# Quick Reference Guide - Linux

eVariX is Silkan's solution to boost your Matlab®/Scilab modeling process. Its technology relies on a source-to-source compiler, Cold, which transforms your models into optimized C++11 codes for single or multi-core CPUs, and GPUs. eVariX's driver, `evarix`, directly produces stand-alone executables, `mex` or `sciw` files from the generated C++ files.

**Package content: eVariX-2.6.0**

eVariX compiler (Cold): `cold-2.6.4`
eVariX driver: `evarix-1.7.3`
Cold run-time library: `crt-2.6.3`

**Web site:** `www.evarix.eu`
**Support:** `evarix.support@silkan.net`

## Installation

**Pre-requisites:**

C++11 compiler:

g++ $\geq$ 4.8 or Intel® icc $\geq$ 14.0

optional: OpenCV, OpenCL

**Installation:**

Extract `evarix-2.6.0-linux_x64.tar.gz`
Add to `PATH`: `EVARIX_INSTALL_DIR/bin`
Set OpenCV path in `evarix.cfg` file

**Getting licence:**

Run `rlmhostid` in `EVARIX_INSTALL_DIR/etc`
Send the hostID to `evarix.support@silkan.net`
Copy licence file to `EVARIX_INSTALL_DIR/etc`

## Usage and Main Options

**Syntax:**

```
evarix inFile [-o outFile] [Options]
```

**Driver options:**

1.General Options

| | |
|---|---|
| –help, -h | Display help |
| –verbose, -v | Increase verbose level |
| –version, -version | Display version |
| –output=<filename>,-o= | Set the name of the output file or name of Matlab or Scilab function |
| –keep | Keep the C++ file generated by Cold |
| –cold-only | Generates the C++ file only |
| –advanced-options | Display advanced options |

2.Backend Options

| | |
|---|---|
| -cc=<compiler> | Change the backend compiler |
| -Wc=<option> | Pass <option> on to the back end compiler |
| -Wl=<option> | Pass <option> on to the linker |
| -Wm=<option> | Pass <option> on to the MEX compiler |
| -Ws=<option> | Pass <option> on to the SCIW compilation |

**Compiler options:**

1. General Options

| | |
|---|---|
| –path=<dir> | Add <dir> to paths recursively searched for functions files |
| –import-lib=<filename> | Provides the path to an external lib description file (such as /path/to/libuser.m) |
| –complex | Computation of some functions are done in C |
| –wrn-level, -wN=<N> | Set warning level. $1 \leq N \leq 3$, default 1 |
| –advanced-options | Display advanced options. |

2. Code Generation & Backend Options

| | |
|---|---|
| –mex | Generate a MEX Wrapper |
| –sci | Generate a SCI Wrapper |
| –scilab-root <dir> | Set the path to the Scilab root directory |
| –no-comment, -nc | Disable comments in generated code |
| –no-prefix, -np | Disable prefixing of user defined variables. |
| –stack-size=<N[B\|kB\|mB\|gB]> | Set the size in bytes of the CRT stack <N> must be an integer, the unit is optional (default is 1G) |

# Quick Reference Guide - Linux

## Advanced Options

**Driver options:**

| | |
|---|---|
| –cold-log-file=<file> | Redirect the output stream of COLD to <file> |
| –configuration-file=<configFile> | Select a configuration file |
| –select-configuration=<configName> | Select a specific configuration |
| –check-install-matlab | Check installation for Matlab configuration |
| –check-install-mex | Check installation for MEX configuration |
| –check-install-scilab | Check installation for Scilab configuration |
| –check-install-sciw | Check installation for Sciw configuration |
| –check-install | Check default installation |
| –check-overall | Check installation of all compilers and libraries |
| –emulate | Only print the command. No compilation is done |

**Compiler options:**

1. Optimization Options

| | |
|---|---|
| –inline-level=<level> | Set inline expansion level. <int> should be 0, 1 or 2, default is 0 (beta version). |
| –inline-function=<function> | Expand inline the <name> function if –inline-level is >= 1 (beta version) |
| –inline-function-pattern=<pattern> | Expand inline functions whose name begins with <pattern> if –inline-level is >= 1 (beta version) |
| –inline-add-comments | Surround inlined code by comments giving the name of the inlined function and the actual/formal parameter correspondances |
| –fes-level=<level> | Set the forward expression substitution level. <level> can be 0, 1 or 2, default is 0. |
| –array-static-indexing, -asi | Specify that array sizes are not modified through array indexing. |

2. Parallelization Options

| | |
|---|---|
| –enable-parallel-pragmas, -epp | Enable parallel pragmas. |
| –auto-parallelization-level=<int> | Set automatic parallelization level. <int> can be:<br>0: no automatic parallelization (default) 1: vector/matrices operations are parallelized<br>2: well-formed for-loops are parallelized 3: both are parallelized |
| –parallelization-report-level=<int> | Set the reporting level of the automatic parallelization passes. <level> can be 0, 1 or 2, default is 0 (no output). |
| –parallel-for-threshold=<int> | Set threshold for 'for'-loop automatic parallelization, default is 1000. |
| –gpu-codegen-policy=<int> | Set the Gpu communications codegen policy. <level> should be 0, 1 or 2, default is 1. |
| –gpu-log | Generate code with GPU usage logs. |
| –enable-openmp-crt | Enable the use of the OpenMP versions of the CRT functions. |

3. Other:

| | |
|---|---|
| –input-information | Display detailed information about input source code. |
| –inline-max-stmt-nb=<int> | Set max number of statements in caller after inlining a function call. Default is 199. |
| –inline-max-weight=<int> | Set max weight of caller after inlining a function call, default is 199. |
| –log-file=<filename> | Log all streams of COLD into file. |
| –print-crt-default | Displays default CRT version and quit. |
| –pragma-entry=<entryDescr> | Specify entry point and its argument types (see syntax below). |

# Quick Reference Guide - Linux

## Loading `evarix` in the Matlab® environment

**Lauching Matlab®:**

```
LD_PRELOAD=/path/to/your/libstdc++.so matlab
```

**Loading `evarix`:**

```
addpath('EVARIX_INSTALL_DIR/bin/');
```

## Loading `evarix` in the Scilab environment

**Lauching Scilab:**

```
LD_PRELOAD=/path/to/your/libstdc++.so scilab
```

**Loading `evarix`:**

```
exec('EVARIX_INSTALL_DIR/bin/evarix_init.sce');
```

## Matlab® functions

**evarix_exec**`(script)`
**evarix_exec**`(script, options)`

compiles the script and runs the generated executable.

*script*: name of the script file (string)
*options*: compilation options (cell array of strings)

**evarix_compile**`(script, func, s1,..., sn)`
**evarix_compile**`(script, func, options, s1,..., sn)`

compiles the input function and add it to the current workspace.

*script*: name of the script or function to compile
*func*: name of the function to generate, compilation builds `func.mexa64`
*s1, ..., sn*: variables of same type as the input parameters of *func*.
*options*: compilation options (cell array of strings)

`entry_fmt =` **evarix_get_entry_format**`(args)`

returns the representation of the type/shape of *arg* in the eVariX syntax for an entry option or annotation

*args*: set of variables of supported types

## Scilab functions

**evarix_exec**`(script.sce)`
**evarix_exec**`(script.sce, options)`

compiles the script and runs the generated executable.

*script*: name of the script file (string)
*options*: compilation options (array of strings)

**evarix_compile**`([script.sce,inputfunc],func, s1,...,sn)`
**evarix_compile**`([script.sce,inputfunc],func,options,s1,...,sn)`

compiles the input function and add it to the current workspace.

*script*: name of the script to compile
*inputfunc*: name of the function to compile
*func*: name of the function to generate, compilation builds `func.mexa64`
*s1, ..., sn*: variables of same type as the input parameters of *func*.
*options*: compilation options (array of strings)

`entry_fmt =` **evarix_get_entry_format**`(args)`

returns the representation of the type/shape of *arg* in the eVariX syntax for an entry option or annotation

*args*: set of variables of supported types

# Quick Reference Guide - Linux

## Supported Matlab® features

**Data types:**

Scalars: integers, double precision reals, character strings
Arrays: row and column vectors, matrices, ND-arrays
Structures
Objects

**Expressions:**

Constants: `eps`, `i`, `Inf`, `j`, `NaN`, `pi`
Colon notation: `a:b` or `a:b:c`
Array definition: `[1.1 2.2 3.3; 4.4 5.5 6.6]`
Array indexing: `A(3)`, `A(:)`, `A(2:4)`, `A(end-1:2)`, `A(B>3)`

**Control flow:**

`if`, `for`, `while`, `switch`

**User functions:**

`nargin`, `nargout`

**Classes:**

Properties with initial values
Methods: constructors, ordinary or static methods
Inheritance from `handle` class
Method calls with *dot* notation: `obj.foo(n)`

## Supported Scilab features

**Data types:**

Scalars: integers, double precision reals, character strings
Arrays: row and column vectors, matrices, ND-arrays
Structures

**Expressions:**

Constants: `%e`, `%eps`, `%i`, `%Inf`, `%j`, `%NaN`, `%pi`
Colon notation: `a:b` or `a:b:c`
Array definition: `[1.1 2.2 3.3; 4.4 5.5 6.6]`
Array indexing: `A(3)`, `A(:)`, `A(2:4)`, `A($-1:2)`, `A(B>3)`

**Control flow:**

`if`, `for`, `while`, `select`

**User functions**

## Specifying types and shapes:

| | | |
|---|---|---|
| **scalar types**: | | `bool`, `int`, `int8`, `uint8`, `double`, `complex`, `string` |
| **shapes**: | row vector | `row{<T>}` or `row{<T>, n}` for fixed-size vector of type <T> |
| | column vector | `col{<T>}` or `col{<T>, n}` for fixed-size vector of type <T> |
| | matrix | `matrix{<T>}` or `matrix{<T>, m, n}` for fixed-size matrix of type <T> |
| | ND-array: | `array{<T>,N}`, `N` is the number of dimensions |

# Quick Reference Guide - Linux

## Annotations

**Optimization:**

Inlining:

```
%pragma evarix inline
```

Static parameter:

```
%pragma evarix static parameter N
```

Loop invariant:

```
%pragma evarix invariant [in loop(I)]
```

**Parallelization:**

Parallel block: (triggered by `--enable-parallel-pragmas`)

```
%pragma evarix parallel block [onvariable(var-list)] [on(gpu)]
...  statements
%pragma evarix end
```

Parallel Region: (triggered by `--enable-parallel-pragmas`)

```
%pragma evarix parallel [regionkind] [clauses]
```

`regionkind`: `for`, `element-wise`
`clauses`: `nthreads(N)`, `if(condition)`, `firstprivate|lastprivate|private|shared(var-list)`

Gpu section: (triggered by `--enable-parallel-pragmas` or `--auto-parallelization-level > 1`)

```
%pragma evarix begin on(gpu)
...  statements
%pragma evarix end
```

**Driving the compilation process:**

Ignore section of code:

```
%pragma evarix ignore
...  statements
%pragma evarix end
```

Set internal stack size:

```
%pragma evarix stacksize N
```

Specify entry function argument types and shapes:

```
%pragma evarix entry(funcname, list-of-types)
```

Set scalar type of variable:

```
%pragma evarix type (var_name, scalar-type)
```