

OpenSCENARIO 2.0 Implementers Forum

Introduction to the Language

Pierre R. Mai

PMSF IT Consulting

2022-01-17

Teams



Introduction to the Language

Key Language Features

Python-like syntax

- Line structured
- Indentation-based blocks

Nature of language

- Declarative language
- Static type system
- Extensibility via inheritance and extension
- Conditional inheritance for classifications
- Constraint-based parameter resolution

Large-scale structuring

- Import mechanism
- Extensibility via external methods

Semantics

- Denotational semantics provided in standard
- Duality of interpretation of scenarios

Outlook / Postponed to Future Development (2.x)

- Namespaces for structuring of large sets of scenarios/libraries
- Standardized foreign function interface (FFI)
- More formal standardization of action semantics
- Operationalized semantic specification

Resources for working with the language

- EBNF grammar (using Python-style EBNF syntax)
- py-osc2 antlr4 parser
- Implementers Forum discussions

OpenSCENARIO 2.0 Syntax

- Python-like base Syntax
- Line structured
- Blocks are indentation-based
- Keywords to aid readability

```
import "foo.osc"
import osc.standard

scenario foo:
  field: float
  bar: float = 4.0

scenario baz inherits foo:
  newfield: float
  def mymethod(abc:float, dd:float = 42.0) -> float is expression abc*dd

scenario bazzle:
  another: float
  do baz(field: 2.0, newfield: another)

enum storm_type: [rain_storm, ice_storm, snow_storm]

struct storm_data:
  storm: storm_type
  wind_velocity: speed

scenario env.snowstorm:
  storm_data: storm_data with:
    keep(it.storm == snow_storm)
    keep(default it.wind_velocity >= 30kmph)
  cover(wind_velocity, expression: storm_data.wind_velocity, unit: kmph)
```

OpenSCENARIO 2.0 Language Constructs

Overall structure of a scenario file

- Prelude statements: `import`
- Type declarations:
 - Non-structured types:
 - Physical types / units
 - Enumerations
 - Aggregate types:
 - Lists
 - Structured types:
 - `struct`
 - `actor`
 - `scenario/action`
 - `modifier`
- Scenario entry point:
Implementation/user-defined in PRC

```
import "foo.osc"
import osc.standard

type speed is SI(m: 1, s: -1)
unit mps of speed is SI(m: 1, s: -1)
unit |m/s| of speed is SI(m: 1, s: -1)
unit kmph of speed is SI(m: 1, s: -1, factor: 3.6)

enum storm_type: [rain_storm, ice_storm, snow_storm]

struct storm_data:
    storm: storm_type
    wind_velocity: speed

actor env:
    air_temp: temperature

scenario env.snowstorm:
    storm_data: storm_data with:
        keep(it.storm == snow_storm)
        keep(default it.wind_velocity >= 30kmph)
        cover(wind_velocity, expression: storm_data.wind_velocity, unit: kmph)

modifier fast_storm of env.snowstorm:
    keep(it.storm_data.wind_velocity >= 80kmph)
```

OpenSCENARIO 2.0 Language Constructs

Structured types, contents and extension

- All structured types can contain:
 - Fields: Parameters / Variables
 - Events
 - Constraints
 - Methods
 - `cover/record`
- Scenarios/actions/modifiers add:
 - Modifier invocation
 - `on` directive
 - `do` directive (scenario/action only)
- Structured types can be extended:
 - (Single-)Inheritance (`inherits`)
 - Extension of type from outside (`extend`)
 - Inheritance can be conditional:
Instances are automatically
of a derived type, if condition holds.

```
scenario blackout:  
  duration: time  
  intermittent: bool  
  var variable: float = 4.0  
  event blackout_started(duration: time)  
  keep(default duration == 30s)  
  def calc_duration(step: time, times: float) -> time \  
    is expression step*times  
  cover(duration)  
  do emit blackout_started(duration: duration)
```

```
scenario level_blackout inherits blackout:  
  level: float  
  def calc_duration(step: time, times: float) -> time \  
    is only expression step*times + (1-level)*1s
```

```
extend level_blackout:  
  timefactor: time  
  def calc_duration(step: time, times: float) -> time \  
    is only expression step*times + (1-level)*timefactor
```

```
actor vehicle:  
  is_electric: bool
```

```
actor electric_vehicle inherits vehicle(is_electric: true):  
  battery_capacity: energy
```

OpenSCENARIO 2.0 Language Constructs

Behavior specification, temporal operators

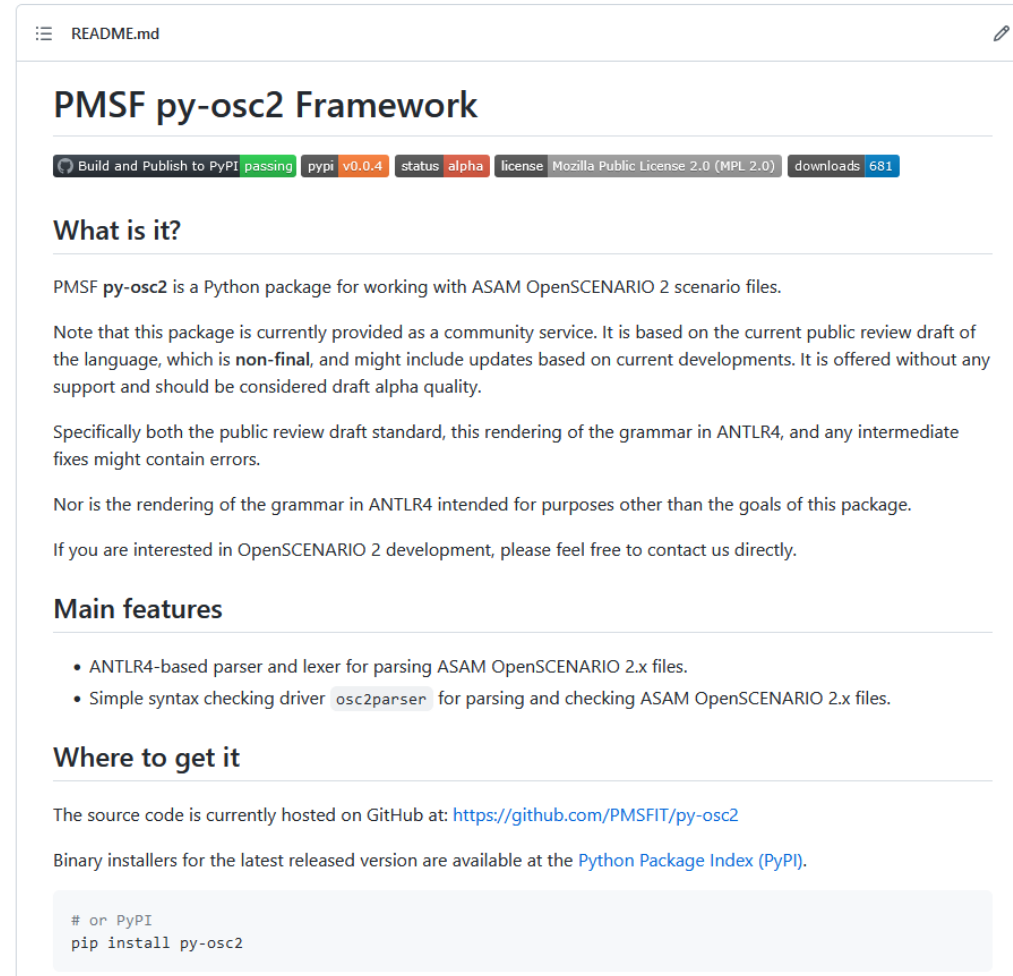
- Behavior specification
 - `on` directive: React to events
 - `do` directive: Unconditional behavior
- `on` directive content:
 - `emit` an event
 - `call` a method
- `do` directive content:
 - Invoke scenario/action
 - `wait` directive (wait for something)
 - `emit` an event
 - `call` a method
 - Temporal composition:
 - `serial`
 - `one_of`
 - `parallel`

```
scenario foo:  
  car: vehicle  
  event my_event  
  def my_method(abc: float, dd: float = 42.0) is external xxx.yyy()  
  
  on blackout.blackout_started as ev if ev.duration >= 20s:  
    call my_method(abc: 42)  
    emit my_event  
  
  do serial:  
    wait car.speed >= 50kmph  
    parallel:  
      car.drive()  
      blackout(duration: 30s)
```

PMSF py-osc2 Framework

Python-based Framework for OpenSCENARIO 2.x Files

- Currently provides:
 - ANTLR4-based parser
 - CLI tool for syntax checking
- Future enhancements:
 - More sophisticated syntax checking
 - Semantic checks (incl. Imports)
 - ...
- Status:
 - Initial alpha releases, based on PRC
 - Will be updated as PRC is finalized
- Available on PyPI, GitHub (MPL 2.0):
 - <https://pypi.org/project/py-osc2/>
 - <https://github.com/PMSFIT/py-osc2>



The screenshot shows the README.md file for the PMSF py-osc2 Framework. At the top, it says "PMSF py-osc2 Framework" and includes a badge for "Build and Publish to PyPI" with a "passing" status, a version badge for "v0.0.4", a status badge for "alpha", a license badge for "Mozilla Public License 2.0 (MPL 2.0)", and a downloads badge for "681". Below this, the text reads "What is it?" and describes the package as a Python package for working with ASAM OpenSCENARIO 2 scenario files. It notes that the package is currently provided as a community service based on a public review draft of the language, which is non-final and might include updates based on current developments. It is offered without any support and should be considered draft alpha quality. Specifically, both the public review draft standard, this rendering of the grammar in ANTLR4, and any intermediate fixes might contain errors. Nor is the rendering of the grammar in ANTLR4 intended for purposes other than the goals of this package. If you are interested in OpenSCENARIO 2 development, please feel free to contact us directly. The "Main features" section lists two items: "ANTLR4-based parser and lexer for parsing ASAM OpenSCENARIO 2.x files." and "Simple syntax checking driver `osc2parser` for parsing and checking ASAM OpenSCENARIO 2.x files." The "Where to get it" section states that the source code is currently hosted on GitHub at <https://github.com/PMSFIT/py-osc2> and that binary installers for the latest released version are available at the [Python Package Index \(PyPI\)](#). At the bottom, there is a code block with the text "# or PyPI" and "pip install py-osc2".

Languages



Q & A Session

Thank you for your attention!

Join us in the Implementers Forum!

Pierre R. Mai
PMSF IT Consulting

Phone: +49-8161-97696-11
Email: pmai@pmsf.de