pyA2L Documentation

Release 0.9

Christoph Schueler

Mar 27, 2020

Contents:

CHAPTER 1

Installation

Pythons: *Python* >= 3.4

Platforms: No platform-specific restrictions.

Documentation: get latest.

1.1 Prerequisites

CHAPTER 2

Building pyA2L

pyA2L uses code generated from ANTLR grammars for much of the heavy lifting when parsing ASAP2 documents. This code is not version controlled, but instead generated during build-time. That means that if you just `git clone`'d this repository, it is not yet ready to use. You must first build it, which requires a working Java installation to run ANTLR.

2.1 Prerequisites

1. Install Java
2. Install ANTLR

2.2 Code generation

To generate code, call setup.py with the `antlr` command:

```
python setup.py antlr
```

However, it is generally not necessary to run this command directly.

2.3 Development environment

If you want to contribute to the pyA2L project, running

```
python setup.py develop
```

will also auto-generate code.

2.4 Binary distribution

The same is true when creating a binary for distribution with

```
python setup.py bdist
```

or

```
python setup.py bdist_wheel
```

CHAPTER 3

Getting Started with pyA2L

You'll find the example code here.

3.1 Import an .a2l file to database

```
from pya2l import DB

db = DB()
session = db.import_a2l("ASAP2_Demo_V161.a2l")
```

If nothing went wrong, your working directory now contains a file named *ASAP2_Demo_V161.a2ldb*, which is simply a Sqlite3 database file.

Unlike other ASAP2 toolkits, you are not required to parse your *.a2l* files over and over again, which can be quite expensive.

3.2 Open an existing .a2ldb database

```
from pya2l import DB

db = DB()
session = db.open_existing("ASAP2_Demo_V161")    # No need to specify extension .a2ldb
```

You may have noticed, that in both cases the return value is stored in an object named *session*:

Enter SQLAlchemy!

SQLAlchemy offers, amongst other things, a powerful expression language.

3.3 Running a first database query

```
from pya2l import DB
import pya2l.model as model

db = DB()
session = db.open_existing("ASAP2_Demo_V161")
measurements = session.query(model.Measurement).order_by(model.Measurement.name).all()
for m in measurements:
    print("{:48} {:12} 0x{:08x}".format(m.name, m.datatype, m.ecu_address.address))
```

Yields the following output:

```
ASAM.M.ARRAY_SIZE_16.UBYTE.IDENTICAL              UBYTE        0x00013a30
ASAM.M.MATRIX_DIM_16_1_1.UBYTE.IDENTICAL          UBYTE        0x00013a30
ASAM.M.MATRIX_DIM_8_2_1.UBYTE.IDENTICAL           UBYTE        0x00013a30
ASAM.M.MATRIX_DIM_8_4_2.UBYTE.IDENTICAL           UBYTE        0x00013a30
ASAM.M.SCALAR.FLOAT32.IDENTICAL                   FLOAT32_IEEE 0x00013a10
ASAM.M.SCALAR.FLOAT64.IDENTICAL                   FLOAT64_IEEE 0x00013a14
ASAM.M.SCALAR.SBYTE.IDENTICAL                     SBYTE        0x00013a01
ASAM.M.SCALAR.SBYTE.LINEAR_MUL_2                  SBYTE        0x00013a01
ASAM.M.SCALAR.SLONG.IDENTICAL                     SLONG        0x00013a0c
ASAM.M.SCALAR.SWORD.IDENTICAL                     SWORD        0x00013a04
ASAM.M.SCALAR.UBYTE.FORM_X_PLUS_4                 UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.IDENTICAL                     UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.TAB_INTP_DEFAULT_VALUE        UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.TAB_INTP_NO_DEFAULT_VALUE     UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.TAB_NOINTP_DEFAULT_VALUE      UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.TAB_NOINTP_NO_DEFAULT_VALUE   UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.TAB_VERB_DEFAULT_VALUE        UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.TAB_VERB_NO_DEFAULT_VALUE     UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.VTAB_RANGE_DEFAULT_VALUE      UBYTE        0x00013a00
ASAM.M.SCALAR.UBYTE.VTAB_RANGE_NO_DEFAULT_VALUE   UBYTE        0x00013a00
ASAM.M.SCALAR.ULONG.IDENTICAL                     ULONG        0x00013a08
ASAM.M.SCALAR.UWORD.IDENTICAL                     UWORD        0x00013a02
ASAM.M.SCALAR.UWORD.IDENTICAL.BITMASK_0008        UWORD        0x00013a20
ASAM.M.SCALAR.UWORD.IDENTICAL.BITMASK_0FF0        UWORD        0x00013a20
ASAM.M.VIRTUAL.SCALAR.SWORD.PHYSICAL              SWORD        0x00000000
```

The classes describing an *.a2ldb* database live in pya2l.model, they are required to query, modify, and add model instances.

The test-suite found here is a good starting point for further experimentations, because it touches virtually every A2L element/attribute.

CHAPTER 4

Tutorial

CHAPTER 5

Configuration

CHAPTER 6

HOW-TOs

CHAPTER 7

pya2l

7.1 pya2l package

7.1.1 Subpackages

7.1.2 Submodules

7.1.3 pya2l.a2l module

7.1.4 pya2l.a2lListener module

7.1.5 pya2l.a2lparser module

7.1.6 pya2l.aml module

7.1.7 pya2l.amlLexer module

7.1.8 pya2l.amlListener module

7.1.9 pya2l.amlParser module

7.1.10 pya2l.amlVisitor module

7.1.11 pya2l.amlcl module

7.1.12 pya2l.amllib module

7.1.13 pya2l.classes module

7.1.14 pya2l.ifdata module

7.1.15 pya2l.logger module

7.1.16 pya2l.parserlib module

CHAPTER 8

Indices and tables

- genindex
- modindex
- search