Trebuchet

A multi-scheme file-transfer API and client library

Downloads/Documentation.

- There is full support for Trebuchet in Ogrescript. One can run Trebuchet in Ogrescript by downloading and using the most recent version of **ELF**.
- The Trebuchet libraries can also be retrieved in a stand-alone distribution from **Trebuchet Stand-alone Libraries**. Included in the latter is also a small shell script (UNIX systems) for restarting interrupted or failed operations (see below).
- Trebuchet Javadocs can also be accessed at: **API**.
- For configuration options, see **Configuration Options**.

Purpose

The principal aim of this library is to provide an abstraction layer over the standard file-related operations (such as directory listing, directory creation, file transfer and file deletion) which allows switching between protocols without alteration of either source code or scripting. The library is also designed to be extensible so that new protocol support can be added in a reasonably clean manner.

Currently, there are two ways in which Trebuchet can be utilized:

- As a normal Java package import (i.e., programmatic calls to the library by other Java code);
- Through the available Ogrescript tasks: see the Ogrescript Trebuchet plug-in.

Certain capabilities, such as restarting operations and inspecting the binary "cache" files, are also available from the command-line. There are plans to provide sometime in the near future an Eclipse-based RCP Trebuchet client for easy management of file transfers across multiple hosts.

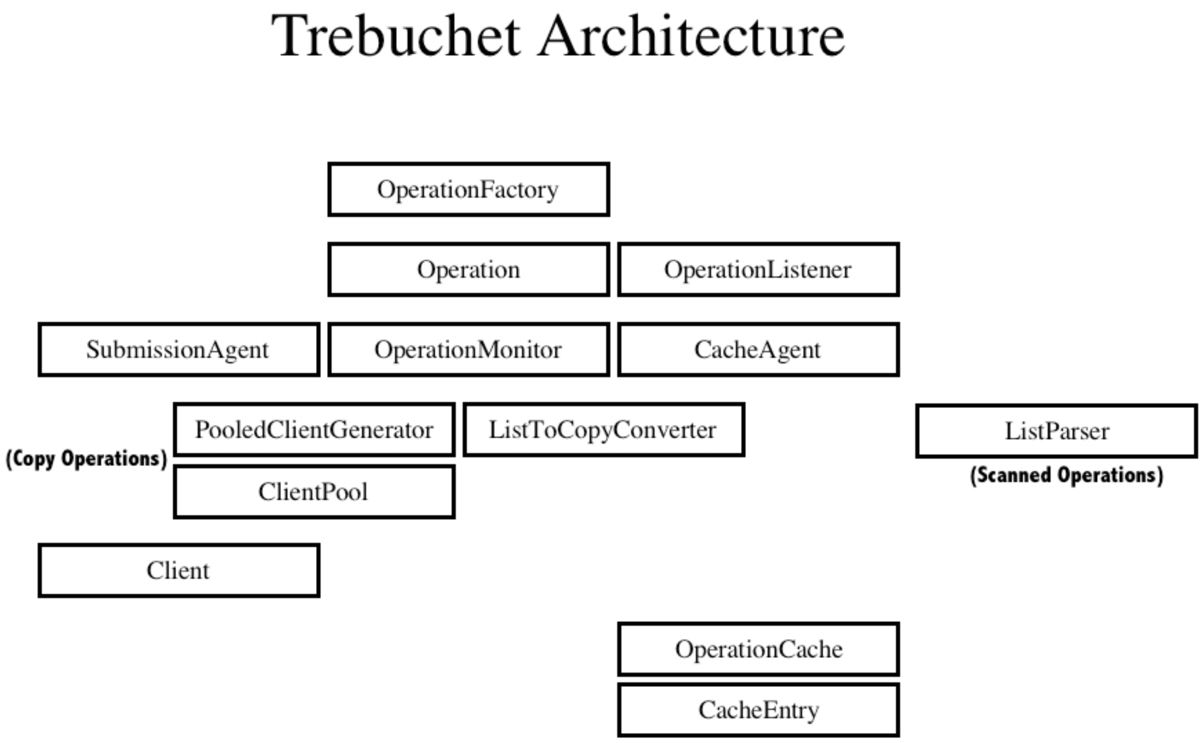
Features

The following are some of the more salient features offered by the Trebuchet library/tasks:

1. Full support for UNIX-style operations (`ls`, `touch`, `mkdir`, `cp`, `mv`, `rm`) locally and via the SSH/SCP protocols.
2. Support for all of these operations except `touch` via GRIDFTP and WEBDAV.
3. GSI/certificate-based authentication/authorization (SSH and GRIDFTP).
4. Automatic one-hop handling of third-party transfers over mixed protocols (e.g., SCP on host A to GRIDFTP on host B).
5. Two ways of achieving file transfer or deletion:
    - By specifying exact locations/paths;
    - By scanning or listing.
6. Fully recursive pattern-based scanning (using the '*' and '**' wildcard characters; see UriPattern).
7. All operations configured (using the available settings appropriate to the given protocol) via a general-purpose configuration object.
8. All GRIDFTP options available in the `jglobus` library exposed for configurability; in particular, optimization settings such as:
    - TCP buffer size;
    - setting active mode on the target.
9. Automated support for both LIST and MLST/MLSD (GRIDFTP); options for forcing existence checking through the LIST command.
10. Automated staging of files from UNITREE tape archive using GRIDFTP (= MSSFTP).
11. Full access (i.e., by non-Trebuchet-related code), if so desired, to source and target paths during and after operations.
12. Thread-pooled parallel copy operations.
13. Automated use of multiple GRIDFTP connections for a given endpoint, when available, for non-striped operations.
14. Fail-over and retry capabilities on a file-by-file basis.
15. Flexibility in the kind and number of events the user can choose to receive.

Design overview

The following is a high-level simplification of the layers constituting the Trebuchet core library:



The main component is the operation, which must be configured and run. An operation has in turn a monitor, which mediates between the two agents involved in managing the execution: on the one hand, there is an agent responsible for interacting with the cache (see further below); on the other, an agent which controls the submission of any given piece of work to a client. In the layer just below the agents are found specialized pieces necessary for scanning or for a copy operation. Every operation requires a particular client type capable of handling it. Finally, at the bottom-most layer lies the data abstraction through which all interactions during the operation flow: the "cache". In certain optimized cases, it is possible to avoid creating a cache for the operation, but in most instances a cache or pair of caches will serve as the data source and implicit log.

Scheme-to-protocol mapping

The basis for Trebuchet's multi-protocol functionality lies in mapping (via the Eclipse-RCP extension-point mechanism) URI schemes to a set of implementations.

As an example, let us consider the `ssh` protocol; in order to support the available operations (in this case, all of them), the following classes needed to be implemented:

Function	Abstract Class	Concrete Class
exists, *is file*, *is dir*	`ncsa.tools.trebuchet.core.clients.VerifyClient`	`ncsa.tools.trebuchet.ssh.clients.SSHVerifyClient`
ls	`ncsa.tools.trebuchet.core.clients.ListClient`	`ncsa.tools.trebuchet.ssh.clients.SSHListClient`
touch	`ncsa.tools.trebuchet.core.clients.TouchClient`	`ncsa.tools.trebuchet.ssh.clients.SSHTouchClient`
mkdir	`ncsa.tools.trebuchet.core.clients.MkdirClient`	`ncsa.tools.trebuchet.ssh.clients.SSHMkDirClient`
rm	`ncsa.tools.trebuchet.core.clients.DeleteClient`	`ncsa.tools.trebuchet.ssh.clients.SSHDeleteClient`
cp, *mv*	`ncsa.tools.trebuchet.core.clients.CopyClient`	`ncsa.tools.trebuchet.ssh.clients.SSHCopyClient`

Then a scheme-to-client mapping needed to be provided via extensions to the `ncsa.tools.trebuchet.core.clientTypes` extension point:

Operation	Source Scheme	Target Scheme	Client
verify		`ssh`, `scp`, `gsissh`, `gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHVerifyClient`
list		`ssh`, `scp`, `gsissh`, `gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHListClient`
touch		`ssh`, `scp`, `gsissh`, `gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHTouchClient`
mkdir		`ssh`, `scp`, `gsissh`, `gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHMkDirClient`
delete		`ssh`, `scp`, `gsissh`, `gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHDeleteClient`
copy	`file`	`ssh`, `scp`, `gsissh`, `gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHCopyClient`
copy	`ssh`, `scp`, `gsissh`, `gsiscp`	`file`	`ncsa.tools.trebuchet.ssh.clients.SSHCopyClient`

copy	`ssh, scp, gsissh, gsiscp`	`ssh, scp, gsissh, gsiscp`	`ncsa.tools.trebuchet.ssh.clients.SSHCopyClient`

This mapping is referred to when Trebuchet processes a URI or URIs for a given operation, so that the URI schemes indicate which client to use for the operation.

There are two other classes, the `PooledClientGenerator` and `ListToCopyConverter` which also need to be mapped for each protocol, but usually the default implementations for these classes will be sufficient; also, depending on the file system, a special parser may be necessary for interpreting directory-listing lines, but in most cases the core parsers will work. Finally, for each scheme associated with the protocol, a small definition class implementing `ncsa.tools.trebuchet.schemes.IScheme` needs to be created; this class defines the underlying protocol used for the operation for Trebuchet's internal use, representing the operations which the protocol can support.

The schemes which have been implemented in the current version of Trebuchet are listed here.

Operation Caches

The bottom-most layer of the Trebuchet architecture consists of a binary file for the operation, accessed using Java's `NIO` library, and abstracted out as a Trebuchet *Cache* object. This is admittedly something of a misnomer, since no entries are actually being cached in memory, and therefore no fixed size is maintained by booting entries from it; but it is cache-like in that it provides an access-point through which all aspects of an operation pass and, under normal conditions (successful termination of the entire operation), is transient, i.e., deleted (the cache can optionally be held on to after the operation; if the operation is incomplete or failed it can be restarted from its cache(s) without having to generate the listings from scratch or redo the successful transfers).

There are two standard caches, one for listing or scanning operations, and one for copy or transfer operations. When a copy operation relies on scanning or listing to provide it with the source locations, there is a conversion procedure (supplied by the `ListToCopyConverter` mentioned above) for creating the copy cache entries from the associated list cache entries. Scanned operations for `touch`, `delete` and `copy` by default do the conversion asynchronously using a listener API: as a list entry is added to the list cache, one listener passes it to the converter to be added to the copy cache, while another listener is responsible for passing off the copy entries to the appropriate client as they become available (multiple clients are pooled and this single listener agent designates work for them as they become free). There is an option to override this behavior such that the entire listing or scanning be done first, but in most cases the parallelized list-convert-copy is to be preferred.

The reasons for making all operations rest on a disk-I/O layer are primarily:

- Greater scalability: large or deeply recursive directory copies, for instance, can be handled without risk of running out of memory;
- Greater reliability: because the cache serves as a full operation log, the failed parts of the operation can be retried simply by pointing to the original cache; moreover, the cache will be there should the JVM in which the operation was running crash.

As stated above, the underlying cache file is written in binary. The following tables describe the byte-structure of its respective entry. As can be seen, these are organized similarly to network packets.

LIST CACHE ENTRY

Fixed length entry "header" = 65 bytes. The subscripted properties are specific to the metadata returned by a given file system. "Length" refers to the number of bytes in a variable-length segment of the entry itself; size refers to file size in bytes.

CONTENTS	TYPE	BYTE POSITION
status	`byte`	0
entry id	`long`	1
previous id	`long`	9
type	`byte`	17
symlinked parent	`byte`	18
mode	`char`	19
links	`int`	21
size	`long`	25
modified	`long`	33
user length	`int`	41
group length	`int`	45
relative dir length	`int`	49
name length	`int`	53
symlink length	`int`	57
n = num properties	`int`	61
property name i length	`int`	65 + 8i
property value i length	`int`	69 + 8i
user	`bytes`	65 + 8n

group	`bytes`	65 + 8n + user length
relative dir	`bytes`	65 + 8n + user length + group length
name	`bytes`	65 + 8n + user length + group length + relative dir length
symlink	`bytes`	65 + 8n + user length + group length + relative dir length + name length
property name i	`bytes`	65 + 8n + user length + group length + relative dir length + name length + symlink length + Sum[0 <= j < i] property name j length
property value i	`bytes`	65 + 8n + user length + group length + relative dir length + name length + Sum[0 <= j < n] property name j length + Sum[0 <= j < i] property value j length
(end)		65 + 8n + user length + group length + relative dir length + name length + Sum[0 <= j < n] property name j length + Sum[0 <= j < n] property value j length

COPY CACHE ENTRY

Fixed length entry "header" = 90 bytes. As before, "length" refers to the number of bytes in a variable-length segment of the entry itself; size refers to file size in bytes.

CONTENTS	TYPE	BYTE POSITION
entry id	`long`	0
previous id	`long`	8
status	`byte`	16
type	`byte`	17
first update	`long`	18
last update	`long`	26
retry count	`int`	34
duplicate target tag	`int`	38
source size	`long`	42
tmp size	`long`	50
target size	`long`	58
source modified	`long`	66
source length	`int`	74
symlink length	`int`	78
tmp length	`int`	82
target length	`int`	86
source	`bytes`	90
symlink	`bytes`	90 + source length
tmp target	`bytes`	90 + source length + symlink length
target	`bytes`	90 + source length + symlink length + tmp length
(end)		90 + source length + symlink length + tmp length + target length

The contents of a cache can be printed in human-readable form either by calling the `print` method on the cache object (programmatically), or by using the `TrebuchetCacheReader` from the command-line and pointing it at the cache file. Use this tool to dump the cache either to stdout or to a text file. The commandline options are:

-l <list-cache path>
-c <copy-cache path>
-v (verbose, i.e, display full cache-line string rather than the abbreviated one)
-f write the output to this file

There is also an Ogrescript task `<print-cache>`.

Operation and Cache Listeners

The following APIs are defined in Trebuchet core for listeners to add to an operation in order to receive monitoring events:

```
/**
 * Used by both the operation and its cache to notify changes in status.
 * A "cache update" means an entry has been added to the cache.
 */
public interface IOperationListener
{
   void operationStarted( String operationId );
   void operationFinished( String operationId );
   void cacheUpdated( String operationId, long lastEntryId );
   void scanningDirectory( String operationId, long id );
}

/**
 * Cache entry updates are called whenever a fixed-length field on a given
 * entry line is altered (variable-length fields are write-once).
 */
public interface IUpdateOperationListener extends IOperationListener
{
   void cacheEntryUpdated( String operationId, long entryId );
}
```

Custom listeners of course can be built and added, but there are a number of types already available in the `ncsa.tools.trebuchet.core` and `ncsa.tools.trebuchet.listeners` plug-ins. Further details on these listeners appear in the Ogrescript user's guide: see IOperationListener.

Here are some sample outputs from two of the more common listeners:

1) The **`CopyProgressListener`** tracks the progress of the entire copy operation over time:

```
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 1, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.609, BYTES COPIED 30.0,
THROUGHPUT 4.69790303648399E-5
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 2, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.686, BYTES COPIED 60.0,
THROUGHPUT 8.341174779063411E-5
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 3, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.901, BYTES COPIED 90.0,
THROUGHPUT 9.526158543458657E-5
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 4, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 1.155, BYTES COPIED 120.0,
THROUGHPUT 9.908304586038961E-5
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 5, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 1.302, BYTES COPIED 150.0,
THROUGHPUT 1.098703129500288E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 6, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 1.466, BYTES COPIED 180.0,
THROUGHPUT 1.1709507295574693E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 7, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 1.848, BYTES COPIED 210.0,
THROUGHPUT 1.0837208140980113E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 8, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 1.971, BYTES COPIED 240.0,
THROUGHPUT 1.1612472650304414E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 9, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 2.135, BYTES COPIED 270.0,
THROUGHPUT 1.2060518287104802E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS COPY_STARTED, TOTAL
FILES 10, FILES COMPLETED 10, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 2.26, BYTES COPIED 300.0,
THROUGHPUT 1.2659393580613938E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target, TO /Users/arossi/elf-archive/elf-test/run/report, STATUS DONE, TOTAL FILES 10, FILES
COMPLETED 10, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 2.401, BYTES COPIED 300.0, THROUGHPUT
1.1915963970090588E-4
```

2) The **`FileProgressListener`** tracks the progress of individual files:

```
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_0.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_0.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_0.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_0.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.056, BYTES COPIED 30.0,
THROUGHPUT 5.108969552176339E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_1.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_1.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_0.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_0.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.172, BYTES COPIED
30.0, THROUGHPUT 1.6633854355922967E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_1.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_1.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.141, BYTES COPIED 30.0,
THROUGHPUT 2.029094290226064E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_1.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_1.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.205, BYTES COPIED
30.0, THROUGHPUT 1.3956209508384147E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_2.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_2.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_2.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_2.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:14, ELAPSED TIME 0.052, BYTES COPIED 30.0,
THROUGHPUT 5.501967210036058E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_2.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_2.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:14, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.156, BYTES COPIED
30.0, THROUGHPUT 1.8339890700120194E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_3.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_3.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_3.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_3.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.058, BYTES COPIED 30.0,
THROUGHPUT 4.932798188308189E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_4.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_4.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_3.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_3.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.132, BYTES COPIED
30.0, THROUGHPUT 2.1674416281960225E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_4.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_4.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.105, BYTES COPIED 30.0,
THROUGHPUT 2.7247837611607144E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_4.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_4.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.211, BYTES COPIED
30.0, THROUGHPUT 1.3559350470231042E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_5.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_5.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_5.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_5.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.076, BYTES COPIED 30.0,
THROUGHPUT 3.764503880550987E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_6.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_6.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_5.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_5.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.151, BYTES COPIED
30.0, THROUGHPUT 1.8947171849130796E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_6.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_6.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.055, BYTES COPIED 30.0,
THROUGHPUT 5.201859907670455E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_7.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_7.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_6.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_6.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.147, BYTES COPIED
30.0, THROUGHPUT 1.946274115114796E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_7.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_7.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:15, ELAPSED TIME 0.091, BYTES COPIED 30.0,
THROUGHPUT 3.1439812628777475E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_7.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_7.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:15, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.364, BYTES COPIED
30.0, THROUGHPUT 7.859953157194369E-5
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_8.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_8.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:16, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_8.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_8.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:16, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.12, BYTES COPIED 30.0,
THROUGHPUT 2.384185791015625E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_8.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_8.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:16, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.158, BYTES COPIED
30.0, THROUGHPUT 1.8107740184928798E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_9.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_9.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:16, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.0, BYTES COPIED 0.0,
THROUGHPUT ?
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_9.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_9.txt,
STATUS COPY_STARTED, SIZE 30, START 2006/11/20 13:18:16, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.069, BYTES COPIED 30.0,
THROUGHPUT 4.1464100713315213E-4
OP ID null, FROM /Users/arossi/elf-archive/elf-test/run/target/copies/test_file_9.txt, TO /Users/arossi/elf-archive/elf-test/run/report/copies/test_file_9.txt,
STATUS COPY_COMPLETED, SIZE 30, START 2006/11/20 13:18:16, UPDATED 2006/11/20 13:18:16, ELAPSED TIME 0.111, BYTES COPIED
30.0, THROUGHPUT 2.577498152449324E-4
```

Utility Classes

The following sections are primarily of interest to those wishing to develop code using the Trebuchet library.

ncsa.tools.trebuchet.core.Trebuchet

Provides convenience wrappers around the standard operations. All methods block and do not allow for monitoring or post-processing.

More fine-grained control over the operations can be obtained by using the methods in the `OperationFactory`; see below.

```
static ListCache listRecursive(UriPattern);
static ListCache listRecursive(URI);
static ListCache list(UriPattern, TrebuchetConfiguration, boolean);
static ListCache list(UriPattern);
static ListCache list(URI, TrebuchetConfiguration, boolean);
static ListCache list(URI);

static void touch(UriPattern, TrebuchetConfiguration, boolean);
static void touch(UriPattern);
static void touch(URI[], TrebuchetConfiguration, boolean);
static void touch(URI[]);
static void touch(URI, TrebuchetConfiguration, boolean);
static void touch(URI);

static void mkdir(URI[], TrebuchetConfiguration, boolean);
static void mkdir(URI[]);
static void mkdir(URI, TrebuchetConfiguration, boolean);
static void mkdir(URI);

static void deleteDir(URI, TrebuchetConfiguration, boolean);
static void deleteDir(URI);
static void delete(UriPattern, TrebuchetConfiguration, boolean);
static void delete(UriPattern);
static void delete(URI[], TrebuchetConfiguration, boolean);
static void delete(URI[]);
static void delete(URI, TrebuchetConfiguration, boolean);
static void delete(URI);

static void copyDir(URI, URI, TrebuchetConfiguration, boolean);
static void copyDir(URI, URI);
static void copy(UriPattern, URI, TrebuchetConfiguration, boolean);
static void copy(UriPattern, URI);
static void copy(URI[], URI[], TrebuchetConfiguration, boolean);
static void copy(URI[], URI[]);
static void copy(URI[], URI, TrebuchetConfiguration, boolean);
static void copy(URI[], URI);
static void copy(URI, URI, TrebuchetConfiguration, boolean);
static void copy(URI, URI);

static void moveDir(URI, URI, TrebuchetConfiguration, boolean);
static void moveDir(URI, URI);
static void move(UriPattern, URI, TrebuchetConfiguration, boolean);
static void move(UriPattern, URI);
static void move(URI[], URI[], TrebuchetConfiguration, boolean);
static void move(URI[], URI[]);
static void move(URI[], URI, TrebuchetConfiguration, boolean);
static void move(URI[], URI);
static void move(URI, URI, TrebuchetConfiguration, boolean);
static void move(URI, URI);

static void maybeTransfer(String, File, boolean, TrebuchetConfiguration);
static void maybeTransfer(String, File, boolean);

static boolean supports(URI, String);                 // tells whether the scheme of the URI supports the
operation
static boolean supportsExecutedOverride(URI, String); // tells whether the scheme of the URI supports an
optimized procedure for the single-target operation
                                                      // (touch, mkdir, delete) that bypasses the cache-
mechanism (i.e., a lightweight operation)
```

---

ncsa.tools.trebuchet.core.util.OperationFactory

The factory returns an object which encapsulates the actual Trebuchet operation, allowing for the programmatic control over how and when this may run (`ncsa.tools.trebuchet.core.operations.Operation` implements `java.lang.Runnable` and `ncsa.tools.common.Stoppable`). An operation can take various listeners and also return an instance of its underlying caches.

The methods in this class do a minimal setup on the operation. Typically, the steps subsequent to invoking them will be:

1. Create a `TrebuchetConfiguration` object with the appropriate settings (each operation provides a `getDefaultConfiguration()` method). If a `GSSCredential` is required and is held in memory, it can be set directly on the configuration object (otherwise, the usual procedure for locating the `GSSCredential` on-disk home will be followed).
2. Initialize the operation using the configuration. If the operation is scanned (`Ls...`), the configuration will be automatically persisted. Otherwise, if you wish to store the configuration, call `operation.storeConfiguration( configuration )`. (If you try to store the config after the operation has begun – i.e., after the cache has been written to --, you will get an exception.)
3. Add listeners to the operation. If these listeners themselves require a reference to the cache, they should implement `ncsa.tools.trebuchet.core.ICacheAccessor` (the cache will automatically set a reference to itself on them in this case).
4. Either call `run()` or `new Thread( operation ).start()`.
5. A reference to the underlying cache can be obtained by calling `operation.getReadOnlyCacheInstance()`.
6. Operations can be interrupted by calling `halt()`.
7. All operations except `Ls` allow one to check for fatal exceptions thrown during runtime by calling `getOperationError()`.
8. Calling `cleanup()` will delete the underlying cache(s).

NOTES

- Operations have not been defined as extension points (only clients, entry converters and client generators are). Given the specificity of an operation, generic code support beyond simple initialization methods is unfeasible. As mentioned above, adding a new scheme will involve implementing the relevant clients for the supported operations, and may (but usually will not) require custom converters and generators for the pool; the `Ls` client may also require a custom parser, but this is also an implementation issue which has not been exposed as an extension point.
- When unprocessed (unscanned) URIs are used to construct the operation, the `Touch`, `Mkdir` and `Delete` creation methods check to see if the client for the given scheme supports executed overrides.
- An instance of the factory must be constructed first; this can be done with or without a TrebuchetConfiguration object.

The following object methods are available:

```
Ls createUninitializedListOperation(UriPattern);
Ls createUninitializedListOperation(URI);

Touch createUninitializedTouchOperation(URI[], String);                // operation id
Touch createUninitializedTouchOperation(URI[], boolean, String);       // include ancestors, operation id
Touch createUninitializedTouchOperation(OperationCache);
Touch createUninitializedTouchOperationFromListCache(String);
Touch createUninitializedTouchOperationFromCopyCache(String);

LsTouch createUninitializedTouchOperation(UriPattern);
LsTouch createUninitializedTouchOperationFromPartial(String);

Mkdir createUninitializedMkdirOperation(URI[], boolean, String);       // include ancestors, operation id
Mkdir createUninitializedMkdirOperation(OperationCache);
Mkdir createUninitializedMkdirOperationFromListCache(String);
Mkdir createUninitializedMkdirOperationFromCopyCache(String);

Delete createUninitializedDeleteOperation(URI[], String);              // operation id
Delete createUninitializedDeleteOperation(URI[], boolean, String);     // include ancestors, operation id
Delete createUninitializedDeleteOperation(OperationCache);
Delete createUninitializedDeleteOperationFromListCache(String);
Delete createUninitializedDeleteOperationFromCopyCache(String);

LsDelete createUninitializedDeleteDirOperation(URI);
LsDelete createUninitializedDeleteOperation(UriPattern);
LsDelete createUninitializedDeleteOperationFromPartial(String);

Copy createUninitializedCopyOperations(URI[], URI);
Copy createUninitializedCopyOperation(URI[], URI, String);             // operation id
Copy createUninitializedCopyOperations(URI[], URI[]);
Copy createUninitializedCopyOperation(URI[], URI[], String);           // operation id
Copy createUninitializedCopyOperation(URI[], URI[], boolean, String);  // include ancestors, operation id
Copy createUninitializedCopyOperation(String);
Copy createUninitializedCopyOperation(CopyCache);

LsCopy createUninitializedCopyOperation(UriPattern, URI);
LsCopy createUninitializedCopyDirOperation(URI, URI);
LsCopy createUninitializedCopyOperation(ListCache, URI);
LsCopy createUninitializedCopyOperationFromListCache(String, URI);
LsCopy createUninitializedCopyOperation(String, String);               // list cache path, copy cache path
```

---

ncsa.tools.trebuchet.core.TrebuchetRestart

Provides both a programmatic and command-line way for restarting an operation which has been interrupted or which has not entirely completed successfully.

The command-line options are:

```
-l <list-cache path>
-c <copy-cache path>
-o <operation (touch, mkdir, delete, scanned-touch, scanned-delete, copy, scanned-copy)>
-v (verbose)
-r <number of retries>
```

Point the operation at the cache or cache-pair with which the original operation was associated.