Volatile Plug-in Manual

For Frama-C 33.0 beta (Arsenic)

Patrick Baudin

Contents

Overview 1

This is the user manual of the Volatile plug-in of Frama-C[1]. The content of this document corresponds to the version 33.0 beta (Arsenic), released on June 25, 2026, of Frama-C.

This plug-in adds to Frama-C support for C qualifier `volatile`. It builds a new project where volatile accesses (loads from and stores to volatile data) are simulated by function calls, according to the ACSL [1] clauses `volatile`. Therefore, it is useful to perform analyses on this new project without taking care of the `volatile` qualifiers since the introduced function calls model the effects of these volatile accesses.

The detection of volatile accesses is purely syntactic. The plug-in looks at the l-values of the source code. ACSL [1] clauses `volatile` should inform the plug-in about the functions to introduce for modeling the accesses to the l-values having the qualifier `volatile` or being composed of a subpart (i.e. a field) having that `volatile` qualifier. A warning message is emitted when no function is found for such l-value.

The plug-in warns also about casts from pointers to volatile data since accesses to these data may be undetected by the syntactic search.

An instrumentation of calls through function pointers is also available.

[1] http://frama-c.com

Volatile 2

You may refer to the user manual of Frama-C [2] for non-specific options of Frama-C.
After its installation, the plug-in is listed by `frama-c -plugins` as `Volatile`.
The plug-in options are:

- `-volatile-help` lists all options specific to the plug-in.
- `-volatile` invokes the plug-in.
- `-volatile-fct` <*f, ...*> specifies the list of functions to process (defaults to all functions).
- `-volatile-binding` <*f, ...*> allows binding of the volatile accesses to the specified functions.
- `-volatile-binding-auto` allows automatic binding to functions from the type name of the volatile accesses: functions `Rd_`<*type-name*> and `Wr_`<*type-name*> for read and write accesses (the use of the option `-volatile-msg-key="binding"` could help to guess the necessary <*type-name*>).
- `-volatile-binding-prefix` <*str*>: adds <*str*> as prefix to function name for automatic binding (defaults to `c2fc2_`).
- `-volatile-binding-call-pointer` replaces calls through function pointers by direct calls to functions:
  <*prefix*>`Call_`<*result-type-name*>`_`<*param-type-names*>
- `-volatile-call-pointer` <*f, ...*>: stub call to pointer functions to the provided functions (indexed by type).

The use of the plug-in could be the following:

```
> frama-c -volatile <volatile options> <C files> \
    -then-on Volatile -set-project-as-default \
    -then <analysis options>
```

Here is a small example C program contained in file `demo.c`:

```
volatile int v;
int job(int x) {
  v=x;
  return v;
}

//@ logic integer wr_trans(integer status, int x) = status+x;
//@ logic integer rd_trans(integer status) = status-1;
//@ logic integer rd_value(integer status) = status;

int state_v = 0;

/*@ requires p == &v;
  @ assigns state_v;
  @ ensures state_v==wr_trans(\old(state_v),x);
```

```
  @*/
int wr_v (int volatile *p, int x) ;

/*@ requires p == &v;
  @ assigns state_v;
  @ ensures state_v==rd_trans(\old(state_v));
  @ ensures \result==rd_value(\old(state_v));
  @*/
int rd_v (int volatile *p) ;

//@ volatile v reads rd_v writes wr_v;
```

Here is the result of the about the `job` function:

```
> frama-c -volatile demo.c -then-last -print
```

```
int job(int x){
  int __volatile_tmp;
  wr_v(& v,x);
  __volatile_tmp = rd_v(& v);
  return __volatile_tmp;
}
```

When the plug-in is used in combination with the ACSL importer plug-in, the use could be the following:

```
frama-c <importing options> <C files> \
  -then -volatile <volatile options> \
  -then-on Volatile <analysis options>
```

Binding for volatile accesses 3

The volatile accesses can be bound to functions from (sorted by decreasing priority order):

1. global `volatile` clauses written into *ACSL* (as into the previous example)
2. the option `-volatile-binding`
3. the option `-volatile-binding-auto`

When no binding is found for an lvalue, a dedicated warning message is emitted. However, these warnings can be silent using `-volatile-no-warning-on-volatile-lvalues`.

The prototype of the functions bound using the options must be compatible to the ones specified into global `volatile` clauses of *ACSL*:

- <*T*> <*function-name*>`(volatile` <*T*> `*)` for read accesses to lvalues of type <*T*>
- <*T*> <*function-name*>`(volatile` <*T*> `*, <T>)` for write accesses to lvalues of type <*T*>

For the binding feature related to the `-volatile-binding`, using the `-volatile-basetype` breaks that rule since it enables a search among the functions specified by `-volatile-binding` from the base type of the volatile access to instrument, and not the real C type. Mainly, the type attributes and qualifiers are not considered during the search. Of course, casts are introduced when necessary into the code instrumentation to get a compilable C code.

When using automatic binding, the option `-volatile-msg-key="binding"` could help to guess the necessary <*function-name*> in giving the inferred <*type-name*>.

Function Pointers 4

The Volatile plug-in is also capable of instrumenting calls through function pointers. For instance, the following statement:

```
r = (*p)(a,b,c);
```

can be instrumented by a static call to some well-chosen function `F`:

```
r = F(p,a,b,c);
```

There are two mechanisms for choosing the appropriate function `F` associated to a function pointer:

- Either an option `-volatile-call-pointer F` is used and the prototype of `F` is compatible with the type of `(*p)`;
- Or the option `-volatile-binding-call-pointer` is set and `F` has a canonical name, like for volatile read/write functions.

The canonical name is `<prefix>Call_<result-type>_<param-type>_..._<param-type>`, where:

- `<prefix>` is set by option `-volatile-binding-prefix <prefix>`
- `<result-type>` is the result type name of the call through function pointer
- `<param-type>` is the type name of the parameter of the call (using the same order from the left to right)

Changelog

Frama-C 32.0 (Germanium)

- The plug-in is now part of the Frama-C open-source distribution

From plugin Volatile 1.9.xx and Frama-C 19 Potassium

- Changes for `-volatile-basetype` option about pointer types
- Breaks type circularities visiting type attributes and leading to stack overflow

From plugin Volatile 1.8.xx and Frama-C 18 Argon

- Using warning categories for some messages
- Added option `-volatile-basetype`

From plugin Volatile 1.7.xx and NUPW-v4.06, Frama-C 16 Sulfur

- Added option `-volatile-call-pointer` and `-volatile-binding-call-pointer` to instrument call to function pointers
- Fixes some bugs (c.f. Changelog file).

From plugin Volatile 1.6.xx and Frama-C 15 Phosphorus

- Added option `-volatile-fct` to process only a list of functions

From plugin Volatile 1.3.xx and NUPW-v2.02, Frama-C 12 Magnesium

- Fixes some bugs (c.f. Changelog file).

From plugin Volatile 1.2.xx and Frama-C 9 Fluorine

– Fixes some bugs (c.f. Changelog file).

From plugin Volatile 1.1.xx and Frama-C 8 Oxygen

– Options for automatic binding added.
– Fixes some bugs (c.f. Changelog file).

From plugin Volatile 1.0.xx and Frama-C 7 Nitrogen

– Initial document revision.

Bibliography

[1] Patrick Baudin, Pascal Cuoq, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto. *ACSL: ANSI/ISO C Specification Language*. Latest Frama-C implementation release at https://frama-c.com/download/frama-c-acsl-implementation.pdf.

[2] Loïc Correnson, Pascal Cuoq, Florent Kirchner, André M Maroneze, Virgile Prevosto, Armand Puccetti, Julien Signoles, and Boris Yakobowski. *Frama-C User Manual*. CEA LIST, Software Safety Laboratory. Latest release at https://frama-c.com/download/frama-c-user-manual.pdf.