The Falcon Programming Language

Command line tools

Giancarlo Niccolai

— Release 0.8.14 —

The Falcon programming language – Command line tools.

Table of contents

Falcon as a toolset....4
The falcon interpreter....5
- SYNOPSIS....5
- DESCRIPTION....5
- OPTIONS....5
- FILES....7
- ENVIRONMENT....7

Falcon disassembler....8
- SYNOPSIS....8
- DESCRIPTION....8
- OPTIONS....8
- FILES....8

Falcon module launcher....9
- SYNOPSIS....9
- DESCRIPTION....9
- OPTIONS....9
- FILES....9
- ENVIRONMENT....9

The Falcon test suite tool....10
- SYNOPSIS....10
- DESCRIPTION....10
- UNIT TEST SCRIPTS....10
- THE TESTSUITE MODULE....11
- OPTIONS....12
- SAMPLE....13
- FILES....14

Falconeer – module configurator....15
- SYNOPSIS....15
- DESCRIPTION....15
- OPTIONS....15
- BUGS....15
- NOTES....16

Fallc – Language table compiler....17
- SYNOPSIS....17
- DESCRIPTION....17
- OPTIONS....17
- NOTES....17

Falcon as a toolset

The Falcon Programming Language is not just an embeddable scripting engine for applications. It is also a standalone scripting language that can be used directly under the hood of an operating system. Falcon comes with a set of command line tools that can be used to run scripts, compile them in binary format, analyze them and a few more things.

This manual illustrates the usage of the Falcon toolset, which is currently composed of:

- **falcon**: the command line compiler and interpreter.
- **flcdisass**: the module disassembler.
- **flcrun**: the standalone module launcher.
- faltest: interface for the unit test engine.
- falconeer.fal: script to configure skeleton module.
- fallc.fal: script to compile translation tables for internationalization.

This manual has been automatically generated from the Falcon manpages; for this reason it follows typographic and naming conventions typical of that format.

The falcon interpreter

SYNOPSIS

falcon [options] [*main_script*] **[script_options]**

falcon [options] -i

DESCRIPTION

The **falcon** command line interpreter is used to launch, compile, assemble or debug scripts written in the Falcon Programming Language. The command line interpreter pushes the **core** module and the **rtl** module in the script load table, so they are available by default to all the scripts.

The default operation is that of launching the given script, or read it from the standard input if the script name is not given. By default, **falcon** saves also the compiled module of the script in the same directory where it is found, changing the extension to ".fam".

The **main_script** can be a "logical" module name, a relative path or an absolute path. In case it's a logical module name, that is, a script name without extension nor leading path, it is searched through the load path. The default load path is determined by the compilation options of the interpreter, and usually it includes the current directory. The environment variable **FALCON_LOAD_PATH** and the command line option -L.

When the main module is found, its path is added to the module search path; in other words, there isn't any need to specify the path containing the main module to have other modules in its same directory to get loaded. The main module and other source Falcon module it loads can be stored in a directory that is not listed in the module search path; indicating an absolute or relative path as the **main_script** parameter will add that path on top of the active search path.

If not differently specified, falcon will search for and will load those ones instead of compiling the sources.

Options past the script name will be directly passed in the **args[]** global variable of the script.

The interpreter is compatible with the UNIX script execution directive "#!". A main script can have on the very first line of the code the directive

#!/path/to/falcon

to declare to the shell that the falcon command line is to be loaded. If falcon command line interpreter is also in the system PATH environment variable, which is usually the case of a normal installation, then the interpreter directive may also be simply

#!/bin/env falcon

It is then simply necessary to make the main script executable with

chmod 744 script_name

to be able to call the script directly.

Scripts executed in this way will add their path to the falcon module load path as soon as they are loade, so other modules referenced by them will be searched in the directory where they resides before being searched elsewhere.

Options to the falcon compiler may be passed normally by writing them after the execution directive in the main script.

Since version 0.8.12, the falcon command line interpreter has also an interactive mode which accepts statements and provide results as the expressions are evaluated.

OPTIONS

-a

Assembles the given script. The resulting module is not executed; this options causes Falcon to just assemble the module and exit. By default, the output module is written in a file with the same name of the input file with the ".fam" extension.

-c

Compile but do not execute. This makes falcon to compile the given module into a .fam file and then terminate. By default, the .fam file is written to a file with the same name as the input one, with the .fam extension.

-C

Check for memory leaks in VM execution. Like the -M option of faltest, this function sets the falcon engine memory allocators to slower functions that checks for memory to be allocated and deallocated correctly during the execution of a module. If the script is executed, before Falcon exits it writes a small report to the standard output.

-d <directive>=<value>

Sets the given directive to the desired value. Compilation directives and their values are the ones that scripts can set through the directive statement.

-D <constant>=<value>

Sets the given constant to the desired value. Constants are made available at compile time, and can be employed in macro and meta compilation.

-e

Set given encoding as default for VM I/O. Unless the scripts select a different I/O encoding, the streams that are provided to the falcon VM (i.e. the output stream for printf) are encoded using the given ISO encoding. This overrides the default encoding that is detected by reading the environment settings. In example, if your system uses iso-8859-1 encoding by default, but you want your script to read and write utf-8 files, use the option

-e utf-8

The **-e** option also determines the default encoding of the source files. To override this, use **-E**

-E

Set source script encoding. As **-e** , but this determines only the encoding used by falcon when loading the source scripts. This options overrides **-e** values, so it can be used to set the script encoding when they have to read and write from different encodings.

-f

Force recompilation of modules even when .fam are found.

-h

Prints a brief help on stdout and exits.

-i

Interactive mode. Falcon interpreter reads language statements from a prompt and present evaluation results to the user.

-l <lang_code>

Select a different language code for internationalized programs. This option loads an alternate string table for all the modules loaded. If the table doesn't exist or if the modules doesn't have a .ftr file containing the translation, the operation silently files and the original strings are used instead. Language codes should be in the international ISO format of five characters with a language name, an underscore and the regional code, like in “en_US”.

-L

Changes the default load path. This overrides both the internal built in settings and the contents of environment variable FALCON_LOAD_PATH. Each directory in the path should be separated by ";" and use forward slashes, like this:

falcon -L ./;/usr/share/falcon_mod;./myapp

-o

Redirects output to <fn>. This is useful to control the output of falcon when using options as -c, -a, -S etc. If **<fn>** is a dash (-) the output is sent to stdout.

-m

Use temporary files for intermediate steps. By default compilation is completely performed in memory; this option makes falcon to use temporary files instead.

-M

Do NOT save the compiled modules in '.fam' files.

-p

Preloads the given module as if it were loaded by the main script.

-r

Ignore source files and only use available .fam. This does not affects the main script; use the -x option if also the main script is a pre-compiled .fam module and source script must be ignored.

-s

Instead of compiling the source and generate a virtual machine code directly, falcon compiles all the sources into an intermediate assembly representation, and then it compiles the given assembly. Usually, you won't use this option unless you are debugging the assembly generator.

-S

Produce an assembly output. Writes an assembly representation of the given script to the standard output aznd the exit. Use -o to change file destination.

-t

Generates a syntactic tree of the source and writes it to the standard output, then exits. The syntactic tree is a representation of the script that is known by the compiler and used by the generators to create the final code. This option is useful when debugging the compiler and to test for the correct working of optimization algorithm.

-T

Force input parsing as a Falcon Template Document. Normally, only files ending with ".ftd" (case sensitive) are parsed as template document; when this switch is selected, the input is treated as a template document regardless of its name.

-v

Prints copyright notice and version and exits.

-w

After execution, requires the user to confirm program termination by pressing <enter>. This helps in point & click environments, where Falcon window is closed as soon as the program terminates.

-x

Executes a pre-compiled .fam module.

-y

Creates a template file for internationalization. This option creates a single .ftt file from a single source, .fam module or binary module. By default, the name of the template is the same as the module plus “.temp.ftt” added at the end; it is possible to change the destination template file using the -o option.

FILES

/usr/lib/libfalcon_engine.so

Default location of the Falcon Engine loadable module.

/usr/lib/falcon

Default directory containing Falcon binary modules.

ENVIRONMENT

FALCON_LOAD_PATH
Default search path for modules loaded by the scripts.

FALCON_SRC_ENCODING
Default encoding for the source scripts loaded by falcon (when different from the system default).

FALCON_VM_ENCODING
Default encoding for the VM I/O streams (when different from system default).

Falcon disassembler

SYNOPSIS

faldissass [options] *module_file.fam*

DESCRIPTION

The **flcdissass** command line tool disassembles a compiled .fam Falcon module. The output human-readable Falcon Virtual Machine assembly is sent to the standard output.

The tool has mainly two working modes. The standard mode procues an assembly dump containing the PC counter address that will be associated with each instruction in the VM. This allows to see exactly on which VM instruction an error was raised (as the PC at error raisal is always shown in error dumps), or to debug the VM by following the PC register in step-by-step mode.

The isomorphic mode creates a compilable assembly source that can the be feed into the falcon assembler to obtain a compiled module. In example, this can be used for VM level hand-made finetuning optimizations.

OPTIONS

-d
: Dump the dependency table (list of load directives).

-h
: Show version and a short help.

-i
: Create an isomorphic version of the original assembly.

-l
: add line informations.

-s
: Dump the string table.

-S
: Write the strings inline instead of using #strid

-y
: Dump the symbol table.

FILES

/usr/lib/libfalcon_engine.so
: Default location of the Falcon Engine loadable module.

Falcon module launcher

SYNOPSIS

falrun [options] [main_script] [script_options]

DESCRIPTION

The **falrun** command line tool is a subset of the **falcon** (1) command line interpreter which only executes pre-compiled falcon scripts in ".fam" module files. This is meant to launch Falcon based application which were compiled on a different system, and whose source is not shipped.

OPTIONS

-e
: Set given encoding as default for VM I/O. Unless the scripts select a different I/O encoding, the streams that are provided to the falcon VM (i.e. the output stream for printf) are encoded using the given ISO encoding. This overrides the default encoding that is detected by reading the environment settings. In example, if your system uses iso-8859-1 encoding by default, but you want your script to read and write utf-8 files, use the option

  -e utf-8

-h
: Prints a brief help on stdout and exits.

-L
: Changes the default load path. This overrides both the internal built in settings and the contents of environment variable FALCON_LOAD_PATH. Each directory in the path should be separated by ";" and use forward slashes, like this:

  falrun -L "./;/usr/share/falcon_mod;./myapp"

-p
: Preloads the given module as if it were loaded by the main script.

-v
: Prints copyright notice and version and exits.

FILES

/usr/lib/libfalcon_engine.so
: Default location of the Falcon Engine loadable module.

/usr/lib/falcon
: Default directory containing Falcon binary modules.

ENVIRONMENT

FALCON_LOAD_PATH

Default search path for modules loaded by the scripts.

FALCON_VM_ENCODING
: Default encoding for the VM I/O streams (when different from system default).

The Falcon test suite tool

SYNOPSIS

faltest [-d unit_test_dir] [options] [unit_list] *module_file.fam*

DESCRIPTION

The **faltest** command line tool is a powerful interface to the Falcon unit testing system.

The basic working principle of faltest is that of taking all the .fal script files contained in a directory, compile and execute them, eventually keeping track of errors, elapsed times and execution failures. After running all the scripts, **faltest** may print a report on what happened if requested to do so.

A list of one or more unit test may be indicated in the **faltest** command line after the options. Also, the executed tests can be limited to named subsets.

The unit test directory is added to the module load path, so **load** directives will be resolved searching the required scripts in the test path.

UNIT TEST SCRIPTS

Scripts being part of unit test have to start with a common header indicating some information about them. The header is actually a formatted Falcon comment which is read by the faltest utility.

This is a typical header:

```
/*****************************************
* Falcon test suite
*
* ID: 10a
* Category: cat-name
* Subcategory: subcat-name
* Short: The short name
* Description:
*   Long description of this test
*   Spanning on many lines...
* [/Description]
*****************************************/
```

The header has a free form; faltest recognizes the following fields, being inside a comment and eventually preceded by a "*".

ID:

The only mandatory field, it declares the ID under which this unit test is known. It will be used in listing the tests and in selecting them as argument of the **faltest** command line. The id must be an integer number, eventually followed by a single lowercase letter. Similar tests should be filed under the same ID with a different specification letter.

Scripts not having this field will be ignored by faltest.

Category:

The name of the category under which this test is filed. Faltest can select a subset of scripts to be executed to a

certain category.

Subcategory:

The name of the subcategory under which this test is filed. Faltest can select a subset of scripts to be executed to a certain subcategory.

Short:

Short description (or symbolic name) for this unit test.

Description:

A longer description explaining what this test is supposed to do. The description always spans on more lines, and is closed by a **[/Description]** tag.

THE TESTSUITE MODULE

Falcon system provides a loadable module called "testsuite". The module is actually embedded in **faltest** , and is provided to all the scripts it runs. The module provides the following functions:

success()

The script is terminated and recorded as a success.

failure(reason)

The script is terminated and recorded as a failure. An optional parameter containing a description of the failure condition may be optionally provided; it will be written as part of the report and may be used to track which part of the test wasn't working.

testReflect(item)

Returns the passed item. This is used to test for engine responsiveness to external calls and item copy through external functions.

alive(percent)

In tests running for some human sensible time, this function should be called periodically to inform the system and the user that the test is proceeding.

An optional "percent" parameter can be provided. It will be interpreted as a value between 1 and 100 representing the amount of test that has been performed up to this moment.

alive(percent)

In tests running for some human sensible time, this function should be called periodically to inform the system and the user that the test is proceeding.

An optional "percent" parameter can be provided. It will be interpreted as a value between 1 and 100 representing the amount of test that has been performed up to this moment.

timings(total_time, performed_ops)

In case the execution time is relevant for the test, like in benchmarks, this function can be used to communicate

back to **faltest** the time elapsed in the operations being tested and the number of operations performed. Those parameters will be recorded and eventually saved in the report file, to be used as benchmarks against falcon engine modifications.

timeFactor()

Lengthy tests are often performed by looping over the operation to be tested for a certain time. Benchmarks and performance tests should be written so that they can normally complete in a reasonable time, between one and ten seconds. In case the user wants the tests to perform longer, in order to obtain better statistical data, it can pass the -T option to **faltest** command line. The time factor will be a number greater than 1, and should be used by tests that may perform lengthy operation to customize the number of performed tests.

OPTIONS

-c
: Select this category and ignore the rest.

-C
: Select this subcategory and ignore the rest.

-d
: Directory from where to load the unit tests. If not specified, it will be the current directory.

-f
: Set time factor to N. Some scripts may use this information to perform more loops or lengthy operations.

-h
: Show version and a short help.

-l
: List the selected tests and exit. Combine with **-v** to have the path of the tests, as **-l** only lists the script ID, its short name and the category/subcategory pair.

-L
: Changes the default load path. This overrides both the internal built in settings and the contents of environment variable FALCON_LOAD_PATH. Each directory in the path should be separated by ";" and use forward slashes, like this:

faltest -L ./;/usr/share/falcon_mod;./myapp

-m
: Do **not** compile in memory. Use disk temporary files.

-M
: Checks for memory leaks. This option makes faltest to report unclaimed memory after each script execution, and a final report at the end of the tests. The check is extended to all the engine operations, so errors in the engine are detected too.

-o
: Write final report to the given output file.

-s
: Perform module serialization test. Other than compiling the file, the module is also saved and then restored

before being executed. This allows to check for errors in module serialization (that is, loading of .fam files). The operation is performed in memory, unless the option **-m** is also specified.

-S

Compile via assembly. This test the correct behavior of the assembler generator and compiler instead of the binary module generator.

-t

Records and display timings. The statistics of compilation, linking and execution overall times are recorded and written as part of the report.

-T

Records timings() calls from scripts. This allows the scripts to declare their own performance ratings, and collects the results in the final report.

-v

Be verbose. Normally, execution and failures are sparsely reported. This is because the normal execution mode is meant for automated runs. Tests can be executed by automated utilities and errors can be reported to system administrator by simple checks on the output data.

A developer willing to fix a broken test must run that test alone with the **-v** enabled. A more complete error report (including compilation or execution errors, if they were the cause for the failure) will be then visualized. The -v options also allows to see the path of the original script, which is otherwise hidden (masked by the testsuite ID).

-V

Prints version number and exits.

SAMPLE

This is a simple and complete example from the Falcon benchmark suite.

```
/*****************************************
* Falcon direct benchmarks
*
* ID: 2a
* Category: benchmark
* Subcategory: calls
* Short: Benchmark on function calls
* Description:
*    Performing repeated function calls and returns.
*    This test calls a function without parameters.
*
* [/Description]
*****************************************/

loops = 1000000 * timeFactor()
each = int(loops/10)

function toBeCalled()
end

// getting time
time = seconds()
```

```
for i = 1 to loops

   // perform the call
   toBeCalled()

   if i % each == 0
      alive( i/loops*100 )
   end
end
// taking end time
time = seconds() - time

// subtract alive time
timeAlive = seconds()
for i = 1 to loops
   if i % each == 0
      alive( i/loops*100 )
   end
end
timeAlive = seconds() - timeAlive
time -= timeAlive

timings( time, loops )

/* end of file */
```

FILES

/usr/lib/libfalcon_engine.so

Default location of the Falcon Engine loadable module.

Falconeer – module configurator

SYNOPSIS

falconeer.fal -n moduleName *[options]*

DESCRIPTION

The falconeer.fal script configures a directory containing the Falcon Skeleton Module so that it becomes ready for compilation under all the systems supported by Falcon.

Although not mandatory, a developer willing to write native modules for Falcon may use this facility to speed up the startup phase and begin with an already compilable module.

The configuration consists in the renaming of the module files into the module name specified in the command line, and in the update of the makefiles and project files provided for the various development platform Falcon can be built on.

Other than the project name, the script allows to configure other options, that will affect the template variables that will be substituted in the modified files.

Once configured and built, the skeleton module provides already a skeleton() symbol that is exported to the VM, and a service that exports that function (defined in the fskelmod_mod.cpp file) to C++ applications.

OPTIONS

-a <author>
: Specifies the author name.

-c <Description>
: Indicates the copyright owner, if different from the author, to be set on the copyright line, right beside the copyright year.

-d <Description>
: Sets a brief description of the process.

-l <file>
: Loads a license plate (a long statement indicating the license under which the files are distributed) from a template file. If not given, the standard FPLL license plate is applied to the generated files.

-n <name>
: Sets the (short) name of the project. Files will be renamed accordingly to this value, and also the final module name will be configured using this setting.

-p <name>

Sets the main project hood under which the file is created. Usually, modules are part of wider projects, or can be distributed as sets. If not set, the text "The Falcon Programming Language" will be used instead.

BUGS

The file version.h cannot currently be properly configured. Edit it by hand.

NOTES

On some systems, falconeer.fal script can be “proxied” with a *falconeer* pseudo command (shell script, batch file and so on).

Fallc – Language table compiler

SYNOPSIS

fallc.fal [-m merge_file] [...*options*...] ftt_list...

DESCRIPTION

The Falcon language table compiler is used to compile .ftt files (Falcon translation tables) into a binary .ftr file (Falcon translation). The ftr file is a collection of translations relative to a single module (source file, .fam pre-compiled Falcon file or binary shared object/dynamic load library), that can be shipped side by side with the module and providing a translation to one of the compiled-in languages on request.

Ftt files are generated by falcon command line compiler using the -y option.

The program operates on the list of ftt files provided on the command line.

OPTIONS

-c
: Do NOT check for inline variables consistency. If not given, fallc will check for all the variables indicated with $varname or $(varname[:...]) to be both in the original strings and in the translation, and will warn if this doesn't happens.

-h
: Prints an help screen with command synopsis.

-m <source_table.ftt>
: Merges a new source table with previously performed translations. If the source program that must be translated changes after that some work has been performed on the translation table, it is necessary to integrate the changes into the work files. In this mode, a new template source table generated by falcon can be integrated in already existing translations, which will be modified directly on place. It is possible to provide a different output file for the merged result using the -o <outputfile> option, but in this case it will be possible to merge only one file at a time.

-o <outputfile>
: By default, fallc sends the binary translation table to a file named after the original module with the .ftr extension added. This option allows to specify a different destination for the compiled translation table. The option can be used in conjunction with the -m option to send the result of a merged table to a different file.

-v
: Prints program version and exits.

NOTES

On some systems, fallc.fal script can be “proxied” with a *fallc* pseudo command (shell script, batch file and so on).