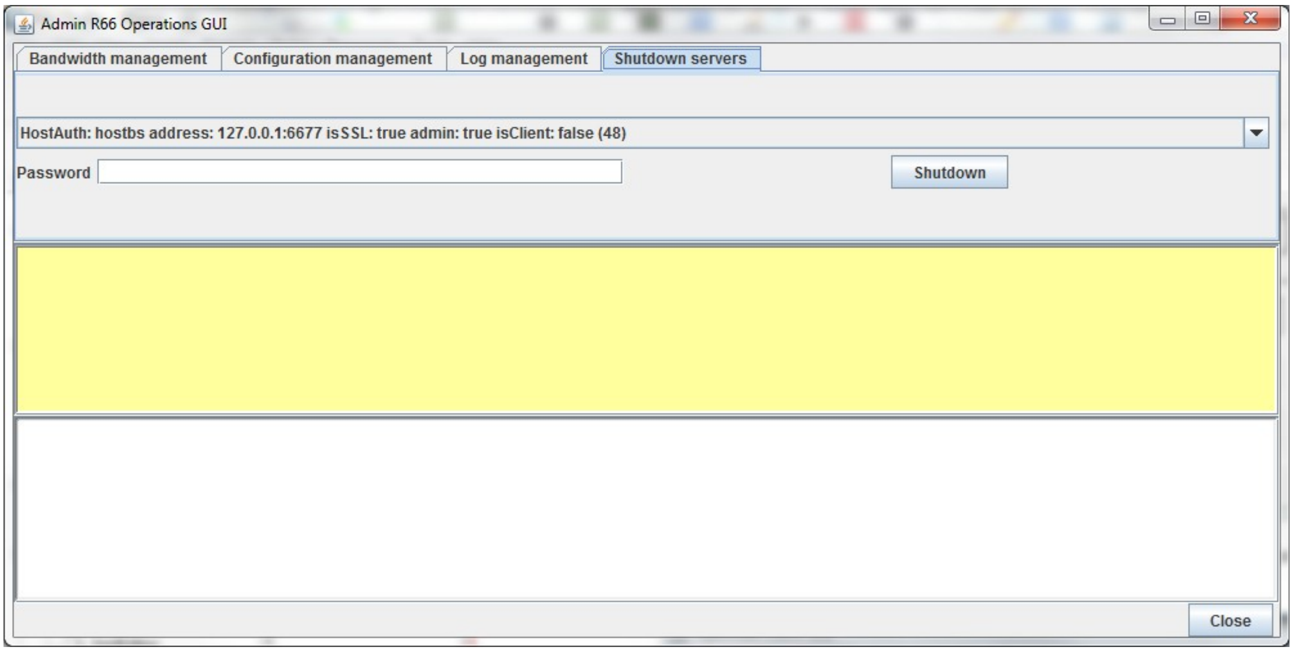
HostAuth: hostbs address: 127.0.0.1:6677 isSSL: true admin: true isClient: false (48)

☐ Purge ☐ Clean NB: if no dates specified, all before yesterday midnight ; Date format : yyyyMMddHHmmssSSS (completed on right side by '0')

Dates

Rule to Export

- Shutdown



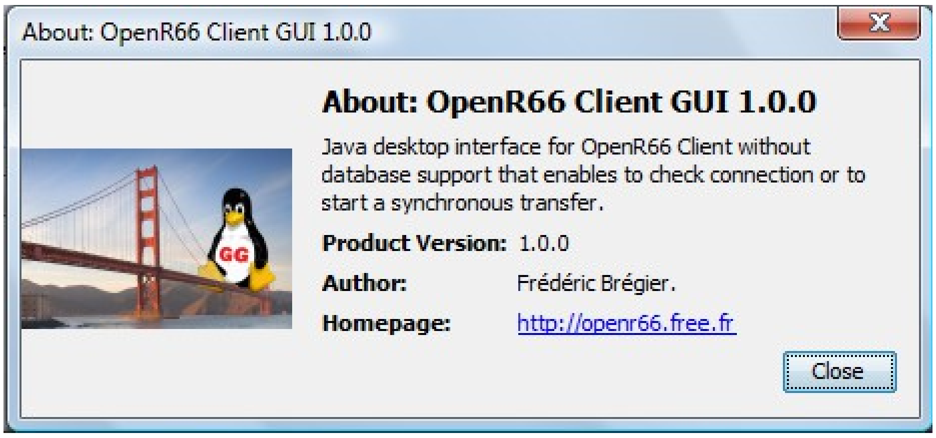
HostAuth: hostbs address: 127.0.0.1:6677 isSSL: true admin: true isClient: false (48)

Waarp R66 GUI

A Graphical User Interface has been done to enable:

- Checking the connection to a remote Host
- Initiate a synchronous transfer to a remote Host

Note that the client is by default without any database support, meaning that the client cannot be the target of a new transfer initiated by a remote host. So the reason that transfers allowed are only synchronous one, but they can be either in receive or send mode.


About: OpenR66 Client GUI 1.0.0

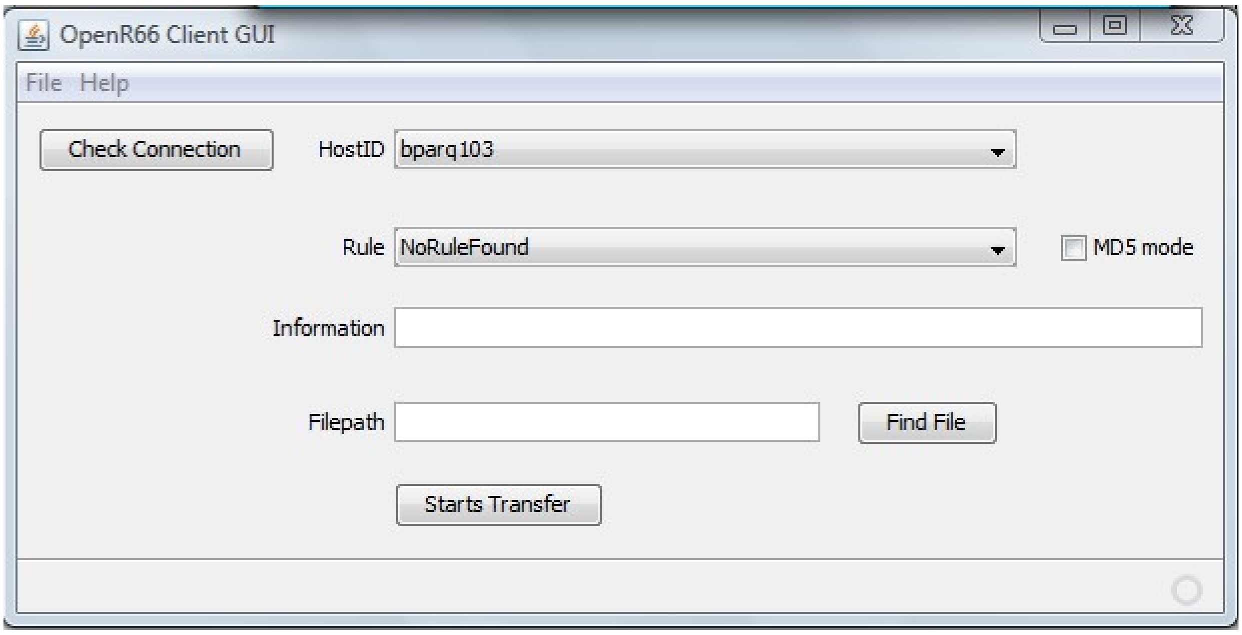
About: OpenR66 Client GUI 1.0.0

Java desktop interface for OpenR66 Client without database support that enables to check connection or to start a synchronous transfer.

Product Version: 1.0.0

Author: Frédéric Brégier.

Homepage: http://openr66.free.fr


OpenR66 Client GUI

HostID: bparq103

Rule: NoRuleFound

MD5 mode: [ ]

Information:

Filepath:

From version 2.1.2, the filepath is now changed to an URL notation as file:/path in order to cover the Windows path with "space" character and "accents" in names.

Thrift support for R66

Waarp Thrift is repository of Thrift definitions for Waarp project.

It implements a Thrift interface. It allows one to develop his own interface to one Waarp project using his own langage (according to Thrift support).

The first item is for Waarp R66.

The “usethrift” option (specifying a port > 0) allows to enable the Thrift server support in one R66 server. Currently only Synchronous Binary thrift protocol is allowed, so a client should use “Tsocket” for its Ttransport and “TbinaryProtocol” for its Tprotocol.

One example in Java is given in org.waarp.openr66.protocol.test.TestThriftClientExample to show how to interact with the Thrift R66 service.

This option should enables more capabilities to R66 to be embedded in existing applications, in a more large cover than just Java.

The current methods available are:

- `transferRequestQuery`: allows to initiate a submitted transfer from the related R66 server to another one. Note that the request could be asynchronous (immediately returns once the request is submitted) or synchronous (returns only once the request is done, whatever in error or in success, but it does not take into account future reschedule if any as it will return the status once the current try is over).
- `InfoTransferQuery`: allows to request some information on one particular transfer request
- `isStillRunning`: allows to quickly have the information on one particular transfer request running status or not
- `infoListQuery`: allows to get the information on file list on the local R66 server

Note that the Thrift service, for security reason, is only opened on 127.0.0.1 address since no authentication is made, as it stands for a local service.

Waarp Local Exec

This daemon enables to execute efficiently external command like System.exec using an independent daemon. The main idea behind is that each System.exec will fork the current Java process. Therefore with high memory usage, this fork takes long time and is memory limited. With this daemon, the memory footprint is limited to a minimum and therefore the fork costs really less than standard JVM process. The gain is about 2 to 3 times faster than internal standard execution.

No SSL version takes 3 optional arguments:

- no argument: implies 127.0.0.1 + 9999 port
- arguments:
  - `"addresse" "port"`
  - `"addresse" "port" "default delay"`

SSL version takes 3 to 8 arguments (last 5 are optional arguments):

- mandatory arguments: `filename keystorepaswwd keypassword`
- if no more arguments are provided, it implies 127.0.0.1 + 9999 port and no certificates
- optional arguments:
  - `"port"`
  - `"port" "trustfilename" "trustpassword"`
  - `"port" "trustfilename" "trustpassword" "addresse"`
  - `"port" "trustfilename" "trustpassword" "addresse" "default delay"`

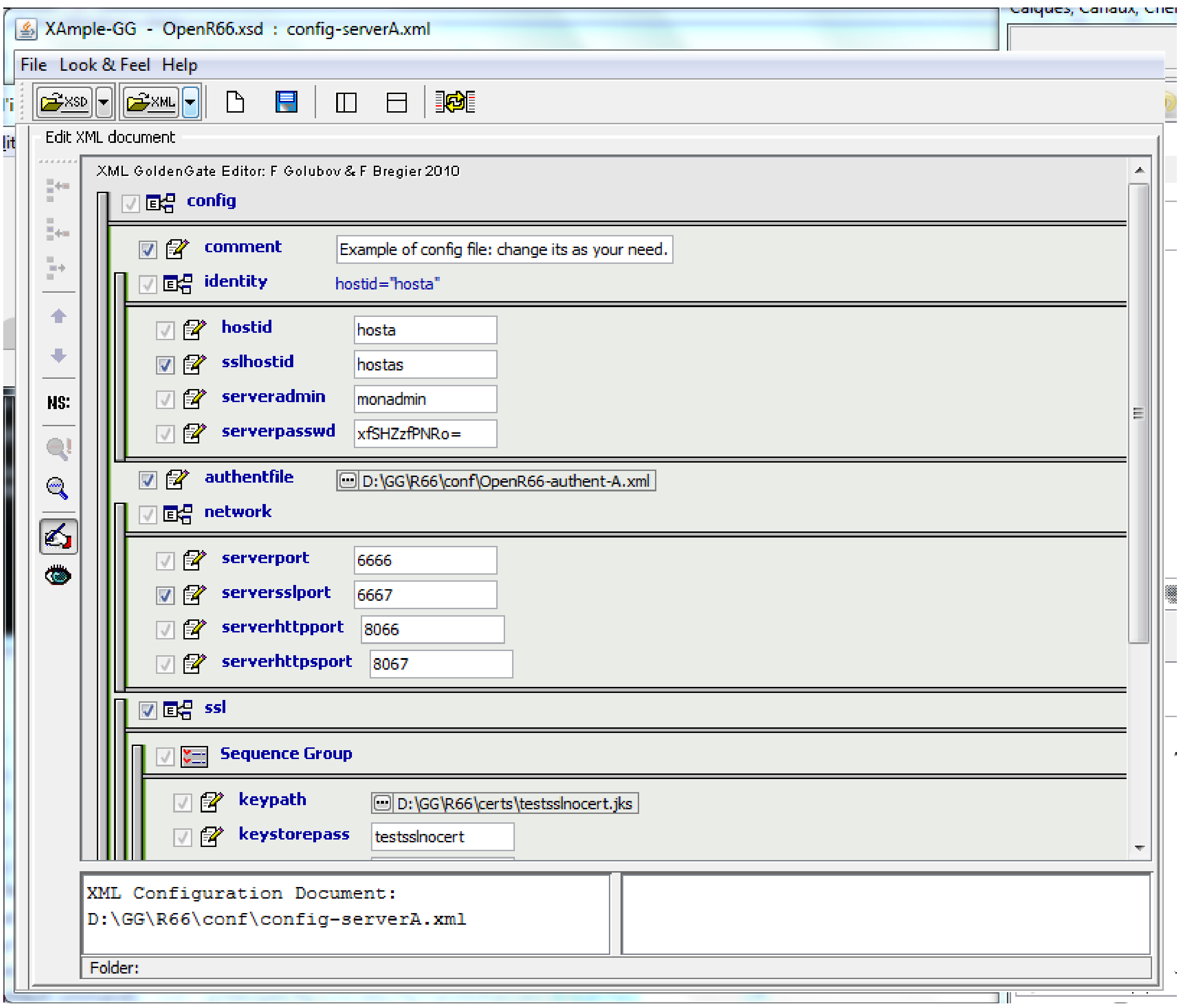
Waarp Xml Editor

To configure an Waarp programs, most of the time you need to first create the XML files as needed.

To help the administrator to generate correct files, XSD models are defined to be used with an extension of the project Xample (XML Gui Editor) from Felix Golubov. This program was extended to support more flexibility on File and Directory selection.

We provide both program (original one from Felix Golubov and the extended one). By no means Waarp is proprietary of this software. The only work was to extend through the API given by its original creator: Thank to Felix Golubov.

Below is an example of the GUI.



XML GoldenGate Editor: F Golubov & F Bregier 2010

- config
  - comment: Example of config file: change its as your need.
  - identity hostid="hosta"
    - hostid: hosta
    - sslhostid: hostas
    - serveradmin: monadmin
    - serverpasswd: xfSHZzfPNRo=
  - authentfile: D:\GG\R66\conf\OpenR66-authent-A.xml
  - network
    - serverport: 6666
    - serversslport: 6667
    - serverhttpport: 8066
    - serverhttpsport: 8067
  - ssl
    - Sequence Group
      - keypath: D:\GG\R66\certs\testsslnocert.jks
      - keystorepass: testsslnocert

```
XML Configuration Document:
D:\GG\R66\conf\config-serverA.xml
```

Waap Password Tool

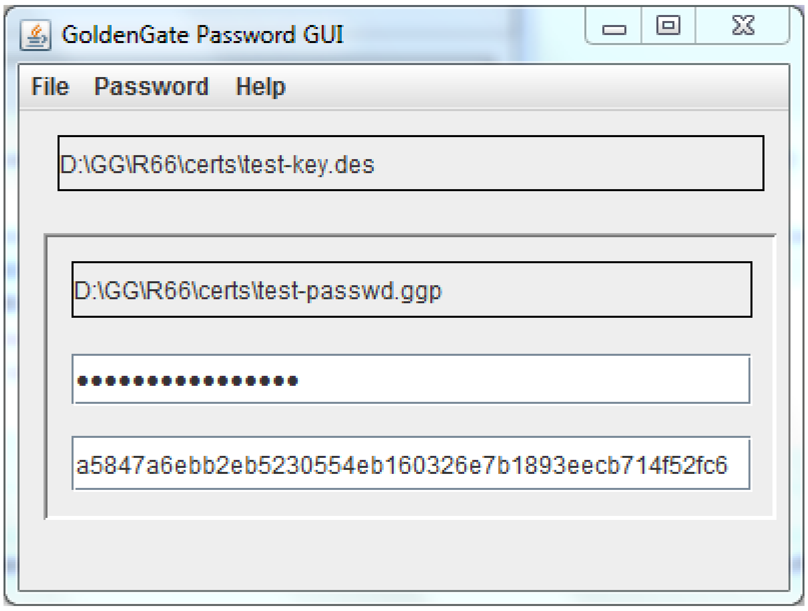
For Waarp project, you might need to provide a password in the format accepted by Waarp. You need a specific DES Key for the crypted password support. DES crypted support and generation are available through the Waarp Password GUI project.

It enables the creation of DES key file and the associated password file (Waarp Password was GoldenGate Password = GGP).

Both a GUI and command are available.

Command options are:

* -ki file to specify the Key File by default
* -ko file to specify a new Key File to build and save
* -pi file to specify a GGP File by default(password)
* -des to specify DES format (default)
* -blf to specify BlowFish format
* -pwd to specify a clear password as entry
* -cpwd to specify a crypted password as entry
* -po file to specify a GGP File as output
* -clear to specify uncrypted password shown as clear text



D:\GG\R66\certs\test-key.des

D:\GG\R66\certs\test-passwd.ggp

●●●●●●●●●●●●●●●●

a5847a6ebb2eb5230554eb160326e7b1893eecb714f52fc6

Waarp FTP

The Waarp project starts with a new fresh FTP(S) server implementation, fully in Java according to the following RFC:

- RFC 959
- RFC 775
- RFC 2389
- RFC 2428
- RFC 3659
- RFC 4217

It includes also the following extra commands from version 0.9.2:

- XCRC to compute CRC on a remote file
- XMD5 to compute MD5 on a remote file
- XSHA1 to compute SHA-1 on a remote file
- INTERNALSHUTDOWN to allow to shutdown the server (protected command)

It is based mainly on the NETTY framework (NIO great framework support) and is tend to be really efficient, both in term of memory, threads and network bandwidth. Bandwidth limitation can be configured both in store and retrieve, per session (although only one value is set by default, but it can be changed dynamically if needed) or globally to the server and of course with no limitation at all if wanted. Limitation should be enough to change the bandwidth behaviour for instance depending on the time in the day, so as to allow to limit bandwidth usage when users are at work and in contrary to allow more speed when only batchs are running.

The specificity of this project is that you can adapt this software to your particular needs by:

- changing the pre or post action on commands (not ony transfer),
- changing the underlying representation of files and directories (for instance with database entries),
- using any particular authentication mechanism.

Currently this FTP Server handle the following implementations but you can extend it to fit your needs:

- File and Directory can be true File and Directory (default version), but they can also be something else, for instance data from database, from LDAP or whatever you want, as long as you can implement the interface (DirInterface and FileInterface).

- Authentication can be whatever you want: file based (default version), or others like database, LDAP or whatever you want, as long as you can implement the interface (AuthInterface)
- Actions on pre or post commands can be configured from 2 classes, one business from control connection and one business from data connection (abstract classes BusinessNetworkHandler and DataBusinessNetworkHandler)
- Logger can be setup as you want: SLF4J (default using LOGBACK) or whatever you want, as long as you can implement the abstract classes (FtpInternalLogger and FtpInternalLoggerFactory)

So at most, if you want to implement all new, you will have to implement only 7 classes, plus of course the main class one that will launch the server itself.

If you just want a simple FTP Server, then you've just have to instantiate the main class and reused the default implementation that is proposed. The SimpleImpl package shows one example.

Extending the server should not be difficult. For instance, adding a new command is as easy as:

- Create a new class that extends the AbstractCommand class
- Add a reference to it into the FtpCommandCode class

That's it!

Adding a new response code is even easier, you have just to add an entry in FtpReplyCode class.

From version 1.1.2, FTP server allows SSL support (FTPS) both as Implicit (native SSL/TLS support on wire level) and Explicit (using commands such as AUTH and PROT).

This project is in production in the French Ministery of Finances to enable file transfers from an FTP protocol (client side) to an Waarp R66 protocol (server side protocol) since end of 2007.

Configuration of Waarp FTP

How to configure Waarp Ftp using default implementation

In the zip distribution WaarpFtp-X.Y.Z-dist.zip, you will find an extra directory named: src/main/config where two examples files are: config.xml and authent.xml. Those two files are related to the simple implementation of the Ftp Server.

The first one (config.xml) specifies the general behavior:

- server password (for admin functions)
- server port, home directory, bandwidth limitation, optional server address (in case of NAT for instance)
- server special configuration like threads number, timeout of connection, deleteOnAbort, usenio, fastmd5, blocksize, rangeport and the localization of the authentfile (second file)

The second one is a simple authentication file containing user definition (user name, password, account, admin status).

These two files are for the simple implementation (which is a simple FtpServer without any special actions), and using a simple authentication mechanism (from authent.xml). The simple Ftp Server is included in the WaarpGatewayFtp-Simpleimpl-X.Y.Z.jar . It can be started like:

java ... classpath and jvm settings ... org.waarp.ftp.simpleimpl.SimpleGatewayFtpServer src/main/config/config.xml

Note that in the classpath you need :

- External jars:
  - Netty,
  - Apache Common IO,
  - DOM4J,
  - Jaxen,
  - Logback
- Waarp jars:
  - WaarpDigest,
  - WaarpCommon
  - and of course the WaarpGatewayFtp-Core
  - probably the WaarpGatewayFtp-Filesystem (except if you want to implement virtual Directory and File in something different than true directories and files)
  - (and if you are ok with the default implementation) the WaarpGatewayFtp-Simpleimpl jars.

Some of them can be replaced if you change the implementation (Common IO is optional, Dom4J and Jaxen are only used if XML configuration file is used, Logback could be replaced with other logger framework, see the org.waarp.common.logging from WaarpCommon where JDK is also supported, others can easily be supported).

If you want to implement something different, here are the files you may want to re write. They can all be found in the simpleimpl sub package at Waarp.ftp.simpleimpl package. A perfect example is the WaarpGatewayFtp sub project.

- the server startup itself : SimpleGatewayFtpServer
- the Configuration file : config.FileBasedConfiguration

By default an XML file is used. If you want to use other ways to implement the load of the configuration, you can change this implementation by the one you want (for instance from a database).

If you want to include some others configurations properties and XML is ok for you, you can extend it to fit your needs.

- the virtual Directory, File implementation : file.FileBasedDir, file.FileBasedFile

For instance, if you want to implement File and Directory as content in a database, you will have to not use the WaarpFtp-Filesystem-X.Y.Z.jar package an to implement your own File and Directory representation.

If you are OK with real file and directory, then nothing has to be done.

- the authentication system : file.FileBasedAuth and its associated file.SimpleAuth

For instance, if you want to inherit authentication from a LDAP, you will have to implement your own extension of org.waarp.ftp.filesystembased.FilesystemBasedFtpAuth instead of this simple one.

- then you have two classes that implements business actions on pre and post actions on transfer:
  - control.SimpleBusinessHandler : for the control connection of FTP service
  - data.FileSystemBasedDataBusinessHandler : for the data connection of FTP service

By default, this classes does nothing except logging.

To shutdown the service, either you do a CTRL-C (or better if Unix but not IBM JDK kill -SIGUSR1 <processID>), or better you connect as an admin user and execute the special command:

```
internalshutdown <password>
```

where the password is the one in the config.xml for admin actions.

For instance, using FTP from windows you have to type `>quote internalshutdown password`

Note that this FTP Server can be used as a simple one, but also for more complicated cases. It is in the middle of a self FTP server (simpleimpl package) and a framework to implement its own FTP service, which is the main reason of this project since I've not found any other open source implementations allowing to have pre or post actions on transfers or commands.

WaarpGatewayFtp

A specific package names WaarpGatewayFtp is a real example (in production) of a Waarp Gateway FTP server implementing pre or post actions on transfers and links with Waarp R66.

It add also some functionalities like:

- The ability to change dynamically the authentication through an extended SITE command
- The ability to specify explicit command to be executed before (RETR) or after (STOR like operations)

From V2.0

- The ability to specify command for each User
- The ability to save logs of transfers in a database (optional)
- An administrator interface in HTTPS
- The ability to use the Waarp LocalExec Daemon instead of internal System.exec()
- The ability to use limitation on CPU or number of connections
- The support of SNMP agent included in the WaarpFtpExec daemon

From version 2.1.2, FTPS is supported (both Implicit and Explicit forms).

From version 2.1.4, JAVAEXEC is allowed, with an implementation similar to R66. A default Log action is available for testing.

From version 2.1.6, using a JAVAEXEC and the class JavaExecutorWaarpFtp4jClient from the module Waarp FTP Client (see Waarp Commons), it is possible to use a remote FTP server as target of this Gateway.

The configuration files are specific and different than WaarpGatewayFtp standard server. Like for Waarp R66, xsd files are provided and compatible with the Waarp XmlEditor (Waarp Xml Editor).

The SSL support for the administrator is configurable as for Waarp R66 (see For SSL connection without authentication of clients).