firehose Documentation

Release 0.5

David Malcolm

Jul 25, 2017

Contents

"firehose" is a Python package intended for managing the results from code analysis tools (e.g. compiler warnings, static analysis, linters, etc).

It currently provides parsers for the output of gcc, clang-analyzer, cppcheck, and findbugs. These parsers convert the results into a common data model of Python objects, with methods for lossless roundtrips through a provided XML format. There is also a JSON equivalent.

It is available on pypi here: https://pypi.python.org/pypi/firehose

and via git from: https://github.com/fedora-static-analysis/firehose

The mailing list is: https://admin.fedoraproject.org/mailman/listinfo/firehose-devel

Firehose is Free Software, licensed under the LGPLv2.1 or (at your option) any later version.

It requires Python 2.7 or 3.2 onwards, and has been successfully tested with PyPy.

It is currently of alpha quality.

The API and serialization formats are not yet set in stone (and we're keen on hearing feedback before we lock things down more).

Contents:

CHAPTER 1

Motivation

Motivation: http://lists.fedoraproject.org/pipermail/devel/2012-December/175232.html

We want to slurp the results from static code analysis into a database, which means coercing all of the results into some common interchange format, codenamed "firehose" (which could also be the name of the database).

The idea is a common XML format that all tools can emit that:

- describes a warning
- gives source-code location of the warning: filename, function, line number.
- optionally with a CWE identifier
- potentially with other IDs and URLs, e.g. the ID "SIG30-C" with URL https://www.securecoding.cert.org/confluence/display/seccode/SIG30-C.+Call+only+asynchronous-safe+functions+within+signal+handlers
- optionally describes code path to get there (potentially interprocedural across source files), potentially with "state" annotations (e.g. in the case of a reference-counting bug, it's useful to be able to annotate the changes to the refcount).

together with a simple Python API for working with the format as a collection of Python objects (creating, write to XML, read from XML, modification, etc)

The data can be round-tripped through both XML and JSON.

There is a RELAX-NG schema for validating XML files.

References to source files in the format can include a hash of the source file itself (e.g. SHA-1) so that you can uniquely identify which source file you were talking about.

This format would be slurped into the DB for the web UI, and can have other things done to it without needing a server: e.g.:

- convert it to the textual form of a gcc compilation error, so that Emacs etc can parse it and take you to the source
- be turned into a simple HTML report locally on your workstation

Projects using Firehose:

- mock-with-analysis can rebuild a source RPM, capturing the results of 4 different code analysis tools in Firehose format (along with all source files that were mentioned in any report).
- The "firehose" branch of cpychecker can natively emit Firehose XML reports
- https://github.com/paultag/storz/blob/master/wrappers/storz-lintian

CHAPTER 2

Examples

A first example

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="cpychecker" version="0.11"/>
    <sut>
      <source-rpm name="python-ethtool" version="0.7" release="4.fc19" build-
↪arch="x86_64"/>
    </sut>
    <file given-path="examples/python-src-example.c">
      <hash alg="sha1" hexdigest="6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
    </file>
    <stats wall-clock-time="5"/>
  </metadata>

  <results>
    <!-- Example of a warning without a trace -->
    <issue cwe="681" test-id="mismatching-type-in-pyarg-format-string">
      <message>Mismatching type in call to PyArg_ParseTuple with format code
↪"i"</message>
      <notes>  argument 3 ("&amp;amp;count") had type
    "long int *" (pointing to 64 bits)
  but was expecting
    "int *" (pointing to 32 bits)
  for format code "i"</notes>
      <location>
        <file given-path="examples/python-src-example.c">
          <hash alg="sha1" hexdigest=
↪"6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
        </file>
        <function name="make_a_list_of_random_ints_badly"/>
        <point line="29" column="26"/>
```

```
      </location>
      <custom-fields>
        <str-field name="function">PyArg_ParseTuple</str-field>
        <str-field name="format-code">i</str-field>
        <str-field name="full-format-string">i</str-field>
        <str-field name="expected-type">"int *" (pointing to 32 bits)</str-
↪field>
        <str-field name="actual-type">"long int *" (pointing to 64 bits)</
↪str-field>
        <str-field name="expression">&amp;count</str-field>
        <int-field name="argument-num">3</int-field>
      </custom-fields>
    </issue>
  </results>
</analysis>
```

Example with a trace of activity

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="cpychecker" version="0.11"/>
    <sut>
      <source-rpm name="python-ethtool" version="0.7"
                  release="4.fc19" build-arch="x86_64"/>
    </sut>
  </metadata>

  <results>
    <issue cwe="401" test-id="refcount-too-high">

  <!-- Example of a report with a trace -->

  <message>ob_refcnt of '*item' is 1 too high</message>
  <notes>was expecting final item->ob_refcnt to be N + 1 (for some unknown N)
due to object being referenced by: PyListObject.ob_item[0]
but final item->ob_refcnt is N + 2</notes>

  <location>
    <file given-path="examples/python-src-example.c">
      <hash alg="sha1" hexdigest="6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
    </file>
    <function name="make_a_list_of_random_ints_badly"/>
    <point line="40" column="4"/>
  </location>

  <trace>
    <state>
      <location>
        <file given-path="examples/python-src-example.c">
          <hash alg="sha1" hexdigest=
↪"6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
        </file>
        <function name="make_a_list_of_random_ints_badly"/>
        <point line="36" column="14"/>
```

```
      </location>
      <notes>PyLongObject allocated at:         item = PyLong_
↪FromLong(random());</notes>
    </state>

    <state>
      <location>
        <file given-path="examples/python-src-example.c">
          <hash alg="sha1" hexdigest=
↪"6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
        </file>
        <function name="make_a_list_of_random_ints_badly"/>
        <point line="37" column="8"/>
      </location>
      <notes>when PyList_Append() succeeds</notes>
    </state>

    <state>
      <location>
        <file given-path="examples/python-src-example.c">
          <hash alg="sha1" hexdigest=
↪"6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
        </file>
        <function name="make_a_list_of_random_ints_badly"/>
        <point line="40" column="4"/>
      </location>
    </state>
  </trace>
    </issue>
  </results>
</analysis>
```

Example of analysis failures

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="cpychecker" version="0.11"/>
    <sut>
      <source-rpm name="python-ethtool" version="0.7" release="4.fc19" build-
↪arch="x86_64"/>
    </sut>
    <file given-path="examples/python-src-example.c">
      <hash alg="sha1" hexdigest="6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
    </file>
    <stats wall-clock-time="5"/>
  </metadata>

  <results>
    <!-- Example of an analysis failure where we have nothing except the
         knowledge of a segfault: -->
    <failure failure-id='bad-exit-code'>
      <custom-fields>
        <int-field name="returncode">-11</int-field>
      </custom-fields>
```

```
    </failure>
  </results>
</analysis>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="cpychecker" version="0.11"/>
    <sut>
      <source-rpm name="python-ethtool" version="0.7" release="4.fc19" build-
↪arch="x86_64"/>
    </sut>
    <file given-path="wspy_register.c">
      <hash alg="sha1" hexdigest="6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
    </file>
    <stats wall-clock-time="5"/>
  </metadata>

  <results>
    <!-- Example of an analysis failure where we have a traceback
         and the location of the code that broke the checker: -->
    <failure failure-id="python-exception">
      <location>
        <file given-path="wspy_register.c"/>
        <function name="register_all_py_protocols_func"/>
        <point line="159" column="42"/>
      </location>
      <custom-fields>
        <str-field name="traceback">wspy_register.c: In function 'register_
↪all_py_protocols_func':
wspy_register.c:159:42: error: Unhandled Python exception raised calling
↪'execute' method
Traceback (most recent call last):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/_
↪_init__.py", line 75, in execute
    self._check_refcounts(fun)
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/_
↪_init__.py", line 79, in _check_refcounts
    self.show_possible_null_derefs)
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪refcounts.py", line 3668, in check_refcounts
    limits=limits)
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
```

```
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/
↪absinterp.py", line 2929, in iter_traces
    depth + 1):
```

```
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2929, in iter_traces
    depth + 1):
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2893, in iter_traces
    transitions = curstate.get_transitions()
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2013, in get_transitions
    return self._get_transitions_for_stmt(stmt)
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2029, in _get_transitions_for_stmt
    return self._get_transitions_for_GimpleCall(stmt)
  File "/usr/lib/gcc/x86_64-redhat-linux/4.6.2/plugin/python2/libcpychecker/absinterp.py", line 2212, in _get_transitions_for_GimpleCall
    raise NotImplementedError('not yet implemented: %s' % fnname)
NotImplementedError: not yet implemented: PySequence_Check
</str-field></custom-fields>
    </failure>
  </results>
</analysis>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="cpychecker"/>
  </metadata>
  <results>
    <!--
       Example of a failure-to-analyze in which we have an error message
       and a location, but other failure fields (stdout, returncode)
       wouldn't make sense and so are omitted.

       The error message is a warning from cpychecker that the results
       are only a partial analysis; it's not achieving full coverage.

       (this was added to cpychecker in:
          http://git.fedorahosted.org/cgit/gcc-python-plugin.git/commit/?h=firehose&id=1fbb678bb121099a8161031aae9e39c75e3faea7 )
    -->
    <failure failure-id="too-complicated">
      <location>
        <file given-path="tests/cpychecker/refcounts/combinatorial-explosion/input.c"/>
        <function name="test_adding_module_objects"/>
        <point column="1" line="31"/>
      </location>
      <message>this function is too complicated for the reference-count checker to fully analyze: not all paths were analyzed</message>
    </failure>
```

```
  </results>
</analysis>
```

Example of ranges

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="cpychecker" version="0.11"/>
    <sut>
      <source-rpm name="python-ethtool" version="0.7" release="4.fc19" build-
↪arch="x86_64"/>
    </sut>
    <file given-path="examples/python-src-example.c">
      <hash alg="sha1" hexdigest="6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
    </file>
    <stats wall-clock-time="5"/>
  </metadata>

  <results>
    <!-- Example of a warning that uses a range -->
    <issue cwe="681" test-id="mismatching-type-in-pyarg-format-string">
      <message>Mismatching type in call to PyArg_ParseTuple with format code
↪"i"</message>
      <notes>  argument 3 ("&amp;count") had type
    "long int *" (pointing to 64 bits)
  but was expecting
    "int *" (pointing to 32 bits)
  for format code "i"</notes>
      <location>
        <file given-path="examples/python-src-example.c">
          <hash alg="sha1" hexdigest=
↪"6ba29daa94d64b48071e299a79f2a00dcd99eeb1"/>
        </file>
        <function name="make_a_list_of_random_ints_badly"/>
        <range>
          <point line="10" column="9"/>
          <point line="10" column="44"/>
        </range>
      </location>
    </issue>
  </results>
</analysis>
```

Debian Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="handmade" version="0.1"/>
    <sut>
      <debian-source name="python-ethtool" version="0.7" release="4.1+b1" />
```

```
    </sut>
  </metadata>

  <results>
      <!-- we check for results elsewhere, no need to populate this with
           senseless error messages. -->
  </results>
</analysis>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis>
  <metadata>
    <generator name="handmade" version="0.1"/>
    <sut>
        <debian-binary name="python-ethtool" version="0.7" release="1.1"
            build-arch="amd64" />
    </sut>
  </metadata>

  <results>
      <!-- we check for results elsewhere, no need to populate this with
           senseless error messages. -->
  </results>
</analysis>
```

etc

CHAPTER 3

Data model

class `firehose.model.`**`Analysis`**

The *`Analysis`* class represents one invocation of a code analysis tool.

It corresponds to the `<analysis>` XML element, the top-level element of a Firehose XML document.

`metadata`
`Metadata`

`results`
A list of *`Result`* objects, representing the various issues, failures, and other information found during the analysis.

`customfields`
`CustomFields` or `None`

Here is the pertinent part of the XML schema:

```
<start>
  <!-- Results from the invocation of an analysis tool -->
  <element name="analysis">
    <ref name="metadata-element"/>
    <element name="results">
      <zeroOrMore>
        <choice>
          <ref name="issue-element"/>
          <ref name="failure-element"/>
          <ref name="info-element"/>
        </choice>
      </zeroOrMore>
    </element>
    <optional>
      <ref name="custom-fields-element"/>
    </optional>
  </element>
</start>
```

__init__(self, metadata, results, customfields=None):

Parameters

- **metadata** (*Metadata*) –
- **results** (list(*Result*)) –
- **customfields** (*CustomFields* or None) –

classmethod from_xml(*cls*, *fileobj*)
Parse XML from fileobj, and return an *Analysis* instance representing the data seen there.

to_xml(*self*)
Generate an `ET.ElementTree()` representing the data within self.

to_xml_bytes(*self*)
Generate a `bytes` instance containing an XML serialization of the data within self.

Results

class `firehose.model.`**Result**
Result is a base class

There are three subclasses:

- an *Issue* represents a report from the analyzer about a possible problem with the software under test.
- an *Info* represents additional kinds of information generated by an analyzer that isn't a problem per-se e.g. code metrics, licensing info, etc.
- a *Failure* represents a report about a failure of the analyzer itself (e.g. if the analyzer crashed).

class `firehose.model.`**Issue**(*Result*)
An *Issue* represents a report from the analyzer about a possible problem with the software under test.

It corresponds to the `<issue>` XML element within a Firehose XML document.

cwe
(`int` or `None`): The Common Weakness Enumeration ID (see http://cwe.mitre.org/index.html) e.g. "131" representing CWE-131 aka "Incorrect Calculation of Buffer Size" http://cwe.mitre.org/data/definitions/131.html

testid
(`str` or `None`): Each static analysis tool potentially has multiple tests, with its own IDs for its own tests. These can be captured here, as free-form strings.

location
(*Location*): Where is the problem?

message
(*Message*): A message summarizing the problem.

notes
(*Notes* or `None`): Additional descriptive details.

trace
(*Trace* or `None`): An optional list of events that describe the circumstances leading up to a problem.

severity
(`str` or `None`): Each static analysis tool potentially can report a "severity", which may be of use for filtering.

The precise strings are likely to vary from tool to tool. To avoid data-transfer issues, support storing it as an optional freeform string here.

See: http://lists.fedoraproject.org/pipermail/firehose-devel/2013-February/000001.html

customfields

(*CustomFields* or `None`): A given tool/testid may have additional key/value pairs that it may be useful to capture.

write_as_gcc_output(*self*, *out*)
Write the issue in the style of a GCC warning to the given file-like object.

```
>>> issue.write_as_gcc_output(sys.stderr)
examples/python-src-example.c:40:4: warning: ob_refcnt of '*item' is 1 too␣
↪high [CWE-401]
was expecting final item->ob_refcnt to be N + 1 (for some unknown N)
due to object being referenced by: PyListObject.ob_item[0]
but final item->ob_refcnt is N + 2
examples/python-src-example.c:36:14: note: PyLongObject allocated at:      ␣
↪item = PyLong_FromLong(random());
examples/python-src-example.c:37:8: note: when PyList_Append() succeeds
```

get_cwe_str(*self*)
Get a string giving the CWE title, or None:

```
>>> issue.get_cwe_str()
'CWE-131'
```

get_cwe_url(*self*)
Get a string containing the URL of the CWE id, or None:

```
>>> issue.get_cwe_url()
'http://cwe.mitre.org/data/definitions/131.html'
```

class firehose.model.**Info**(*Result*)
An *Info* represents additional kinds of information generated by an analyzer that isn't a problem per-se e.g. code metrics, licensing info, cross-referencing information, etc.

It corresponds to the <info> XML element within a Firehose XML document.

infoid
(`str` or `None`): an optional free-form string identifying the kind of information being reported.

location
Location or `None`

message
Message or `None`

customfields
CustomFields or `None`

class firehose.model.**Failure**(*Result*)
A *Failure* represents a report about a failure of the analyzer itself (e.g. if the analyzer crashed).

If any of these are present then we don't have full coverage.

For some analyzers this is an all-or-nothing affair: we either get issues reported, or a failure happens (e.g. a segfault of the analysis tool).

Other analyzers may be more fine-grained: able to report some issues, but choke on some subset of the code under analysis. For example cpychecker runs once per function, and any unhandled Python exceptions only affect one function.

It corresponds to the `<failure>` XML element within a Firehose XML document.

failureid
(`str` or `None`): Each static analysis tool potentially can identify types of way that it can fail.

Capture those that do here, as (optional) free-form strings.

location
Location: Some analysis tools may be able to annotate a failure report by providing the location *within the software-under-test* that broke them.

For example, gcc-python-plugin has a `gcc.set_location()` method which can be used by a code analysis script to record what location is being analyzed, so that if unhandled Python exception happens, it is reported at that location. This is invaluable when debugging analysis failures.

message
Message: A summary of the failure.

customfields
CustomFields or `None`: Every type of failure seems to have its own kinds of data that are worth capturing:

- stdout/stderr/returncode for a failed subprocess
- traceback for an unhandled Python exception
- verbose extra information about a cppcheck failure

etc. Hence we allow a `<failure>` to optionally contain extra key/value pairs, based on the `failureid`.

Metadata

class `firehose.model.`**Metadata**
Holder for metadata about an analyzer invocation.

It corresponds to the `<metadata>` XML element within a Firehose XML document.

generator
Generator

sut
Sut or `None`

file_
File or `None`

stats
Stats or `None`

class `firehose.model.`**Generator**

name
`str`

version
str or None

class firehose.model.**Stats**
Stats is an optional field of *Metadata* for capturing stats about an analysis run.

wallclocktime
float: how long (in seconds) the analyzer took to run

Describing the software under test

Warning: this part of the schema may need more thought/work

class firehose.model.**Sut**
Base class for describing the software-under-test.

class firehose.model.**SourceRpm**(*Sut*)
It corresponds to the <source-rpm> XML element within a Firehose XML document.

name
str

version
str

release
str

buildarch
str

class firehose.model.**DebianBinary**(*Sut*)
Internal Firehose representation of a Debian binary package. This Object is extremely similar to a SourceRpm.

It corresponds to the <debian-binary> XML element within a Firehose XML document.

name
str: the *binary* package name.

version
str: should match Upstream's version number

release
str or None: should be the Debian package local version. This should only be omited if the package is a Debian Native package.

buildarch
str: valid entries include amd64`', ``kfreebsd-amd64, armhf, hurd-i386, among others for Debian.

class firehose.model.**DebianSource**(*Sut*)
Internal Firehose representation of a Debian source package. This Object is extremely similar to a SourceRpm, but does not include the *buildarch* attribute.

It corresponds to the <debian-source> XML element within a Firehose XML document.

name
str: should be the *source* package name

version
str: should match Upstream's version number

release
str or None: if given, should be the Debian package local version. This should only be omited if the package is a Debian Native package.

class firehose.model.**Message**
Summary text aimed at a developer. This is required for an *Issue*, but is also can (optionally) be provided by a *Failure* or *Info*.

It corresponds to the <message> XML element within a Firehose XML document.

text
str

class firehose.model.**Notes**
Additional optional descriptive details for a *Result* or for a *State*.

It corresponds to the <notes> XML element within a Firehose XML document.

text
str

Describing source code

class firehose.model.**Location**
A particular source code location.

It corresponds to the <location> XML element within a Firehose XML document.

file
File

function
Function or None. The function (or method) containing the problem.

This is optional. Some problems occur in global scope, and unfortunately, some analyzers don't always report which function each problem was discovered in. Given that function names are less likely to change than line numbers, this is something that we should patch in each upstream analyzer as we go.

We can refer to either a location, or a range of locations within the file:

point
Point or None

range_
Range or None

class firehose.model.**File**
A description of a particular source file.

It corresponds to the <file> XML element within a Firehose XML document.

givenpath
str: the filename given by the analyzer.

This is typically the one supplied to it on the command line, which might be absolute or relative.

Examples:
- "foo.c"
- "./src/foo.c"

•"/home/david/libfoo-1.0/src/foo.c"

abspath
(`str` or `None`): Optionally, a record of the absolute path of the file, to help deal with collating results from a build that changes working directory (e.g. recursive make).

hash_
(*Hash* or `None`)

class `firehose.model.`**Hash**
An optional value within *File*, allowing the report to specify a hash value for a particular file.

This can be used for tracking different versions of files when collating different reports and e.g. for caching file content in a UI.

It corresponds to the `<hash>` XML element within a Firehose XML document.

alg
`str`: the name of the hash algorithm.

TODO: what naming convention?

hexdigest
`str`: the hexadecimal value of the digest (lower-case hexdigits, without any leading *0x*).

class `firehose.model.`**Function**
Identification of a particular function within source code.

It corresponds to the `<function>` XML element within a Firehose XML document.

name
`str`: the name of the function or method.

class `firehose.model.`**Point**
Identification of a particular line/column within a source file.

It corresponds to the `<point>` XML element within a Firehose XML document.

line
`int`: the 1-based number of the line containing the point

column
`int`: 1-based number of the column

Note: GCC uses a 1-based convention for source columns, whereas Emacs's `M-x column-number-mode` uses a 0-based convention.

For example, an error in the initial, left-hand column of source line 3 is reported by GCC as:

```
some-file.c:3:1: error: ...etc...
```

On navigating to the location of that error in Emacs (e.g. via `next-error`), the locus is reported in the Mode Line (assuming `M-x column-number-mode`) as:

```
some-file.c   10%   (3, 0)
```

i.e. `3:1:` in GCC corresponds to `(3, 0)` in Emacs.

---

class `firehose.model.`**Range**
Identification of a range of text within a source file.

It corresponds to the `<range>` XML element within a Firehose XML document.

start
(*Point*)

end
(*Point*)

Capturing the circumstances leading up to a problem

class firehose.model.**Trace**
An optional list of events within an *Issue* that describe the circumstances leading up to a problem.

It corresponds to the <trace> XML element within a Firehose XML document.

See *example of a trace*.

states
list of *State*

class firehose.model.**State**
A state within a *Trace*.

location
Location

notes
Notes or None

Other data

class firehose.model.**CustomFields**(*OrderedDict*)
A big escape-hatch in the data model: support for arbitrary, ordered key/value pairs for roundtripping data specific to a particular situation. e.g. debugging attributes for a particular failure

It corresponds to the <custom-fields> XML element within a Firehose XML document.

CHAPTER 4

Parsers

There are various parsers that take the output of specific analyzers and turn them into *firehose.model.Analysis* instances.

- clanganalyzer.py

  Parser for the `.plist` files emitted by the clang-static-analyzer, when `-plist` is passed as an option to "scan-build" or "clang"

- cppcheck.py

  Parser for output from cppcheck, specifically, version 2 of its XML format as generated by:

  ```
  cppcheck PATH_TO_SOURCES --xml --xml-version=2
  ```

- findbugs.py

  Parser for xml output from findbugs.

- frama_c.py

  Parser for warnings emitted by frama-c.

- gcc.py

  Parser for warnings emitted by GCC.

- flawfinder.py

  Parser for warnings emitted by flawfinder.

- splint.py

  Parser for the `-csv` output format from splint.

CHAPTER 5

Schema for the XML format

For reference, here's the RELAX-NG schema for the XML serialization format:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright 2013 David Malcolm <dmalcolm@redhat.com>
    Copyright 2013 Red Hat, Inc.

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301
    USA
-->
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <!-- Results from the invocation of an analysis tool -->
    <element name="analysis">
      <ref name="metadata-element"/>
      <element name="results">
        <zeroOrMore>
          <choice>
            <ref name="issue-element"/>
            <ref name="failure-element"/>
            <ref name="info-element"/>
          </choice>
```

```
        </zeroOrMore>
      </element>
      <optional>
        <ref name="custom-fields-element"/>
      </optional>
    </element>
  </start>

  <define name="metadata-element">
      <element name="metadata">
        <element name="generator">
          <attribute name="name"/>
          <optional>
            <attribute name="version"/>
          </optional>
        </element>

        <!-- "sut" = "Software Under Test" -->
        <optional>
          <element name="sut">
            <choice>
              <element name="source-rpm">
                <attribute name="name"/>
                <attribute name="version"/>
                <attribute name="release"/>
                <attribute name="build-arch"/>
              </element>
              <!-- Debian SUT entries -->
              <element name="debian-source">
                <!-- Report for a Debian source package -->
                <attribute name="name"/>
                <attribute name="version"/>
                <optional>
                  <!-- This entry is optional because Debian packages may be
                       `native' (e.g. no local version, since local and
                       upstream are the same). -->
                  <attribute name="release"/>
                </optional>
                <!-- No build arch; source is arch indep. -->
              </element>
              <element name="debian-binary">
                <!-- Report for a Debian .deb package -->
                <attribute name="name"/>
                <attribute name="version"/>
                <optional>
                  <attribute name="release"/>
                </optional>
                <attribute name="build-arch"/>
                <!-- Valid entries include `amd64', `kfreebsd-amd64', `armhf
↪',
                     `hurd-i386', among others for Debian. -->
              </element>
              <!-- What other options should we have? -->
            </choice>
          </element>
        </optional>

        <optional>
```

```
        <ref name="file-element"/>
      </optional>

      <optional>
        <element name="stats">
          <!-- actual time taken to run the analysis, in seconds -->
          <attribute name="wall-clock-time">
            <data type="float"/>
          </attribute>
        </element>
      </optional>

    </element>
  </define>

  <!--
    Definitions of the various kinds of result follow:
      <issue>
      <failure>
      <info>
  -->

  <!-- A report about a possible problem -->
  <define name="issue-element">
    <element name="issue">
      <optional>
        <!-- The Common Weakness Enumeration ID
             (see http://cwe.mitre.org/index.html )
             e.g. "131" representing CWE-131
             aka "Incorrect Calculation of Buffer Size"
             http://cwe.mitre.org/data/definitions/131.html
        -->
        <attribute name="cwe">
          <data type="integer"/>
        </attribute>
      </optional>

      <optional>
        <!--
            Each static analysis tool potentially has multiple tests,
            with its own IDs for its own tests.
            Capture those that do here, as free-form strings:
        -->
        <attribute name="test-id"/>
      </optional>

      <optional>
        <!--
            Each static analysis tool potentially can report a "severity",
            which may be of use for filtering.

            The precise strings are likely to vary from tool to tool.
            To avoid data-transfer issues, support storing it as an optional
            freeform string here.

            See:
            http://lists.fedoraproject.org/pipermail/firehose-devel/2013-
↪February/000001.html
```

```
      -->
      <attribute name="severity"/>
    </optional>

    <!-- A message summarizing the problem -->
    <ref name="message-element"/>

    <!-- Additional descriptive details
         This might support some simple markup at some point
         (as might <message>) -->
    <optional>
      <element name="notes"><text/></element>
    </optional>

    <!-- Where is the problem? -->
    <ref name="location-element"/>

    <optional>
      <!-- How can the problem occur? -->
      <element name="trace">
        <oneOrMore>
          <element name="state">
            <ref name="location-element"/>
            <optional>
              <element name="notes"><text/></element>
            </optional>
            <!-- optionally we can supply key-value pairs -->
            <zeroOrMore>
              <element name="annotation">
                <element name="key"><text/></element>
                <element name="value"><text/></element>
              </element>
            </zeroOrMore>
            </element>
        </oneOrMore>
      </element>
    </optional>

    <!--
       A given tool/testid may have additional key/value pairs that it
       may be useful to capture:
    -->
    <optional>
      <ref name="custom-fields-element"/>
    </optional>

  </element>
</define>

<!--
    A report about a failed analysis.

    If any of these are present then we don't have full coverage.

    For some analyzers this is an all-or-nothing affair: we either
    get issues reported, or a failure happens (e.g. a segfault of the
    analysis tool).
```

```
    Other analyzers may be more fine-grained: able to report some issues,
    but choke on some subset of the code under analysis.  For example
    cpychecker runs once per function, and any unhandled Python exceptions
    only affect one function.
-->
<define name="failure-element">
  <element name="failure">
    <optional>
      <!--
          Each static analysis tool potentially can identify types of way
          that it can fail.

          Capture those that do here, as (optional) free-form strings:
      -->
      <attribute name="failure-id"/>
    </optional>
    <optional>
      <!--
         Some analysis tools may be able to annotate a failure report by
         providing the location *within the software-under-test* that
         broke them.

         For example, gcc-python-plugin has a gcc.set_location() method
         which can be used by a code analysis script to record what
         location is being analyzed, so that if unhandled Python exception
         happens, it is reported at that location.  This is invaluable
         when debugging analysis failures.
      -->
      <ref name="location-element"/>
    </optional>
    <optional>
      <!-- summary of the failure -->
      <ref name="message-element"/>
    </optional>

    <!--
       Every type of failure seems to have its own kinds of data that
       are worth capturing:
         * stdout/stderr/returncode for a failed subprocess
         * traceback for an unhandled Python exception
         * verbose extra information about a cppcheck failure
       etc.
       Hence allow a <failure> to optionally contain extra key/value
       pairs, based on the failure-id.
    -->
    <optional>
      <ref name="custom-fields-element"/>
    </optional>
  </element>
</define>

<!--
   Sometimes you may want a tool to report other kinds of information
   about the software-under-test that isn't a problem as such, e.g.
   code metrics, copyright/license info, cross-referencing information
   etc, hence the <info> element:
-->
<define name="info-element">
```

```
    <element name="info">
      <optional>
        <!--
            An optional free-form string identifying the kind of information
            being reported:
        -->
        <attribute name="info-id"/>
      </optional>
      <optional>
        <ref name="location-element"/>
      </optional>
      <optional>
        <ref name="message-element"/>
      </optional>
      <optional>
        <ref name="custom-fields-element"/>
      </optional>
    </element>
  </define>

  <!-- ...end of result definitions.  Various supporting elements follow: -->

  <!--
     Summary text aimed at a developer.  This is required for an <issue>,
     but is also can (optionally) be provided by a <failure> or <info>
  -->
  <define name="message-element">
    <element name="message"><text/></element>
  </define>

  <define name="location-element">
    <!-- A particular source code location -->
    <element name="location">
      <ref name="file-element"/>

      <!--
           Ideally, every analyzer would tell us in which function each
           problem was discovered, given that function names are less
           likely to change than line numbers.

           Unfortunately many don't - and we should patch these in each
           upstream analyzer as we go.

           Also, a problem can occur in global scope (e.g. lack of NULL
           termination in an array-initializer for a global, such as in
           this checker:
           http://gcc-python-plugin.readthedocs.org/en/latest/cpychecker.html
↪#verification-of-pymethoddef-tables
           (although arguably there *is* a relevant function there: the
           location of the code that uses that data)
      -->

      <optional>
        <element name="function">
          <attribute name="name"/>
        </element>
      </optional>
      <!-- We can refer to either a location, or a range of locations
```

```
          within the file: -->
      <choice>
        <ref name="point-element"/>
        <element name="range">
          <!-- start of range: -->
          <ref name="point-element"/>
          <!-- end of range: -->
          <ref name="point-element"/>
        </element>
      </choice>
    </element>
  </define>

  <define name="file-element">
      <element name="file">
        <!--
           What filename was given by the analyzer?

           This is typically the one supplied to it on the command line,
           which might be absolute or relative.

           Examples:
           - "foo.c"
           - "./src/foo.c"
           - "/home/david/libfoo-1.0/src/foo.c"
        -->
        <attribute name="given-path"/>

        <!--
            Optionally, record the absolute path of the file,
            to help deal with collating results from a build that changes
            working directory (e.g. recursive make)
        -->
        <optional>
          <attribute name="absolute-path"/>
        </optional>
        <optional>
          <element name="hash">
            <attribute name="alg"/>
            <attribute name="hexdigest"/>
          </element>
        </optional>
      </element>
  </define>

  <define name="point-element">
    <element name="point">
      <attribute name="line"/>
      <attribute name="column"/>
    </element>
  </define>

  <!--
     A big escape-hatch in the schema: support for arbitrary, ordered
     key/value pairs for roundtripping data specific to a particular
     situation.  e.g. debugging attributes for a particular failure
  -->
  <define name="custom-fields-element">
```

```
    <element name="custom-fields">
      <zeroOrMore>
        <choice>
          <element name="str-field">
            <attribute name="name"/>
            <text/>
          </element>
          <element name="int-field">
            <attribute name="name"/>
            <data type="integer"/>
          </element>
        </choice>
      </zeroOrMore>
    </element>
  </define>

</grammar>
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

P

R

S

T

V

W