# text(X)text or 'textX - lightweight xtext alternative for python'

MARKDOWN Just stumpled over textX an xText inspired python-toolbox for creating DSLs. Super simple to use. I love xText but it can be really a pain in the ass here an there. Mainly because you (at least me) always forget about the internal concept and/or structure. Especially how to wire what and how to get the dependency injection working.... :D I actually wanted to give the xText Theia-Workflow a go but instead stumpled over [textX](https://github.com/textX/textX). It is a super lightweight DSL generator and AST-Parser and seems to have an [addon](https://github.com/textX/textX-LS) to create Visual Studio Code addons via [Language Server Protocol](https://en.wikipedia.org/wiki/Language_Server_Protocol). Didn't tried that, yet. A grammar would look like this (very similiar to xText): ``` /* Entity DSL grammar. */ EntityModel: types*=SimpleType // At the beginning of model we can define // zero or more simple types. entities+=Entity // Each model has one or more entities. ; Entity: 'entity' name=ID '{' properties+=Property // Each entity has one or more properties. '}' ; Property: name=ID ':' type=[Type] // type is a reference to Type instance. // There are two built-in simple types // registered on meta-model in entity_test.py ; // Type can be SimpleType or Entity Type: SimpleType | Entity ; SimpleType: 'type' name=ID ; // Special rule for comments. Comments start with // Comment: /\/\/.*$/ ; ``` A sample Model using this grammar: ``` entity Person { name : string // A comment is everything after // to the end of line address: Address // It is defined by the Comment rule in the grammar age: integer // integer and string are built-in objects } // See entity_test.py entity Address { street : string city : string country : string } ``` And to let it work. This couple of lines including custom type (SimpleType),predefined data, obviously parsing and and simple generation: ``` from os import mkdir from os.path import exists, dirname, join from textx import metamodel_from_file this_folder = dirname(__file__) class SimpleType(object): def __init__(self, parent, name): self.parent = parent self.name = name def __str__(self): return self.name def get_entity_mm(): """ Builds and returns a meta-model for Entity language. """ type_builtins = { 'integer': SimpleType(None, 'integer'), 'string': SimpleType(None, 'string') } entity_mm = metamodel_from_file(join(this_folder, 'entity.tx'), classes=[SimpleType], builtins=type_builtins) return entity_mm def main(debug=False): # Instantiate the Entity meta-model entity_mm = get_entity_mm() def javatype(s): """ Maps type names from SimpleType to Java. """ return { 'integer': 'int', 'string': 'String' }.get(s.name, s.name) # Create the output folder srcgen_folder = join(this_folder, 'srcgen') if not exists(srcgen_folder): mkdir(srcgen_folder) # Build a Person model from person.ent file person_model = entity_mm.model_from_file(join(this_folder, 'person.ent')) # Generate Java code for entity in person_model.entities: # For each entity generate java file with open(join(srcgen_folder, "%s.java" % entity.name.capitalize()), 'w') as f: f.write("%s.java" % entity.name) if __name__ == "__main__": main() ``` Would have to port my own templating engine, but I guess with python that would be super easy. Definitely interesting... More examples: [https://github.com/textX/textX/tree/master/examples](https://github.com/textX/textX/tree/master/examp Documentation: [http://textx.github.io/textX/stable/](http://textx.github.io/textX/stable/)

*Published by Thomas Trocha*
*Fri Apr 17 21:17:00 CEST 2020*
*http://thomas.trocha.com:80/pebble/textxtext-or-textx-lightweight-xtext-alternative-for-python*