fbtest Documentation

Release 1.0.7.2

Pavel Císař

May 14, 2019

Contents

fbtest is set of tools used for Firebird QA.

Contents:

CHAPTER 1

Introduction

1.1 Requirements

- Python version 2.7. If you have Python 3.x already installed, you may try it as FBTest should work with it, but it was not tested with Python 3 yet.
  - *Linux:* If you don't have Python already installed you can get it from your distribution's repository.
  - *Windows & MacOS:* You may download Python installation package from python.org, or from ActiveState (recommended).
- Distribute module.
- PIP installer. It's not strictly necessary, but it makes you life with Python much easier.
- FDB Firebird driver for Python.
- PySVN module (optional but recommended).

Tip: On Linux you may find Subversion, PySVN, PIP, Distribute and FDB in your distribution repositories.

1.2 Installation

1. Install the *Python* programming language.
2. Install *Distribute*. On Windows and MacOS (or Linux if its not in your repository) download distribute_setup.py and run next command:

   ```
   python distribute_setup.py
   ```

   You can find detailed installation instructions for Distribute here.

3. Install *pip*. On Windows and MacOS (or Linux if its not in your repository) download get-pip.py and run next command:

```
python get-pip.py
```

   You can find detailed installation instructions for PIP here.

4. Install *FDB*. If you have installed *PIP*, you can simply run next command:

```
pip install fdb
```

   to install FDB from PyPI (Python Package Index). Otherwise you have to download FDB, unpack it and run:

```
python setup.py install
```

   from directory where you unpacked it.

5. Install *PySVN* module.

   On Linux you should find it in your distribution repository (as *python-svn* or *pysvn*). On Windows and MacOS you need to download and install appropriate installation kit for PySVN.

   Although fbtest uses Subversion to access Firebird project's repository, you shouldn't need to install it, as it's part of pysvn installation kit for Windows/MacOS and should be installed automatically on Linux (as dependency to pysvn).

6. Install *fbtest*.

   a) If you want to run Firebird test suite, but do not develop new tests, use next method.

      If you have installed *PIP*, you can simply run next command:

```
pip install fbtest
```

      to install it from PyPI. Otherwise you have to download it from Firebird website, unpack it and run:

```
python setup.py install
```

   b) If you want to run tests and also create new ones (or develop fbtest itself), make a checkout of this path from Firebird Subversion repository, and then run:

```
python setup.py develop
```

      You should also download fbtedit - GUI Test editor for Windows.

1.3 Test Repository Initialization

Create directory where you want it stored and run next command:

```
fbt_update repository
```

It will fetch tests and all other necessary files directly from Firebird project subversion repository.

Important: If you don't have PySVN module installed, you have to checkout *Test Repository* manually.

CHAPTER 2

Usage Guide

2.1 Test Repository

Test Reposotory contains *Test definitions*, *Resource definitions*, *Databases*, *Database backups* and *Other files*. Repository is stored in our Subversion repository at SourceForge. You will need a local copy of this repository to run any test. To do that, you can:

a) Create directory where you want it stored, open a command prompt, change to this directory and run next command:

```
fbt_update repository
```

b) Use Subversion client to make a checkout from trunk at central repository into directory where you want it located.

Important: All *fbtest* command-line tools that work with Test Repository must be run from directory where repository is located.

There are several subdirectories in Test repository:

- `resources` : Some tests use special *resources*, for example Firebird user definitions. This directory contains definitions for proper initialization and finalizations of these resources.
- `fdb` : Contains special pre-made databases that some tests may require.
- `fbk` : Contains backup files that some tests may require.
- `files` : Contains other external files (SQL scripts for example) that some tests may require.
- `tests` : Contains all test definitions structured into suite subdirectories.
- `tmp` : Location for temporary databases. If it does not exists when **`fbt_run`** or **`fbt_server`** is executed, it's automatically created with full access rights for everyone.

Important: Tested Firebird server must be able to access databases in `fdb` directory, otherwise all tests that depend on them will fail.

2.2 Test Environment

Test Environment consists from *Test Repository*, Firebird client library and Firebird command-line tools. You don't need any additional configuration if you want to run tests against current Firebird installation. However, if you want to test another Firebird installation (if you have multiple Firebird installations), you have to make sure that *fbtest* will use Firebird client library and command-line tools **from tested Firebird installation**. Scripts that work with Firebird (*fbt_run* and *fbt_server*) have a command-line switch to specificy a directory where Firebird command-line tools are located, and a swicth to specify Firebird client library to be used.

2.3 Running tests

You can run tests against local or remote Firebird installation. However, when you want to run tests against remote Firebird server, you still need locally installed (or accessible) Firebird client library and command-line tools that match the tested server.

2.3.1 Tests and test suites

Each test is designed to test only specific Firebird feature or bug fix. Tests are grouped into logical groups called *suites*, and these suites could be nested. Each test and suite has a *name*. To identify test or suite, you have to use fully qualified name that consists from all parent suite names plus test or suite name in a row, separated by dot. For example, fully qualified name for test named "isql_01" in suite "isql" that's part of suite "basic" which is part of "functional" suite is "functional.basic.isql.isql_01".

Note: Definition of each test is stored in Test Repository as single text file with ".fbt" extension. Each suite is represented as directory and directory tree represents the suite nesting structure.

Important: Current implementation doesn't allow free test file relocation between directories (suites) without adjustments in each moved test definition.

Single *test run* may run all tests in Test Repository or single test/suite. Running test suite means that all test and sub-suites in it are executed.

Note: All tests are designed to work with specific version(s) of Firebird server. Each test contains one or more "recipes" - how to execute and evaluate test when run in specific conditions (platform and/or Firebird version). If test doesn't contain recipe for actual conditions, it's not executed, which is not considered as bug or problem because it means that test was simply not designed to work in these conditions.

Tests are run using **`fdb_run`** script.

2.3.2 Using fbt_run

Usage:

```
fbt_run [-h] [-b BIN_DIR] [-d DB_DIR] [--archive] [--rerun] [-v]
        [--verbosity {0,1,2}] [-q] [-x] [--remote] [-u] [-w PASSWORD]
        [-o HOST] [-p PERSON] [-a ARCH] [-s SEQUENCE] [-k SKIP] [-c CLIENT]
        [name]

positional arguments:
  name                  Suite or test name

optional arguments:
  -h, --help            show this help message and exit
  -b BIN_DIR, --bin-dir BIN_DIR
                        Directory where Firebird binaries tools are
  -d DB_DIR, --db-dir DB_DIR
                        Directory to use for test databases
  --archive             Save last run results to archive
  --rerun               Run only tests that don't PASSed in last run
  --untested            Run only tests that were UNTESTED in last run
  -v, --verbose         Be more verbose
  --verbosity {0,1,2}   Set verbosity; --verbosity=2 is the same as -v
  -q, --quiet           Be less verbose
  -x, --xunit           Provides test results also in the standard XUnit XML
                        format
  -e FILENAME, --expect FILENAME
                        Test results file to be used as expeted outcomes
  --remote              Connect to remote fbtest server
  -u, --update          Update last run results with re-run results
  -w PASSWORD, --password PASSWORD
                        SYSDBA password
  -o HOST, --host HOST  Remote Firebird or fbtest host machine identification
  -p PERSON, --person PERSON
                        QA person name
  -a ARCH, --arch ARCH  Firebird architecture: SS, CS, SC
  -s SEQUENCE, --sequence SEQUENCE
                        Run sequence number for this target
  -k SKIP, --skip SKIP  Suite or test name or name of file with suite/test
                        names to skip
  -c CLIENT, --client CLIENT
                        Use specified Firebird client library
```

This tool runs all or specified set of tests and collects run result from each test. This result for whole run is saved to `results.trf` file in Test Repository for later reference.

During execution *fbt_run* gives feedback to standard output about progress in usual way for unit test programs, including summary report.

In normal verbosity mode *fbt_run* prints a dot for each passed test, or letter indicating detected problem: 'F' for FAIL, 'E' for ERROR and 'U' for UNTESTED.

Examples:

```
>fbt_run functional.basic.isql
...
----------------------------------------------------------------------
Ran 3 tests in 0.918s
```

(continues on next page)

(continued from previous page)

```
OK
```

```
>fbt_run functional.basic.isql
.F.
======================================================================
FAIL: functional.basic.isql.isql_01
----------------------------------------------------------------------
Expected standard output from ISQL does not match actual output.

----------------------------------------------------------------------
Ran 3 tests in 0.949s

FAILED (fails=1)
```

You may increase or decrease the amount of information printed using *–verbose*, *–quiet* and *–verbosity* options.

Example output for **verbose** mode:

```
>fbt_run -v functional.basic.isql
functional.basic.isql.isql_03 ... ok
functional.basic.isql.isql_01 ... ok
functional.basic.isql.isql_02 ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.939s

OK
```

```
>fbt_run -v functional.basic.isql
functional.basic.isql.isql_03 ... ok
functional.basic.isql.isql_01 ... FAIL
functional.basic.isql.isql_02 ... ok
======================================================================
FAIL: functional.basic.isql.isql_01
----------------------------------------------------------------------
Expected standard output from ISQL does not match actual output.

----------------------------------------------------------------------
Ran 3 tests in 0.922s

FAILED (fails=1)
```

Example output for **quiet** mode:

```
>fbt_run -q functional.basic.isql
----------------------------------------------------------------------
Ran 3 tests in 0.925s

OK
```

```
>fbt_run -q functional.basic.isql
======================================================================
FAIL: functional.basic.isql.isql_01
----------------------------------------------------------------------
Expected standard output from ISQL does not match actual output.
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------
Ran 3 tests in 0.933s

FAILED (fails=1)
```

Tip: You may get more detailed information about run results using **fbt_view** and **fbt_analyze**.

There is no need to use any additional command-line options for quick execution of all or selected test(s) against current Firebird installation. However, you would need to specify some additional options in other cases:

- When SYSDBA password for tested server differs from default 'masterkey', you have to use `--password` option.
- When tested server runs on different machine, you have to use `--host`, `--bin-dir` and `--db-dir` options.
- When tested server runs on local machine but on different port than default one, you have to use `--host` option.
- Temporary databases used by tests are created in `tmp` subdirectory in Test Repository. If you want temporary databases in different location, you will need `--db-dir` option.
- If you want to compare test run results from several server architectures, you should specify server architecture of tested engine using `--arch` option.
- If you want that test run results would be also archived, you have to specify `--archive` option. You should also consider using `--arch` and `--person` options in this case.
- If you want to exclude some tests from execution, you will need `--skip` option. However these tests are included into results file with outcome *SKIPPED* which is special kind of *UNTESTED* outcome.
- If you know that some tests will fail, you can either skip them altogether using `--skip` option, or you can run them but set an expectation using `--expect` option and a result file from previous run. Test will then PASS if test outcome and its cause will match expected one, otherwise it will FAIL. Please note that run details of failure (like content of standard output or error output) are NOT compared, only general description of the cause is checked. So test will fail only if cause of failure significantly changes it's type (for example from difference in standard output to difference in error output).
- If you want to run the same set of tests several times and compare their results using *fbt_analyze*, you have to specify `--sequence` option. Don't forget to copy the results.trf file to safe location after each series run, or use `--archive`.
- If you want to run only tests that didn't passed the last run, use `--rerun` option, and if you want the last run results updated with results from new run, use `--update` option.
- Since version 1.0.4 *fbt_run* checks that Firebird engine is running before each test is executed by creating a connection to Firebird services. If this attempt fails, test is not executed at all, and its outcome is set to *UNTESTED*. When you fix the problem with Firebird engine, you may re-run all these tests using `--untested` option that works similarly to `--rerun` option.
- If you want to send run results to someone, you should specify `--arch` and `--person` options.
- If you need run results also in standard XUnit XML format, use `--xunit` option.

Important: If your test environment is not properly configured, many (if not all) tests would fail or raise errors, which would spoil the test run results. For example if Firebird engine wouldn't have sufficient rights to create/access databases in location for temporary databases, almost every test would fail as most of them use temporary databases.

Tip: Test Repository contains test named *check* that you could run to verify that your test environment is correctly configured before you'll run the whole test series.

2.4 Working with remote test server

Sometimes you may need to run tests on remote Firebird server, for example to test Firebird on different platform than is your primary platform. While you may use local *fbtest* installation to run against remote Firebird, it could be better (and easier to configure) to install *fbtest* also on remote machine and operate it from your workstation almost like it would be all installed locally.

Before your can connect to remote *fbtest*, you have to run it in "server" mode. To do that, run **`fbt_server`** on remote machine.

fbt_server accepts command-line options `--bin-dir`, `--db-dir`, `--password`, `--host`, `--arch` and `--person` that have the same function like *fbt_run* options with the same name.

Normally fbtest server listens on port 18861 and clients must know on which host it runs to contact him. Alternatively fbtest server could anounce itself on network via *remote service registry*. To use this mode you must start it with `--register` option.

Once remote fbtest server is up and running, you may use *fbt_run* to use it as test execution engine, i.e. all tests are executed by remote server on server host, but all output is produced on client side (console output and *results.trf* file).

To use remote fbtest, execute *fbt_run* with `--remote` option. If fbtest server is NOT started with `--register`, you must also specify host machine using `--host` option.

When remote fbtest engine is used, `--bin-dir`, `--db-dir` and `--password` options are ignored when specified.

Note: Note that `--host` option has different meaning when used together with *–remote*.

Warning: Do NOT operate fbtest server on open network! Current implementation gives full control to clients over it, which is potential security risk.

Remote fbtest server is also used by *Test editor* to execute tests on other platforms than Windows.

2.5 Test run result analysis

When test execution doesn't end with success, you need to investigate why did that happen, because *fbt_run* gives only basic information: test run *outcome* and *cause* of failure if test didn't passed. However, test run result information (stored in `results.trf`) contains all details including analytical information. You may inspect these information using *fbt_view* tool, or generate detailed HTML report using *fbt_analyze* tool, which can also compare results from multiple runs.

2.5.1 Test run outcome

Test run may end in four different ways:

PASS Everything went just fine.

FAIL Test executed correctly, but actual outcome does not match expected one.

ERROR An error (exception) occured during test execution.

UNTESTED Test couldn't be executed because some condition wasn't met (typically setup of required resource failed).

SKIPPED Test execution was suppressed using –skip switch.

2.5.2 Failure cause

Failure (or error) cause reported by *fbtest* explains in short why *fbtest* decided about test run outcome.

Example causes:

```
Expected standard output from ISQL does not match actual output.
```

```
Test setup: Exception raised while creating database.
```

Reported cause isn't automatically the real reason (problem source).

FAILure could signal a real problem (functionality was broken) or could be a "false positive" (change was intentional), and requires further analysis to determine which case it is. The quickest way is to examine difference between expectet test output and real output using *fbt_view* tool.

ERROR is typically an outcome of bad setup of your test environment, but sometimes it could also signal a real problem (functionality was broken). The quickest way to see all details about error is using *fbt_view* tool.

2.5.3 Using fbt_view

This tool displays information from run result (*.trf*) files. It can also create XUnit XML run result reports.

Usage:

```
fbt_view [-h] [-x] [-c] [-d] [name]

positional arguments:
  name           Results file or directory with result files

optional arguments:
  -h, --help     show this help message and exit
  -x, --xunit    Save test results in the standard XUnit XML format
  -c, --cause    Print cause of fails and errors.
  -d, --details  Print details for fails and errors.
```

Note: *fbt_view* works with run result files only and thus coud be run from any directory.

Note: Without parameters or options *fbt_view* shows summary information for all run results files in working directory.

Example output:

```
>fbt_view

File:     results.trf
Desc:     Linux64 SS
Version:  2.5.2.26540
Arch:     SS
Platform: Linux
CPU:      64
Sequence: 1
Person:   pcisar (PC)

Passes:   2
Fails:    1
Errors:   0
Untested: 0

=== FAILS ===============================================================
functional.basic.isql.isql_01
```

To see also causes use `--cause` option:

```
>fbt_view --cause

File:     results.trf
Desc:     Linux64 SS
Version:  2.5.2.26540
Arch:     SS
Platform: Linux
CPU:      64
Sequence: 1
Person:   pcisar (PC)

Passes:   2
Fails:    1
Errors:   0
Untested: 0

=== FAILS ===============================================================
functional.basic.isql.isql_01
   Expected standard output from ISQL does not match actual output.
```

To see details why tests didn't passed use `--details` option. For FAIL outcome it shows difference (in standard diff format) between expected and actual outputs:

```
>fbt_view --details

File:     results.trf
Desc:     Linux64 SS
Version:  2.5.2.26540
Arch:     SS
Platform: Linux
CPU:      64
Sequence: 1
Person:   pcisar (PC)

Passes:   2
Fails:    1
```

(continues on next page)

(continued from previous page)

```
Errors:   0
Untested: 0

=== FAILS ===================================================================
functional.basic.isql.isql_01
------------------------------------------------------------------------------
ISQL_stripped_diff:
  Owner: SYSDBA
  PAGE_SIZE 4096
  Number of DB pages allocated = 165
  Sweep interval = 20000
  Forced Writes are ON
- ODS = 11.22
?           -

+ ODS = 11.2
  Default Character set: NONE
```

For ERROR it shows detailed error information:

```
>fbt_view --details

File:     results.trf
Desc:     Linux64 SS
Version:  2.5.2.26540
Arch:     SS
Platform: Linux
CPU:      64
Sequence: 1
Person:   pcisar (PC)

Passes:   0
Fails:    0
Errors:   1
Untested: 0

=== ERRORS ==================================================================
functional.basic.isql.isql_01
------------------------------------------------------------------------------
exception:
ProgrammingError:
Error while creating database:
- SQLCODE: -902
- I/O error during "open O_CREAT" operation for file "/home/job/fbtrepo/tmp/
↪functional.basic.isql.isql_02.fdb"
- Error while trying to create file
- Permission denied
-902
335544344

------------------------------------------------------------------------------
db_unable_to_create:
localhost:/home/job/fbtrepo/tmp/functional.basic.isql.isql_01.fdb
------------------------------------------------------------------------------
traceback:
  File "/home/job/python/envs/pyfirebird/fbtest/fbtest.py", line 635, in run
    conn = kdb.create_database(createCommand, self.sql_dialect)
```

(continues on next page)

(continued from previous page)

```
  File "/home/job/python/envs/pyfirebird/fdb/fdb/fbcore.py", line 704, in create_
↪database
    "Error while creating database:")
```

2.5.4 Using fbt_analyze

This tool analyzes run results file(s) and produces colored HTML report.

Usage:

```
fbt_analyze [-h] [-o OUTPUT] [name]

positional arguments:
  name                  Results file or directory with result files

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Analysis output directory
  -d, --diffs-only      Show only diffs on detail pages
```

Reports consists from summary page (`index.html`) and detail pages for each test that didn't passed.

Example summary page:

QA Analysis for Firebird 2.5.2.26540

Linux												Windows												
32-bit						64-bit						32-bit						64-bit						
cs		sc		ss		cs		sc		ss		cs		sc		ss		cs		sc		ss		
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	
F	F	F	F	P	P	F	F	F	F	P	P	F	F	F	F	P	P	F	F	F	F	P	P	bugs.core_2017
-	-	-	-	-	-	F	F	F	F	F	F	-	-	-	-	-	-	-	-	-	-	F	F	bugs.core_1894
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	functional.arno.optimizer.opt_inner_join_02
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	functional.arno.optimizer.opt_inner_join_03
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	functional.arno.optimizer.opt_inner_join_05
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	functional.arno.optimizer.opt_inner_join_07
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	functional.arno.optimizer.opt_inner_join_09
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	bugs.core_0086
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	bugs.core_0088
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	bugs.core_0091
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	bugs.core_0099
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	bugs.core_0104
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	bugs.core_0116

As you can see, summary is presented as table with row for each test and column for each input results file. Table cells contain test run outcome for each run. Columns are sorted and grouped by platform, CPU, Firebird architecture and test run sequence number.

Note: Since version 1.0.4 this report contains time performance of tests.

Important: If you want to compare results from several test runs, you must specify `--sequence` option to *fbt_run*. Similarly you have to specify `--arch` option to compare results from multiple Firebird architectures. If you forgot

to do so, you can add/change this information to results file later using *fbt_update*.

Tip: You can verify platform, CPU, Firebird architecture and run sequence number values stored in result file using *fbt_view*.

Detail page contains all informations related to test run recorded by *fbtest* from all result files where test doesn't passed. Information is "grouped" by result file so only unique content is included.

Collected information for failed tests contains expected and actual outputs and their difference (in human-readable diff format). If you are interested to see only diffs, use `--diffs-only` option.

Note: *fbt_analyze* works with run result files only and thus coud be run from any directory.

Note: Without parameters or options *fbt_analyze* processes all run results files and produces HTML report in current working directory.

2.6 Using fbt_update

This tools has two purposes:

- Updates local Test Repository from central Subversion repository.
- Updates meta-information in test run results file(s).

Usage:

```
fbt_update [-h] {result,repository} ...

optional arguments:
  -h, --help           show this help message and exit

Commands:
  {result,repository}  Use <command> --help for more information about command.
    result             Change result file metadata.
    repository         Update test repository.
```

```
fbt_update repository [-h]

Update local test repository from Firebird project Subversion repository.

optional arguments:
  -h, --help  show this help message and exit
```

```
fbt_update result [-h] [-a ARCH] [-p PERSON] [-s SEQUENCE] [name]

Changes metadata of result file(s).

positional arguments:
  name                  Results file or directory with result files
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -a ARCH, --arch ARCH  Update result(s): set ARCH
  -p PERSON, --person PERSON
                        Update result(s): set PERSON
  -s SEQUENCE, --sequence SEQUENCE
                        Update result(s): set SEQUENCE NUMBER
```

2.7 Using fbt_archive

fbtest provides simple archive for test run results files. When you specify `--archive` option to **fbt_run**, run results file *results.trf* is also **copied** to archive in *Test Repository* (stored in subdirectory "archive") in subdirectory named as tested Firebird version number and filename that identifies run conditions:

- Number of tests executed
- Platform
- CPU
- Firebird architecture
- QA person
- Run sequence number

For example 3 tests run on 64-bit Linux Firebird v2.5.2.26540 SuperServer by pcisar without sequence number will be stored in *2.5.2.26540/0003-Linux64-SS-PC1.trf*.

Note: If filename already exists in archive, it's replaced with new one.

While you can work with archived results files directly, *fbtest* also provides separate tool **fbt_archive** to *list*, *save*, *retrieve* or *delete* archived results. This is particularly useful when you're working with remote *fbtest* installation.

Usage:

```
fbt_archive [-h] [--remote] [-o HOST] {list,save,retrieve,delete} ...

optional arguments:
  -h, --help            show this help message and exit
  --remote              Connect to remote fbtest server
  -o HOST, --host HOST  Remote fbtest host machine identification

Commands:
  {list,save,retrieve,delete}
                        Use <command> --help for more information about command.
    list                List result(s) in archive.
    save                Save result(s) to archive.
    retrieve            Retrieve result(s) from archive.
    delete              Delete result(s) from archive.
```

2.7.1 List

Usage:

```
fbt_archive list [-h]

List result(s) in archive.

optional arguments:
  -h, --help  show this help message and exit
```

Example:

```
>fbt_archive list
Files in archive:

2.1.5.18497:
   0681-Linux64-CS-PC1.trf
   0681-Linux64-CS-PC2.trf
   0681-Linux64-SS-PC1.trf
   0681-Linux64-SS-PC2.trf
2.5.2.26539:
   0003-Linux64-SS-XX1.trf
   0003-Linux64-SS-XX2.trf
   0824-Linux64-SS-PC1.trf
2.5.2.26540:
   0003-Linux64-SS-XX1.trf
   0823-Linux64-SC-PC2.trf
   0824-Linux64-CS-PC1.trf
   0824-Linux64-CS-PC2.trf
   0824-Linux64-SC-PC1.trf
   0824-Linux64-SS-PC1.trf
   0824-Linux64-SS-PC2.trf
```

2.7.2 Save

Usage:

```
fbt_archive save [-h] [name]

Save result(s) to archive.

positional arguments:
  name        Results file

optional arguments:
  -h, --help  show this help message and exit
```

When filename is not specified, file *results.trf* in current directory is copied.

Example:

```
>fbt_archive save
Results file 'results.trf' stored into archive as '2.5.2.26540/0003-Linux64-SS-XX1.trf
↪'
```

2.7.3 Retrieve

Usage:

```
fbt_archive retrieve [-h] [-v VERSION] [-c] [-o OUTPUT] [-a ARCH] [-p PERSON] [-s␣
↪SEQUENCE]

Retrieve result(s) from archive.

optional arguments:
  -h, --help            show this help message and exit
  -v VERSION, --version VERSION
                        Only specified Firebird version
  -c, --current         Only currently tested Firebird version
  -o OUTPUT, --output OUTPUT
                        Output directory
  -a ARCH, --arch ARCH  Firebird architecture: SS, CS, SC
  -p PERSON, --person PERSON
                        QA person name
  -s SEQUENCE, --sequence SEQUENCE
                        Run sequence number
```

This command copies all archived results files for specified or currently tested Firebird version (either `--version` or `--current` option is required) from archive to specified or current working directory. It's possible to specify additional conditions for Firebird architecture, QA person or run sequence number that must be met.

Examples:

```
>fbt_archive retrieve --current
Current version: 2.5.2.26540
0003-Linux64-SS-XX1.trf retrieved.
0823-Linux64-SC-PC2.trf retrieved.
0824-Linux64-CS-PC1.trf retrieved.
0824-Linux64-CS-PC2.trf retrieved.
0824-Linux64-SC-PC1.trf retrieved.
0824-Linux64-SS-PC1.trf retrieved.
0824-Linux64-SS-PC2.trf retrieved.

>fbt_archive --remote retrieve --version=2.1.5.18497 -a SS
0681-Linux64-SS-PC1.trf retrieved.
0681-Linux64-SS-PC2.trf retrieved.
```

2.7.4 Delete

This command deletes all archived results files for specified or currently tested Firebird version (either `--version` or `--current` option is required) from archive. It's possible to specify additional conditions for Firebird architecture, QA person or run sequence number that must be met.

Usage:

```
fbt_archive delete [-h] [-v VERSION] [-c] [-a ARCH] [-p PERSON] [-s SEQUENCE]

Delete result(s) from archive.

optional arguments:
  -h, --help            show this help message and exit
  -v VERSION, --version VERSION
                        Only specified Firebird version
  -c, --current         Only currently tested Firebird version
  -a ARCH, --arch ARCH  Firebird architecture: SS, CS, SC
```

(continues on next page)

(continued from previous page)

```
  -p PERSON, --person PERSON
                        QA person name
  -s SEQUENCE, --sequence SEQUENCE
                        Run sequence number
```

Examples:

```
>fbt_archive delete --current -a SS -s 2
Current version: 2.5.2.26540
0824-Linux64-SS-PC2.trf deleted.

>fbt_archive --remote delete --version=2.1.5.18497 -a SS
0681-Linux64-SS-PC1.trf deleted.
0681-Linux64-SS-PC2.trf deleted.
```

2.8 Using fbt_db

Beside simple results archive *fbtest* also supports archival of results in Firebird database(s).

Archive database must have next structure:

Table 1: Table RUNS - Information about suite run

Column Name	Type	Description
PK	BIGINT	Primary key (autoincrement)
CREATED	TIMESTAMP	Date and time when result file was imported.
VER	VARCHAR(15)	Firebird version (format: x.x.x)
BUILD	BIGINT	Firebird build number
ARCH	CHAR(2)	Firebird architecture (SS, CS, SC)
PLATFORM	CHAR(1)	Firebird platform code (L=Linux,W=Windows,F=FreeBSD,S=Solaris,H=HP-UX)
CPU	VARCHAR(7)	CPU architecture (32 or 64)
PERSON_ID	CHAR(2)	QA person ID
PERSON	VARCHAR(25)	QA person name
SEQ	INTEGER	Run sequential number
DESCRIPTION	VARCHAR(30)	Run description

Table 2: Table TESTS - Information about tests

Column Name	Type	Description
PK	BIGINT	Primary key (autoincrement)
NAME	VARCHAR(300)	Test ID

Table 3: Table ANN_TYPES - Information about annotation types

Column Name	Type	Description
PK	BIGINT	Primary key (autoincrement)
NAME	VARCHAR(70)	Annotation type name

Table 4: Table OUTCOMES - Information about test run outcomes

Column Name	Type	Description
PK	BIGINT	Primary key (autoincrement)
RUN_ID	BIGINT	FK to RUNS
TEST_ID	BIGINT	FK to TESTS
KIND	CHAR(3)	Outcome type (TST=Test run, Resource setup=R-S, resource cleanup=R-C)
OUTCOME	CHAR(1)	Outcome (P=PASS, F=FAIL, E=ERROR, U=UNTESTED, S=SKIPPED)
RUN_TIME	TIME	Run time

Table 5: Table ANNOTATIONS - Information about outcome annotations

Column Name	Type	Description
PK	BIGINT	Primary key (autoincrement)
ANN_TYPE_ID	BIGINT	FK to ANN_TYPES
OUTCOME_ID	BIGINT	FK to OUTCOMES
ANNOTATION	BLOB sub_type text	Annotation value

To create such archive database you can use the *fbt_db* utility. This utility can also import result files into database.

Usage:

```
fbt_db [-h] [-w PASSWORD] [-o HOST] -d DATABASE {import,create} ...

optional arguments:
  -h, --help            show this help message and exit
  -w PASSWORD, --password PASSWORD
                        SYSDBA password
  -o HOST, --host HOST  Firebird host machine identification
  -d DATABASE, --database DATABASE
                        Archive database name
  -c CLIENT, --client CLIENT
                        Use specified Firebird client library
```

(continues on next page)

(continued from previous page)

```
Commands:
  {import,create}       Use <command> --help for more information about
                        command.
    create              Creates archive database.
    import              Import result(s) to database.
```

2.8.1 Create

This command creates empty archive database. This operation fails if specified database already exists.

Usage:

```
fbt_db [-w PASSWORD] [-o HOST] -d DATABASE create [-h]

Create archive database.

optional arguments:
  -w PASSWORD, --password PASSWORD
                        SYSDBA password
  -o HOST, --host HOST  Firebird host machine identification
  -d DATABASE, --database DATABASE
                        Archive database name
  -c CLIENT, --client CLIENT
                        Use specified Firebird client library
  -h, --help  show this help message and exit
```

2.8.2 Import

This command imports resul file(s) into archive database.

Usage:

```
fbt_db [-w PASSWORD] [-o HOST] -d DATABASE import [-h] [name]

Import result(s) to database.

positional arguments:
  name        Results file or directory with result files

optional arguments:
  -w PASSWORD, --password PASSWORD
                        SYSDBA password
  -o HOST, --host HOST  Firebird host machine identification
  -d DATABASE, --database DATABASE
                        Archive database name
  -c CLIENT, --client CLIENT
                        Use specified Firebird client library
  -h, --help  show this help message and exit
```

CHAPTER 3

How to design new tests

3.1 Where to start

First, it's important to identify what you want to test. To avoid collision with others, take a look at our list of tests, and check if your beloved one is not already created! Then let us know about your intention in *firebird-test* mailing list.

If you want to create functionality tests, then you'll need Firebird SQL Reference Guide. Unfortunately, there isn't any **complete** and freely available Firebird-specific SQL reference documentation right now, but you can use InterBase 6.0 Language Reference Guide together with Language Reference Update documents.

If you want to create regression tests, please refer to Firebird Project Tracker for all bug-related informations. It's also advised to consult Firebird QA team.

3.2 The Golden Rule

Test case should be really simple, and should cover only one aspect of single feature / command in discrete conditions.

Lets take the SELECT statement as an example. SELECT statement is quite complex, so you'll need to break it into clauses and choose one, for example the FIRST/SKIP. Then you need to identify all the features of that statement you want to test.

1. SKIP only
2. FIRST only
3. FIRST and SKIP together

Then you can go to design test cases that would cover these features. Focus on testing all *legal paths* first (positive test) — i.e. does it work correctly as specified? If there are any behaviour-switching value boundaries, concentrate your work around them!

For example positive test cases for FIRST .. SKIP for feature "3. FIRST and SKIP together" could be defined as checking result from "select skip 10 first 5. . . " in next conditions:

1. with no data — *No data* is an important condition for all DML commands

2. with 10 rows — Behaviour-switching value boundary for SKIP
3. with 11 rows — Behaviour-switching value boundary for SKIP and FIRST
4. with 16 rows — Behaviour-switching value boundary for FIRST

When you have these basic test cases, you can specify various work conditions and combine these test cases with them to produce final set of test cases:

1. Data taken from single table without WHERE predicate, i.e. table contains specified number of rows.
2. Data taken from single table, larger resultset narrowed by WHERE predicate to specified number of rows.
3. Data taken from joined tables, where result of this join has required number of rows.
4. Data taken from stored procedure that generates required number of rows.
5. SORTED result from any source of data listed above (there is no need to spawn another dimension in the matrix, as dependancy on source of data is already covered in other groups).

When legal paths are explored and covered, look at important *illegal paths* (negative tests) - does it correctly signal an error when wrong values are submitted? Because negative tests are endless, focus only on most important / expected points of failure. For example:

1. Negative value for SKIP
2. Negative value for FIRST

You should also define test cases for special "bizarre" values that are legal (so they do not raise an error), but are not "right" in common sense. They are used rarely, so they are often overlooked by test designers, but as they are typically behaviour-switching boundary values, their verification is very important. In case of FIRST and SKIP, this "bizarre" parameter value is zero.

Each test case has its own requirements for running environment: database schema and content, tools etc. These requirements must be a part of test case specification.

All tests have common basic structure:

1. **Requirements** for running environment: database schema and content, tools etc.
2. **Tested command(s)**. If test cases are well defined, then each has one and only one directly tested command. Its outcome is verified by expected output (if any), and / or with additional checks (check for right content in system tables for example).
3. **Expected output** from tested command(s). It could be standard command output or error message. The best way to describe it is as standard ISQL output when command(s) is executed (You can use ISQL OUTPUT command to grab it). But you can define it in any other way you see fit for you and the purpose.
4. **Additional checks**. If the direct output from tested command is not enough to verify its correctness (some commands even don't produce any "visible" output), you must use additional means (check the content in system tables, check presence of file on disk etc.)

3.3 Making test cases into tests

In ideal world, each test case would be implemented as single test. This setup would provide most value for QA team, as test failure could be easily analyzed, and broken part of the engine (or in test itself) could be tracked down more precisely. Unfortunately, test implementation could require a lot of work, because each test needs its own running environment created independently from other tests. So if several test cases are closely related and use the same working environment, it could be more practical to give up on fine-grained evidence in test outcome in favour of simplified implementation, and merge them into single test.

In case of "FIRST 5 SKIP 10" we crumbled before into approx. 20 test cases we can implement some groups of test cases that use the same database and source of data in single test. For example group of test cases that take data from single table, with larger resultset narrowed by WHERE predicate to none, 10, 11 and 16 rows can easily use the same setup (database, table and table content), so we can create it as single test.

When you decide to wrap up several test cases into single test, keep clear what are individual test cases, i.e. don't try to make any "shortcuts" or "optimizations" in them. They should share only the common environment, nothing more. It should be also clearly stated and documented, that this paricular test contains multiple tests cases, and which they are.

3.4 From drawing board to production

Once the test design is finished, it's time to implement it. If you do not want to mess up with *fbtest* and implement it yourself, you can simply write the specification for test and send it over to us.

In this case, the test specification document should contain next information:

Test ID Tests have hierarchical, dot-separated names / ID's, that must be unique in whole Firebird test suite. Take a look at test IDs in *Test Repository* for test ID examples. It would be great if test ID would conform to common schema used by Firebird QA team so it could persist, but don't worry too much about it, as it could be easily adjusted later, and the main purpose for Test ID in specification document is to have a tag that could be used to refer on test in communication between you and the QA team.

Author Your name and e-mail

Description Clear specification what is checked by this test. If test contains more than one test case (see above), then all test cases should be described separately.

Dependencies Your test would very likely depend on other SQL commands, tools or Firebird features beside tested ones, so they must work correctly if the test outcome should not be spoiled. Because these features are checked by other tests, we can simply run tests in dependancy order to get unspoiled results. Of course, we could extract this information from other parts of specification, but separate list of dependencies would make the whole specification more clear and concise, and save us some time we would need to figure it out. You can simply describe these dependencies by words, or you can look up IDs for tests that must be run before this one (but it's not necessary)

Prerequisites Any special conditions, tools or environment required for this test (except the test database and standard tools). Most tests do not have any special requirement beyond single work database and availability of standard Firebird command-line tools, so these requirements are fulfiled automatically. But if your test needs anything else beyond that, you must enlist it here.

Database specification It's very likely that your test works with a database. You can give us a backup file for it (if the schema is complex or database must contain a lot of data), or you can specify how it could be created. By default, each test can get a new dialect 3 database owned by SYSDBA, with character set NONE and with page size 4K, so you don't need to specify these parameters if they are not different.If you would need this database with certain schema and populated with data, provide an ISQL script for it here. You can also refer to a database/script used in another test by test ID

Test command(s) Self-explanatory.

Expected result from tested command (returned data or error code etc.)

Additional checks (if any) - verification from database content (for INSERT statement and the likes). DDL commands are checked against system tables. Check may query more than one table, but it's necessary to list each command and its expected output (captured output from ISQL is enough).

Example:

```
Test ID: domain.alter.02
Author: Slavomir Skopalik (skopalik at hlubocky.del.cz)

Description:
Checks ALTER DOMAIN...DROP DEFAULT for VARCHAR defaults

Dependencies:

CREATE DOMAIN
Simple SELECT

Prerequisites: NONE

Database specification: Standard.
Initialization:
CREATE DOMAIN test VARCHAR(63) DEFAULT 'test string';

Tested command: ALTER DOMAIN test DROP DEFAULT;
Expected result: No stdout or stderr.

Additional checks:
command:
SELECT RDB$FIELD_NAME, RDB$DEFAULT_SOURCE FROM rdb$fields WHERE RDB$FIELD_NAME = 'TEST
↪';
Output:
RDB$FIELD_NAME                  RDB$DEFAULT_SOURCE
=============================== ==================
TEST                                        null
```

If you have any suggestions or criticism please drop us an e-mail in Firebird-test mailing list.

CHAPTER 4

Writting new tests

CHAPTER 5

fbtest Reference

5.1 Globals

fbtest.**script_runner**
    `ScriptRunner` instance.

5.2 Functions

5.3 Classes

5.3.1 TestVersion class

5.3.2 Test class

5.3.3 Resource class

5.3.4 UserResource class

5.3.5 Suite class

5.3.6 Repository class

5.3.7 Archive class

5.3.8 Result class

5.3.9 RunResults class

5.3.10 Runner class

5.3.11 ScriptRunner class

5.4 Script Functions

CHAPTER 6

Changelog

6.1 Version 1.0.7.2

- Added pull request from Artyom Smirnov (better support for Python test debugging).

6.2 Version 1.0.7.1

- Added titles of tests in analysis HTML report.

6.3 Version 1.0.7

- Fixed issues with documentation.
- Added CLI option -c, –client to fbt_db utility.

6.4 Version 1.0.6

- Broken release, deleted.

6.5 Version 1.0.5

- New utility fbt_db for managing Firebird database with test results.

6.6 Version 1.0.4

- (fbt_run) Include SKIPPED tests into results file with spec. outcome
- (fbt_run) Check that Firebird is running before test execution
- (fbt_analyze) Show time performance of tests

6.7 Version 1.0.3

- (fbt_run) Allow use of custom FB client library
- (fbt_run) Return proper errorlevel (0 = all passed, 1 = otherwise)
- Allow specification of repository location. Now you can use environment variable FBT_REPO to specify directory where fbtest Repository is located.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

F

S