pycropml Documentation

Release 0.1.1

Cyrille Ahmed Midingoyi

Oct 07, 2020

Contents

Contents .. _pycropml:

CHAPTER 1

CropML documentation

1.1 Module description

1.1.1 What is PyCrop2ML?

PyCrop2ML is a free, open-source library for defining and exchanging CropML models.It is used to generate components of modeling and simulation platforms from the CropML specification and allow component exchange between different platform.

It allows to parse the models described in CropML format and automatically generate the equivalent executable Python, java, C#, C++ components and packages usable from existing crop simulation platform.

1.1.2 What is Crop2ML ?

CropML is a XML-(JSON-)based language used to represent different biological processes involved in the crop models.

CropML project aims to provide common framework for defining and exchanging descriptions of crop growth models between crop simulation frameworks.

Objectives

Our main objectives are:

- define a **declarative language** to describe either an atomic model or a composition of models
- add semantic dimension to CropML language by annotation of the models to allow the composition of components of different platforms by using the standards of the semantic web
- develop a library to allow the transformation and the exchange of CropML model between different Crop modelling and simulation platform
- provide a **web repository** enabling registration, search and discovery of CropML Models

- facilitate Agricultural Model Exchange Initiative

Context

Nowadays, we observe the emergence of plant growth models which are built in different platforms. Although standard platform development initiatives are emerged, there is a lack of transparency, reusability, and exchange code between platforms due to the high diversity of modeling languages leading to a lack of benchmarking between the different platforms.
This project aims to gather developers and plant growth modelers to define a standard framework based on the development of declarative language and libraries to improve exchange model components between platforms.

Motivation

Our motivation is to:

- Strengthen the synergy between crop modelers, users and scientific researchers
- Facilitate model intercomparison (at the process level) and model improvement through the exchange of model components (algorithms) and code reuse between platforms/models.
- Bridge the gap between ecophysiologists who develop models at the process level with crop modelers and model users and facilitate the integration in crop models of new knowledge in plant science (i.e. we are seeking the exchange of knowledge rather than black box models).
- Increase capabilities and responsiveness to stakeholder' needs.
- Propose a solution to the AgMIP community for NexGen crop modeling tools.

Vision

- Facilitate the development of complex models
- Use modular modelling to share knowledge and rapidly develop operational tools.
- Reuse model parts to leverage the expertise of third parties;
- Renovate legacy code.
- Realize the benefit of sharing and complementing different expertise.
- Promote model sharing and reuse

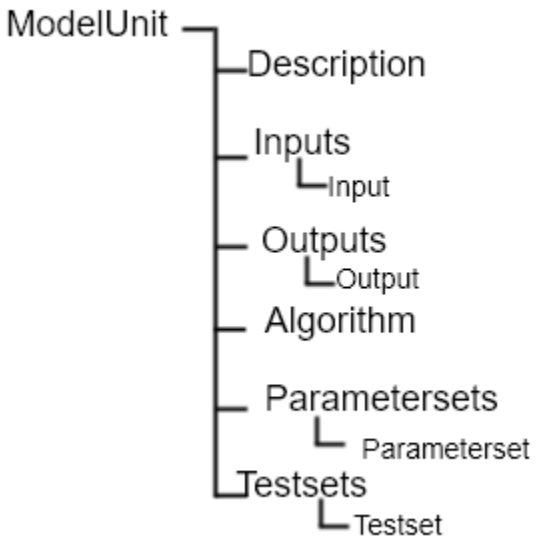
1.1.3 CropML Description

In CropML, a model is either a model unit or a composition of models. A ModelUnit represents the atomic unit of a crop model define by the modelers. A model composition is a model resulting from the composition of two or more atomic model or composite models.
These models have a specific formal definition in CropML.

Formal definition of a Model Unit in CropML

The structure of a Model Unit in CropML MUST be conform to a specific Document Type Definition named ModelUnit.dtd .

So a Model Unit CropML document is a XML document well-formed and also obeys the rules given in the ModelUnit structure.
This structure MAY be described by the below tree:



Element	Description
ModelUnit	The root of an atomic model in CropML which make the difference from a composite model.
Description	some basic information related to the name of the model, its authors and others elements used to reference it.
Inputs	A list of inputs characterized by their names, initial states, the range of values and others. Its input variables are related to climate, soil and cropping system
Outputs	A list of outputs defining the processes involved, the variables whose dynamics we want to observe.
Algorithm	The description of the behaviour of the model made by the mathematical relationship between the inputs and the outputs with some control structure.
Parametersets	Some sets of parameters which are invariant and used for the simulation of the models.
Testsets	Set of model configuration used to compare estimated and desired outputs .

In the next, we define the major elements of a CropML model unit.

ModelUnit element

An atomic model in CropML is declared with <<ModelUnit>> element, the usual root of CropML ModelUnit document.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ModelUnit PUBLIC "-//SIMPLACE/DTD SOL 1.0//EN" "https://raw.
↪githubusercontent.com/AgriculturalModelExchangeInitiative/xml_representation/master/
↪ModelUnit.dtd">
<ModelUnit modelid=" " timestep=" " name=" " version="">
        ....
</ModelUnit>
```

This element MUST contain a Description, an Algorithm, Parametersets and Testsets elements and MAY optionally have Inputs and Outputs elements. The restriction of the length of different lists is not imposed.

ModelUnit element MUST have an modelid and name attributes which are used to reference an atomic model. It MUST also contain a timestep attribute to define the temporality of the model and a version attribute for each version of the model.

Description element

This element gives the general information on the model and is composed by a set of character elements. It MUST contain Title, Authors, Institution and abstract elements and MAY optionally contain URI and Reference elements.

```
<ModelUnit modelid=" " timestep=" " name=" " version =" ">
        <Description>
                <Title>title</Title>
                <Authors>authors</Authors>
                <Institution>institution</Institution>
                <URI>uri</URI>
                <Abstract><![CDATA[abstract]]></Abstract>
        </Description>
        ...
</ModelUnit>
```

Inputs elements

The inputs of Model are listed inside an XML element called Inputs within a dictionary structure composed by their attributes which declarations are optional(default, max, min, parametercategory, variablecategory and uri) or required(name, datatype, description, inputtype, unit) and their corresponding value. *Inputs* element MUST contain one or more *Input* elements.

```
<ModelUnit modelid=" " timestep=" " name=" " version =" ">
   ...
   <Inputs>
      <Input name=" " description=" " parametercategory=" " datatype=" " min=" " max=
↪" " default=" " unit=" " uri="" inputtype=" "/>
```

(continues on next page)

(continued from previous page)

```
      <Input name=" " description=" " parametercategory=" " datatype=" " min=" " max=
↪" " default=" " unit=" " uri=" " inputtype=" "/>
      ...
   </Inputs>
   ...
</ModelUnit>
```

- The required *datatype* attribute is the type of input value specified in *default* (the default value in the input), *min* (the minimum value in the input) and *max* (the maximum value in the input). It MAY be one type of the set of types used in the existing crop modeling platform.
- The *inputtype* attribute makes it possible to distinguish the variables and the parameters of the model. So it MUST take one of two possible values: *parameter* and *variable*.
- The *parametercategory* attribute defines the category of parameter which is specified by one of the following values: *constant*, *species*, *soil* and *genotypic*.
- The *variablecategory* defines the category of variable depending on whether it is a *state*, a *rate* or an "auxiliary" variable. State variable characterize the behavior of the model and rate variable characterizes the changes in state variables.

Outputs element

The outputs of Model are listed inside an XML element called Outputs within a dictionary structure composed by their attributes which declarations are:

- optional(variabletype and URI)
- required(name, datatype, description, unit, max and min)
- and their corresponding value

Outputs MUST contain zero or more output elements.

```
<ModelUnit modelid=" " timestep=" " name=" " version =" ">
   ...
   <Outputs>
      <Output name=" " description=" " datatype=" " min=" " max=" "  unit=" " uri=" "/
↪>
      <Output name=" " description=" " datatype=" " min=" " max=" "  unit=" " uri=" "/
↪>
      ...
   </Outputs>
   ...
</ModelUnit>
```

The definition of different attributes is same as Input's attributes.

Algorithm element

The *Algorithm* element defines the building block of CropML model unit and shows the computational method to determine the outputs from the inputs.
It consists of a set of mathematical equations (relation between inputs), loops and conditional instructions which are well structured in a specific *language*, the algorithm's attribute.

```
<ModelUnit modelid=" " timestep=" " name=" " version =" ">
   ...
   <Algorithm language =""><![CDATA[
      ...
      ]]>
   </Algorithm>
   ...
</ModelUnit>
```

Parametersets element

Parametersets element contains one or more *Parameterset* elements that define the different ways of setting the model. Each *Parameterset* element MUST have *name* and *description* attributes that respectively represents the name and the description of each setting.

The different parameterset MUST contain a list of Param elements that show in attribute the name of the parameter (an input which inputtype equals *parameter*) and the fixed value of this one.

```
<ModelUnit modelid=" " timestep=" " name=" " version =" ">
    ...
    <Parametersets>
       <Parameterset name="" description="" uri = ""/>
       <Parameterset name="" description="" >
          <Param name="">value</Param>
          <Param name="">value</Param>
          ...
       </Parameterset>
       ...
    ...
 </ModelUnit>
```

Testsets element

Testsets element contains one or more *Testset* elements that define the different run for evaluating the outputs of the model.

Each *Testset* element MUST have *name*, *description* and *parameterset* attributes that respectively represents the name, the description of each run and the name of the parameterset related to the Testset. This one allow to retrieve the name and the value of different parameters includes in this parameterset.

The different Testset MUST contain a list of InputValue and OutputValue elements corresponding respectively to the values of inputs used in the run and the values of Outputs that will be asserted.

```
<ModelUnit modelid=" " timestep=" " name=" " version =" ">
   ...
   <Testsets>
```

(continues on next page)

(continued from previous page)

```
      <Testset name="" parameterset = "" description="" uri = ""/>
      <Testset name="" parameterset = "" description="" >
         <Test name="">
            <InputValue name="">value</InputValue>
            ...
            <OutputValue name="" precision ="">value</OutputValue>
            ...
         </Test>
         ...
      </Testset>
      ...
  </Testsets>
  ...
</ModelUnit>
```

Formal definition of a Composite Model in CropML

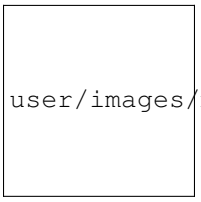
A Composite Model CropML is an assembly of processes which are described by a set of model units or a composition of models. Given a composite model is a model, this one has also inputs, outputs and internal state which describe the orchestration of different independent models composed.

The structure of a Composite Model in CropML MUST conform to a specific Document Type Definition named ModelComposition.dtd .

The composition is represented as a directed port graph of models:

Vertices are the different models that form the composition.
Ports are the inputs and outputs of each model.
Edges are directed and connect one output port to an input port of another model.

It contains in addition to all Elements of a model unit a Composition Element for the composition of models. This structure MAY be described by the below tree:



In the next, we define the major elements of a CropML model unit.

Inputs element

It MUST contain one or more *input* element which provide a set of independent models entries. If two or more input variables of independent models are the same (same unit, interval, description) a link should be made to one input variable of the composite model.

Outputs element

It MUST contain one or more *output* element which provide a set of independent models outputs or a result of a combination of models .

Composition element

It's a list of *models* elements which contains a list of *links* elements. Link provides the mechanism for mapping inputs declared within one modelUnit to output in another modelUnit, allowing information to be exchanged between the various atomic models in the composite model.

Algorithm element

The implementation differs from the platform:

- Discrete Events Models and Formalisms (RECORD)
- Actor model framework (OpenAlea)
- A sequence of algorithmic instructions witch implement the control flow (BIOMA)

1.1.4 PyCropML User Guide

Version 0.1.1

Release 0.1.1

Date Oct 07, 2020

This reference manual details functions, modules, and objects included in OpenAlea.Core, describing what they are and what they do. For a complete reference guide, see core_reference.

Warning: This "Reference Guide" is still very much in progress. Many aspects of OpenAlea.Core are not covered.

Manual

Note: The following examples assume you have installed the packages and setup your python path correctly.

Installation

```
conda install -c openalea pycropml
```

or

```
python setup.py install
```

Overview of the different classes

1.1.5 src

pycropml package

Subpackages

pycropml.interface package

Submodules

pycropml.interface.design module

pycropml.interface.version module

Maintain version for this package. Do not edit this file, use 'version' section of config.

`pycropml.interface.version.`**`MAJOR = 0`**
(int) Version major component.

`pycropml.interface.version.`**`MINOR = 0`**
(int) Version minor component.

`pycropml.interface.version.`**`POST = 2`**
(int) Version post or bugfix component.

Module contents

pycropml.transpiler package

Subpackages

pycropml.transpiler.generators package

Submodules

pycropml.transpiler.generators.checkGenerator module

class pycropml.transpiler.generators.checkGenerator.**CheckCompo**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.checkGenerator.CheckGenerator*

This class used to generates states, rates and auxiliary classes for C# languages.

class pycropml.transpiler.generators.checkGenerator.**CheckGenerator**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.pythonRules.PythonRules*

This class contains the specific properties of python language and use the NodeVisitor to generate a python code source from a well formed syntax tree.

visit_ExprStatNode(*node*)

visit_array(*node*)

visit_assignment(*node*)

visit_binary_op(*node*)

visit_bool(*node*)

visit_breakstatnode(*node*)

visit_call(*node*)

visit_comparison(*node*)

visit_cond_expr_node(*node*)

visit_continuestatnode(*node*)

visit_custom_call(*node*)

visit_datetime(*node*)

visit_declaration(*node*)

visit_dict(*node*)

visit_else_statement(*node*)

visit_elseif_statement(*node*)

visit_float(*node*)

visit_for_iterator(*node*)

visit_for_iterator_with_index(*node*)

visit_for_range_statement(*node*)

visit_for_sequence(*node*)

visit_for_sequence_with_index(*node*)

visit_for_statement(*node*)

visit_function_definition(*node*)

visit_if_statement(*node*)

visit_implicit_return(*node*)

visit_import(*node*)

visit_importfrom(*node*)

visit_index(*node*)

visit_list(*node*)

visit_local(*node*)

visit_method_call(*node*)

visit_module(*node*)

visit_notAnumber(*node*)

visit_pair(*node*)

visit_sliceindex(*node*)

visit_standard_call(*node*)

visit_standard_method_call(*node*)

visit_str(*node*)

visit_tuple(*node*)

visit_unary_op(*node*)

visit_while_statement(*node*)

pycropml.transpiler.generators.csharpGenerator module

class pycropml.transpiler.generators.csharpGenerator.**CsharpCompo**(*tree=None*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.csharpGenerator.CsharpTrans*, *pycropml.transpiler.generators.csharpGenerator.CsharpGenerator*

This class used to generates states, rates and auxiliary classes for C# languages.

assignParam()

constrWrap()

copyconstrWrap()

copyconstructor(*node*)

estimateWrap()

format()

get_mo(*varname*)

initCompo()

initWrap()

instanceModels()

loadParamWrap()

outputWrap()

privateWrap()

setCompo(*p*)

tranAssignParam()

visit_assignment(*node*)

visit_declaration(*node*)

visit_function_definition(*node*)

visit_local(*node*)

visit_module(*node*)

visit_return(*node*)

wrapper()

class pycropml.transpiler.generators.csharpGenerator.**CsharpGenerator**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.csharpRules.CsharpRules*

This class contains the specific properties of Csharp language and use the NodeVisitor to generate a csharp code source from a well formed syntax tree.

add_features(*node*)

internal_declaration(*node*)

retrieve_params(*node*)

transform_return(*node*)

visit_array(*node*)

visit_array_decl(*node*)

visit_assignment(*node*)

visit_binary_op(*node*)

visit_bool(*node*)

visit_bool_decl(*node*)

visit_breakstatnode(*node*)

visit_call(*node*)

visit_comparison(*node*)

visit_cond_expr_node(*node*)

visit_continuestatnode(*node*)

visit_custom_call(*node*)
TODO

visit_datetime(*node*)

visit_datetime_decl(*node*)

visit_decl(*node*)

visit_declaration(*node*)

visit_dict(*node*)

visit_dict_decl(*node*)

visit_else_statement(*node*)

visit_elseif_statement(*node*)

visit_float(*node*)

visit_float_decl(*node*)

visit_for_iterator(*node*)

visit_for_iterator_with_index(*node*)

visit_for_range_statement(*node*)

visit_for_sequence_with_index(*node*)
TODO

visit_for_statement(*node*)

visit_function_definition(*node*)

visit_if_statement(*node*)

visit_implicit_return(*node*)

visit_import(*node*)

visit_importfrom(*node*)

visit_index(*node*)

visit_int_decl(*node*)

visit_list(*node*)

visit_list_decl(*node*)

visit_method_call(*node*)

visit_module(*node*)

visit_notAnumber(*node*)

visit_pair(*node*)

visit_print()

visit_return(*node*)

visit_sliceindex(*node*)

visit_standard_call(*node*)

visit_standard_method_call(*node*)

visit_str(*node*)

visit_str_decl(*node*)

visit_tuple(*node*)

visit_tuple_decl(*node*)

visit_unary_op(*node*)

visit_while_statement(*node*)

class pycropml.transpiler.generators.csharpGenerator.**CsharpTrans**(*models*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.csharpRules.CsharpRules*

This class used to generates states, rates and auxiliary classes for C# languages.

DATATYPE = {'BOOLEAN': 'bool', 'DATE': 'DateTime', 'DATEARRAY': ['array', 'DateTime'],

copyconstructor(*node*)

generate(*nodes*, *typ*)

getset(*node*, *wrap=False*)

model2Node()

private(*node*)

visit_DateTime(*node*)

visit_array_decl(*node*)

visit_bool_decl(*node*)

visit_datetime_decl(*node*)

visit_decl(*node*)

visit_dict_decl(*node*)

visit_float_decl(*node*)

visit_int_decl(*node*)

visit_list_decl(*node*)

visit_str_decl(*node*)

visit_tuple_decl(*node*)

pycropml.transpiler.generators.csharpGenerator.**to_struct_cs**(*models*, *rep*, *name*)

pycropml.transpiler.generators.csharpGenerator.**to_wrapper_cs**(*models*, *rep*, *name*)

pycropml.transpiler.generators.docGenerator module

class pycropml.transpiler.generators.docGenerator.**DocGenerator**(*model=None*, *tag='#'*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*

Generate doc in different target language - desccription of the code - Inputs details - Outputs details

comment(*doc*)

doc(*x*, *name*)

generate_desc(*model*)

generate_header(*model*)

pycropml.transpiler.generators.fortranGenerator module

class pycropml.transpiler.generators.fortranGenerator.**FortranCompo**(*tree=None*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.fortranGenerator.FortranGenerator*

This class used to generates states, rates and auxiliary classes for Fortran90 language.

visit_importfrom(*node*)

self.newline(node) self.write('Use ') for idx, item in enumerate(node.name):

if idx: self.write(', ')

self.write("%smod"%item.split("**model_**")[1].capitalize())

class pycropml.transpiler.generators.fortranGenerator.**FortranGenerator**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.fortranRules.FortranRules*

This class contains the specific properties of fortran language and use the NodeVisitor to generate a fortran code source from a well formed syntax tree.

add_features(*node*)

binop_precedence = {'!=': 4, '%': 11, '&': 7, '*': 11, '': 12, '+': 9, '-': 9**

body(*statements*)

checkIndex(*node*)

doc = None

get constant parameters in models if inp.inputtype=="parameter":

#print(inp.name, model.name) if inp.parametercategory=="constant":

self.mod_parameters.append(inp.name)

Type for inp in self.model.inputs

internal_declaration(*node*)

part_declaration(*node*)

retrieve_params(*node*)

transform_return(*node*)

unop_precedence = {'!': 3, '+': 10, '-': 10, 'not': 3, '~': 10}

visit_ExprStatNode(*node*)

visit_array_decl(*node*)

visit_assignment(*node*)

visit_binary_op(*node*)

visit_bool(*node*)

visit_bool_decl(*node*)

visit_breakstatnode(*node*)

visit_call(*node*)

visit_comparison(*node*)

visit_cond_expr_node(*node*)

visit_continuestatnode(*node*)

visit_custom_call(*node*)

visit_datetime(*node*)

visit_datetime_decl(*node*)

visit_decl(*nodeT*)

visit_declaration(*node*)

visit_else_statement(*node*)

visit_elseif_statement(*node*)

visit_float(*node*)

visit_float_decl(*node*)

visit_for_iterator(*node*)

visit_for_iterator_with_index(*node*)

visit_for_range_statement(*node*)

visit_for_sequence(*node*)

visit_for_statement(*node*)

visit_function_definition(*node*)

visit_if_statement(*node*)

visit_implicit_return(*node*)

visit_import(*node*)

visit_importfrom(*node*)
self.newline(node) self.write('from %s import ' % (node.namespace)) for idx, item in enumerate(node.name):

if idx: self.write(', ')

self.write(item)

visit_index(*node*)

visit_int(*node*)

visit_int_decl(*node*)

visit_list(*node*)

visit_list_decl(*node*)

visit_method_call(*node*)

visit_module(*node*)

visit_notAnumber(*node*)

visit_pair(*node*)

visit_sliceindex(*node*)

visit_standard_call(*node*)

visit_standard_method_call(*node*)

visit_str(*node*)

visit_str_decl(*node*)

visit_subroutine(*node*)

visit_tab(*node*)

visit_unary_op(*node*)

visit_while_statement(*node*)

pycropml.transpiler.generators.fortranGenerator.**checkList**(*list1*, *list2*)

pycropml.transpiler.generators.fortranGenerator.**valParam**(*model*, *name*)

pycropml.transpiler.generators.javaGenerator module

class pycropml.transpiler.generators.javaGenerator.**JavaCompo**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.javaGenerator.JavaTrans*, *pycropml.transpiler.generators.javaGenerator.JavaGenerator*

This class used to generates states, rates and auxiliary classes for java language.

copyconstructor(*node*)

get_mo(*varname*)

initCompo()

instanceModels()

setCompo(*p*)

visit_assignment(*node*)

visit_declaration(*node*)

visit_function_definition(*node*)

visit_implicit_return(*node*)

visit_module(*node*)

visit_return(*node*)

class pycropml.transpiler.generators.javaGenerator.**JavaGenerator**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.javaRules.JavaRules*

This class contains the specific properties of Java language and use the NodeVisitor to generate a java code source from a well formed syntax tree.

add_features(*node*)

gettype(*arg*)

internal_declaration(*node*)

retrieve_params(*node*)

transform_return(*node*)

visit_array(*node*)

visit_array_decl(*node*)

visit_assignment(*node*)

visit_binary_op(*node*)

visit_bool(*node*)

visit_bool_decl(*node*)

visit_breakstatnode(*node*)

visit_call(*node*)

visit_comparison(*node*)

visit_cond_expr_node(*node*)

visit_continuestatnode(*node*)

visit_custom_call(*node*)
TODO

visit_datetime_decl(*node*)

visit_decl(*node*)

visit_declaration(*node*)

visit_dict(*node*)

visit_dict_decl(*node*)

visit_else_statement(*node*)

visit_elseif_statement(*node*)

visit_float(*node*)

visit_float_decl(*node*)

visit_for_iterator(*node*)

visit_for_iterator_with_index(*node*)

visit_for_range_statement(*node*)

visit_for_sequence_with_index(*node*)

visit_for_statement(*node*)

visit_function_definition(*node*)

visit_if_statement(*node*)

visit_implicit_return(*node*)

visit_import(*node*)

visit_importfrom(*node*)

visit_index(*node*)

visit_int_decl(*node*)

visit_list(*node*)

visit_list_decl(*node*)

visit_method_call(*node*)

visit_module(*node*)

visit_notAnumber(*node*)

visit_pair(*node*)

visit_print()

visit_return(*node*)

visit_sliceindex(*node*)

visit_standard_call(*node*)

visit_standard_method_call(*node*)

visit_str(*node*)

visit_str_decl(*node*)

visit_tuple(*node*)

visit_tuple_decl(*node*)

visit_unary_op(*node*)

visit_while_statement(*node*)

class pycropml.transpiler.generators.javaGenerator.**JavaTrans**(*models*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.javaRules.JavaRules*

This class used to generates states, rates and auxiliary classes for java language.

DATATYPE = {'BOOLEAN': 'bool', 'DATE': 'datetime', 'DATEARRAY': ['array', 'datetime'],

access(*node*)

copyconstructor(*node*)

generate(*nodes*, *typ*)

getset(*node*)

gettype(*arg*)

model2Node()

private(*node*)

visit_array_decl(*node*)

visit_bool_decl(*node*)

visit_datetime_decl(*node*)

visit_decl(*node*)

visit_dict_decl(*node*)

visit_float_decl(*node*)

visit_int_decl(*node*)

visit_list_decl(*node*)

visit_str_decl(*node*)

visit_tuple_decl(*node*)

pycropml.transpiler.generators.javaGenerator.**to_struct_java**(*models*, *rep*, *name*)

pycropml.transpiler.generators.openaleaGenerator module

class pycropml.transpiler.generators.openaleaGenerator.**OpenaleaCompo**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.pythonGenerator.PythonCompo*

This class used to generates states, rates and auxiliary classes for C# languages.

generate_factory(*model*)
Create a Node Factory from CropML model unit.

generate_wralea(*mc*)
Generate wralea factories from the meta-information of the the model units.

class pycropml.transpiler.generators.openaleaGenerator.**OpenaleaGenerator**(*tree=None*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.pythonGenerator.PythonGenerator*

This class contains the specific properties of OpenAlea and use the NodeVisitor to generate a csharp code source from a well formed syntax tree.

pycropml.transpiler.generators.openaleaGenerator.**openalea_interface**(*inout*)

pycropml.transpiler.generators.openaleaGenerator.**signature**(*model*)

pycropml.transpiler.generators.pythonGenerator module

class pycropml.transpiler.generators.pythonGenerator.**PythonCompo**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.pythonGenerator.PythonGenerator*

This class used to generates states, rates and auxiliary classes for C# languages.

class pycropml.transpiler.generators.pythonGenerator.**PythonGenerator**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.codeGenerator.CodeGenerator*, *pycropml.transpiler.rules.pythonRules.PythonRules*

This class contains the specific properties of python language and use the NodeVisitor to generate a python code source from a well formed syntax tree.

comment(*doc*)

visit_ExprStatNode(*node*)

visit_array(*node*)

visit_assignment(*node*)

visit_binary_op(*node*)

visit_bool(*node*)

visit_breakstatnode(*node*)

visit_call(*node*)

visit_comparison(*node*)

visit_cond_expr_node(*node*)

visit_continuestatnode(*node*)

visit_custom_call(*node*)

visit_datetime(*node*)

visit_declaration(*node*)

visit_dict(*node*)

visit_else_statement(*node*)

visit_elseif_statement(*node*)

visit_float(*node*)

visit_for_iterator(*node*)

visit_for_iterator_with_index(*node*)

visit_for_range_statement(*node*)

visit_for_sequence(*node*)

visit_for_sequence_with_index(*node*)

visit_for_statement(*node*)

visit_function_definition(*node*)

visit_if_statement(*node*)

visit_implicit_return(*node*)

visit_import(*node*)

visit_importfrom(*node*)

visit_index(*node*)

visit_list(*node*)

visit_method_call(*node*)

visit_module(*node*)

visit_notAnumber(*node*)

visit_pair(*node*)

visit_sliceindex(*node*)

visit_standard_call(*node*)

visit_standard_method_call(*node*)

visit_str(*node*)

visit_tuple(*node*)

visit_unary_op(*node*)

visit_while_statement(*node*)

pycropml.transpiler.generators.recordGenerator module

<vle_project version="1.1.x" date="2012-Oct-03 13:01:13" author="Eric Casellas">

<structures>

<model width="2184" height="1280" name="2CV_parcelle" type="coupled">

<submodels> <model observables="vueDecision" conditions="condDecFSA" dynamics="dynDecFSA" debug="false" width="100" height="75" x="132" y="26" name="Decision" type="atomic"></model> <model width="100" height="165" x="316" y="127" name="2CV" type="coupled"></model>

</submodels> <connections> </connections>

</model>

</structures> <dynamics> </dynamics> <experiment name="2CV_parcelle"> </experiment>

</vle_project>

pycropml.transpiler.generators.simplaceGenerator module

class pycropml.transpiler.generators.simplaceGenerator.**SimplaceCompo**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.javaGenerator.JavaGenerator*

class pycropml.transpiler.generators.simplaceGenerator.**SimplaceGenerator**(*tree*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.javaGenerator.JavaGenerator*

visit_declaration(*node*)

visit_function_definition(*node*)

visit_import(*node*)

visit_local(*node*)

visit_module(*node*)

visit_return(*node*)

pycropml.transpiler.generators.siriusGenerator module

class pycropml.transpiler.generators.siriusGenerator.**SiriusCompo**(*tree=None*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.csharpGenerator.CsharpCompo*

This class used to generates states, rates and auxiliary classes for C# languages.

constrWrap()

copyconstrWrap()

visit_module(*node*)

wrapper()

class pycropml.transpiler.generators.siriusGenerator.**SiriusGenerator**(*tree=None*, *model=None*, *name=None*)

Bases: *pycropml.transpiler.generators.csharpGenerator.CsharpGenerator*

This class contains the specific properties of Csharp language and use the NodeVisitor to generate a csharp code source from a well formed syntax tree.

class pycropml.transpiler.generators.siriusGenerator.**SiriusTrans**(*models*)

Bases: *pycropml.transpiler.generators.csharpGenerator.CsharpTrans*

This class used to generates states, rates and auxiliary classes for Sirius.

pycropml.transpiler.generators.siriusGenerator.**to_struct_sirius**(*models*, *rep*, *name*)

pycropml.transpiler.generators.siriusGenerator.**to_wrapper_sirius**(*models*, *rep*, *name*)

Module contents

pycropml.transpiler.lib package

Module contents

pycropml.transpiler.rules package

Submodules

pycropml.transpiler.rules.csharpRules module

class pycropml.transpiler.rules.csharpRules.**CsharpRules**

Bases: *pycropml.transpiler.rules.generalRule.GeneralRule*

binary_op = {'!=': '!=', '*': '*', '+': '+', '-': '-', '/': '/', '<': '<', '<='

constructor = '\n public %s()\n {\n \n }\n '

copy_constr = '\n public %s(%s toCopy, bool copyAll) // copy constructor \n {\n if (co

copy_constrArray = '\n for (int i = 0; i < %s; i++)\n {\n _%s[i] = toCopy._%s[i];\n }

copy_constrList = '\n for (int i = 0; i < toCopy.%s.Count; i++)\n {\n %s.Add(toCopy.%s

copy_constr_compo = '\n public %s(%s toCopy): this() // copy constructor \n {\n'

functions = {'datetime': {'datetime': ' new DateTime'}, 'io': {'print': 'Display'

methods = {'array': {'append': '.Add', 'len': <function translateLenArray>}, 'dict'

public_properties = '\n {\n get\n {\n return this._%s;\n }\n set\n {\n this._%s= value

public_properties_compo = '\n {\n get\n {\n return _%s.%s;\n }\n set\n {\n %s\n } \n }

public_properties_wrap = '{ get { return %s.%s;}} \n '

types = {'DateTime': 'DateTime', 'array': '%s[] %s= new %s', 'bool': 'bool', 'datet

unary_op = {'+': '+', '-': '-', 'not': '!', '~': '~'}

pycropml.transpiler.rules.csharpRules.**translateLenArray**(*node*)

pycropml.transpiler.rules.csharpRules.**translateLenDict**(*node*)

pycropml.transpiler.rules.csharpRules.**translateLenList**(*node*)

pycropml.transpiler.rules.csharpRules.**translateNotContains**(*node*)

pycropml.transpiler.rules.csharpRules.**translateSum**(*node*)

pycropml.transpiler.rules.csharpRules.**translateget**(*node*)

pycropml.transpiler.rules.csharpRules.**translatekeyDict**(*node*)

pycropml.transpiler.rules.fortranRules module

class pycropml.transpiler.rules.fortranRules.**FortranRules**

Bases: *pycropml.transpiler.rules.generalRule.GeneralRule*

binary_op = {'!=': '.NE.', '*': '*', '': '**', '+': '+', '-': '-', '/': '/',**

functions = {'datetime': {'datetime': <function FortranRules.<lambda>>}, 'math': {'

method()

methods = {'array': {'append': <function FortranRules.<lambda>>, 'len': 'SIZE'}, 'd

dependencies = {

'list': { 'index': 'list_sub', 'append': 'list_sub'

}

}

types = {'array': '%s, DIMENSION(%s)', 'bool': 'LOGICAL', 'datetime': 'CHARACTER(65

unary_op = {'+': '+', '-': '-', 'not': '.NOT. ', '~': '~'}

pycropml.transpiler.rules.fortranRules.**argsToStr**(*args*)

pycropml.transpiler.rules.fortranRules.**translateAppend**(*node*)

pycropml.transpiler.rules.fortranRules.**translateCeil**(*node*)

pycropml.transpiler.rules.fortranRules.**translateContains**(*node*)

pycropml.transpiler.rules.fortranRules.**translateFind**(*node*)

pycropml.transpiler.rules.fortranRules.**translateIndex**(*node*)

pycropml.transpiler.rules.fortranRules.**translateNotContains**(*node*)

pycropml.transpiler.rules.fortranRules.**translatePop**(*node*)

pycropml.transpiler.rules.fortranRules.**translatePow**(*node*)

pycropml.transpiler.rules.generalRule module

class pycropml.transpiler.rules.generalRule.**GeneralRule**

Bases: object

” Abstract class of Rules

pycropml.transpiler.rules.javaRules module

class pycropml.transpiler.rules.javaRules.**JavaRules**
Bases: *pycropml.transpiler.rules.generalRule.GeneralRule*

binary_op = {'!=': '!=', '*': '*', '+': '+', '-': '-', '/': '/', '<': '<', '<='

constructor = '\n public %s()\n {\n \n }'

copy_constr = '\n public %s(%s toCopy, boolean copyAll) // copy constructor \n {\n if

copy_constrArray = '\n for (int i = 0; i < %s; i++)\n {\n _%s[i] = toCopy._%s[i];\n }'

copy_constrList = '\n for (%s c : toCopy.%s)\n {\n _%s.add(c);\n }\n this.%s = _%s;'

copy_constr_compo = '\n public %s(%s toCopy) // copy constructor \n {'

functions = {'datetime': {'datetime': 'format.parse'}, 'math': {'acos': 'Math.acos

get_properties = '\n {\n return %s;\n }'

get_properties_compo = '\n {\n return _%s.get%s();\n }'

methods = {'array': {'append': '.add', 'len': <function translateLenArray>}, 'dict'

set_properties = '\n {\n this.%s= _%s;\n } \n '

set_properties_compo = '\n {\n %s\n } '

types = {'array': '%s[] %s= new %s', 'bool': 'boolean', 'datetime': 'Date', 'dict':

types2 = {'Date': 'Date', 'bool': 'Boolean', 'datetime': 'Date', 'float': 'Double'

unary_op = {'+': '+', '-': '-', 'not': '!', '~': '~'}

pycropml.transpiler.rules.javaRules.**argsToStr**(*args*)

pycropml.transpiler.rules.javaRules.**trans_format_parse**(*node*)

pycropml.transpiler.rules.javaRules.**translateDictkeys**(*node*)

pycropml.transpiler.rules.javaRules.**translateLenArray**(*node*)

pycropml.transpiler.rules.javaRules.**translateLenDict**(*node*)

pycropml.transpiler.rules.javaRules.**translateLenList**(*node*)

pycropml.transpiler.rules.javaRules.**translateNotContains**(*node*)

pycropml.transpiler.rules.javaRules.**translateSum**(*node*)

pycropml.transpiler.rules.pythonRules module

class pycropml.transpiler.rules.pythonRules.**PythonRules**
Bases: *pycropml.transpiler.rules.generalRule.GeneralRule*

binary_op = {'!=': '!=', '*': '*', '+': '+', '-': '-', '/': '/', '<': '<', '<='

functions = {'datetime': {'datetime': 'datetime'}, 'math': {'acos': 'acos', 'asin'

methods = {'array': {'len': 'len'}, 'datetime': {'datetime': 'datetime', 'day': '

types = {'bool': 'bool', 'datetime': 'datetime', 'dict': 'dict', 'float': 'float',

unary_op = {'+': '+', '-': '-', 'not': 'not ', '~': '~'}

pycropml.transpiler.rules.pythonRules.**translateDictkeys**(*node*)

pycropml.transpiler.rules.pythonRules.**translateNotContains**(*node*)

pycropml.transpiler.rules.sqRules module

class pycropml.transpiler.rules.sqRules.**SqRules**
Bases: *pycropml.transpiler.rules.generalRule.GeneralRule*

field_decl(*node*)

header = '--

method()

methods = {'dict': {'len': 'SIZE'}, 'float': {'int': 'INT'}, 'int': {'float': 'R

namespace = {'headNamespace': '\nnamespace SiriusQualityEnergyBalance\n{\n using Syst

Module contents

Submodules

pycropml.transpiler.Parser module

class pycropml.transpiler.Parser.**opt**(***kwds*)
Bases: object

pycropml.transpiler.Parser.**parser**(*module*)
Read, parse a Cython code and generate an abstract syntaxique tree.

Context: Compilation context: contains every pxd ever loaded, path information and the data related to the compilation. Class where it is implemented Cython parse method.

options: To set Compilation Options as language_level: The source language level to use, formal_grammar: to define if it will be used to Parse the file evaluate_tree_assertions: To evaluate parse tree show_version : To display version number use_listing_file: Generate a .lis file errors_to_stderr: Echo errors to stderr when using .lis include_path: Directories to search for include files output_file: Name of generated .c file generate_pxi: Generate .pxi file for public declarations capi_reexport_cincludes: Add cincluded headers to any auto-generated header files. timestamps: Only compile changed source files verbose : Always print source names being compiled compiler_directives: Overrides for pragma options (see Options.py) embedded_metadata: Metadata to embed in the C file as json. evaluate_tree_assertions: Test support: evaluate parse tree assertions cplus : Compile as c++ code

Here default options were used except language level

Scanning.FileSourceDescriptor: Represents a code source. Only file sources for Cython code supported

pycropml.transpiler.api_transform module

class pycropml.transpiler.api_transform.**Standard**
Bases: object

Standard classes should respond to expand and to return valid nodes on expand

class pycropml.transpiler.api_transform.**StandardCall**(*namespace*, *function*, *expander=None*)
Bases: *pycropml.transpiler.api_transform.Standard*

converts to a standard call of the given namespace and function

expand(*args*)

class pycropml.transpiler.api_transform.**StandardCallAttrib**(*namespace*, *function*, *expander=None*)

Bases: *pycropml.transpiler.api_transform.Standard*

converts to a standard call of the given namespace and function

expand(*args=[]*)

class pycropml.transpiler.api_transform.**StandardMethodCall**(*type*, *message*, *default=None*, *expander=None*)

Bases: *pycropml.transpiler.api_transform.Standard*

converts to a method call of the same class

expand(*args*)

class pycropml.transpiler.api_transform.**StandardSwapper**(*type*, *message*)

Bases: *pycropml.transpiler.api_transform.Standard*

expand(*args*)

pycropml.transpiler.api_transform.**abs_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**datetime_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**float_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**int_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**len_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**max_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**min_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**modulo_expander**(*type*, *message*, *args*)

pycropml.transpiler.api_transform.**pow_expander**(*type*, *message*, *args*)

pycropml.transpiler.ast_transform module

class pycropml.transpiler.ast_transform.**AstTransformer**(*tree*, *code*, *model=None*)

Bases: object

assert_translatable(*node*, ***pairs*)

newtype()

notdeclared(*name*, *line*)

retrieve_library(*func*)

transformer()

visit_addnode(*node*, *operand1*, *operand2*, *location*)

visit_attributenode(*node*, *obj*, *location*)

visit_binopnode(*node*, *operand1*, *operand2*, *location*)

visit_boolbinopnode(*node*, *operand1*, *operand2*, *location*)

visit_boolnode(*node*, *location*)

visit_breakstatnode(*node*, *location*)

visit_cargdeclnode(*node*, *base_type*, *declarator*, *default*, *annotation*, *location*)

visit_condexprnode(*node*, *test*, *true_val*, *false_val*, *location*)

visit_continuestatnode(*node*, *location*)

visit_csimplebasetypenode(*node*, *location*)

visit_cvardefnode(*node*, *base_type*, *declarators*, *location*)

visit_definitions()

visit_defnode(*node*, *args*, *star_arg*, *starstar_arg*, *decorators*, *body*, *return_type_annotation*, *location*)

visit_dictnode(*node*, *key_value_pairs*, *location*)

visit_divnode(*node*, *operand1*, *operand2*, *location*)

visit_elements(*elements*, *kind*, *homogeneous=True*)

visit_exprstatnode(*node*, *expr*, *location*)

visit_floatnode(*node*, *location*)
if float(node.value) < 0.0: return { 'type': 'unary_op',

'operator': '-', 'value': str(-float(node.value)), 'pseudo_type': "float" }

visit_forinstatnode(*node*, *target*, *iterator*, *item*, *body*, *else_clause*, *location*)

visit_ifclausenode(*node*, *body*, *condition*, *location*)

visit_ifstatnode(*node*, *if_clauses*, *else_clause*, *location*)

visit_indexnode(*node*, *base*, *index*, *location*)

visit_inplaceassignmentnode(*node*, *lhs*, *rhs*, *location*)

visit_intnode(*node*, *location*)

visit_listnode(*node*, *args*, *mult_factor*, *location*)

visit_modnode(*node*, *operand1*, *operand2*, *location*)

visit_mulnode(*node*, *operand1*, *operand2*, *location*)

visit_namenode(*node*, *location*)

visit_node(*node*)

visit_notnode(*node*, *operand*, *location*)

visit_pownode(*node*, *operand1*, *operand2*, *location*)

visit_primarycmpnode(*node*, *operand1*, *operand2*, *coerced_operand2*, *cascade*, *location*)

visit_printstatnode(*node*, *arg_tuple*, *stream*, *location*)

visit_pyclassdefnode(*node*, *location*)

visit_returnstatnode(*node*, *value*, *location*)

visit_simplecallnode(*node*, *function*, *coerced_self*, *args*, *arg_tuple*, *location*)

visit_singleassignmentnode(*node*, *lhs*, *rhs*, *location*)

visit_sliceindexnode(*node*, *start*, *stop*, *base*, *slice*, *location*)

visit_statlistnode(*node*, *stats*, *location*)

visit_stringnode(*node*, *location*)

visit_subnode(*node*, *operand1*, *operand2*, *location*)

visit_top_level(*nodes*)

visit_tuplenode(*node*, *args*, *mult_factor*, *location*)

visit_unaryminusnode(*node*, *operand*, *location*)

visit_unaryplusnode(*node*, *operand*, *location*)

visit_unicodenode(*node*, *location*)

visit_whilestatnode(*node*, *condition*, *body*, *else_clause*, *location*)

pycropml.transpiler.ast_transform.**transform_to_syntax_tree**(*tree*)
Generate a Node class from the tree in dict format

pycropml.transpiler.builtin_typed_api module

pycropml.transpiler.builtin_typed_api.**add**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**and_**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**arg_check**(*expected_type*, *args*, *a*)

pycropml.transpiler.builtin_typed_api.**binary_and**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**binary_or**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**builtin_type_check**(*namespace*, *function*, *receiver*, *args*)

pycropml.transpiler.builtin_typed_api.**div**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**mod**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**mul**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**or_**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**pow_**(*l*, *r*)

pycropml.transpiler.builtin_typed_api.**simplify**(*kind*, *generics*)

pycropml.transpiler.builtin_typed_api.**sub**(*l*, *r*)

pycropml.transpiler.checkingModel module

class pycropml.transpiler.checkingModel.**Checking**
Bases: object

Module used to check units validity in model equation based on model xml files. This checking can also use for python code with metadata

pycropml.transpiler.codeGenerator module

class pycropml.transpiler.codeGenerator.**CodeGenerator**(*add_line_information=False*)

Bases: *pycropml.transpiler.nodeVisitor.NodeVisitor*

binop_precedence = {'!=': 4, '%': 10, '&': 7, '*': 10, '': 12, '+': 9, '-': 9**

body(*statements*)

body_or_else(*node*)

comma_separated_list(*items*)

emit_sequence(*node*, *parens=(", ")*)

emit_string(*node*, *prefix=''*)

newline(*node=None*, *extra=0*)

operator_enter(*new_prec*)

operator_exit()

safe_double(*node*)

unop_precedence = {'!': 3, '+': 11, '-': 11, 'not': 3, '~': 11}

visit_ExprStatNode(*node*)

visit_array(*node*)

visit_for_sequence(*node*)

visit_int(*node*)

visit_local(*node*)

visit_simpleCall(*node*)

write(*x*)

pycropml.transpiler.env module

class pycropml.transpiler.env.**Env**(*values=None*, *parent=None*)

Bases: object

child_env(*values=None*)

pycropml.transpiler.errors module

exception pycropml.transpiler.errors.**PseudoCythonNotTranslatableError**(*message*, *suggestions=None*, *right=None*, *wrong=None*)

Bases: *pycropml.transpiler.errors.PseudoError*

exception pycropml.transpiler.errors.**PseudoCythonTypeCheckError**(*message*, *suggestions=None*, *right=None*, *wrong=None*)

Bases: *pycropml.transpiler.errors.PseudoError*

exception pycropml.transpiler.errors.**PseudoError**(*message*, *suggestions=None*, *right=None*, *wrong=None*)

Bases: Exception

pycropml.transpiler.errors.**beautiful_error**(*exception*)

pycropml.transpiler.errors.**cant_infer_error**(*name*, *line*)

pycropml.transpiler.errors.**tab_aware**(*location*, *code*)

if tabs in beginning of code, add tabs for them, otherwise spaces

pycropml.transpiler.errors.**translation_error**(*data*, *location=None*, *code=None*, *wrong_type=None*, ***options*)

pycropml.transpiler.errors.**type_check_error**(*data*, *location=None*, *code=None*, *wrong_type=None*, ***options*)

pycropml.transpiler.helpers module

pycropml.transpiler.helpers.**prepare_table**(*types*, *original_methods=None*)

pycropml.transpiler.helpers.**safe_serialize_type**(*l*)

serialize only with letters, numbers and _

pycropml.transpiler.helpers.**serialize_type**(*l*)

pycropml.transpiler.interface module

class pycropml.transpiler.interface.**TreeInterface**(*tree*)

Bases: object

visits recursively nodes of the tree with defined transform_<node_type> methods and transforms in place

transform(*tree*, *in_block=False*)

transform_block(*tree*)

transform_default(*tree*)

class pycropml.transpiler.interface.**middleware**(*tree*)

Bases: *pycropml.transpiler.interface.TreeInterface*

api_translate()

pycropml.transpiler.main module

class pycropml.transpiler.main.**Main**(*file*, *language*, *models=None*, *name=None*)

Bases: object

parse()

to_ast(*source*)

to_source()

translate()

pycropml.transpiler.main.**formater**(*code*)

pycropml.transpiler.main.**formaterNext**(*line*)

pycropml.transpiler.nodeVisitor module

class pycropml.transpiler.nodeVisitor.**NodeVisitor**
Bases: object

Define a method which browse the graph and call a methode constructed from the type of each node of the graph

visit(*node*)

pycropml.transpiler.pseudo_tree module

class pycropml.transpiler.pseudo_tree.**Node**(*type*, ***fields*)
Bases: object

The new Node generated with specific properties. These properties are automatically set"

Example: Node(type='local', name='l', pseudo_type="int") to represent a int variable declaration

y

pycropml.transpiler.version module

Maintain version for this package. Do not edit this file, use 'version' section of config.

pycropml.transpiler.version.**MAJOR = 0**
(int) Version major component.

pycropml.transpiler.version.**MINOR = 0**
(int) Version minor component.

pycropml.transpiler.version.**POST = 2**
(int) Version post or bugfix component.

Module contents

Submodules

pycropml.algorithm module

class pycropml.algorithm.**Algorithm**(*language*, *development*, *platform*, *filename=None*)
Bases: object

pycropml.checking module

class pycropml.checking.**Test**(*name*)
Bases: *pycropml.checking.Testset*

class pycropml.checking.**Testset**(*name*, *parameterset*, *description*, *uri=None*)
Bases: object

Test

pycropml.checking.**testset**(*model*, *name*, *kwds*)

pycropml.code2nbk module

License, Header

Use pkglts

Generate notebook from code source

class pycropml.code2nbk.**Model2Nb**(*model*, *code*, *name*, *dir=None*)
Bases: object

Generate a Jupyter Notebook from a set of models.

generate_nb(*language*, *tg_rep*, *namep*)

nb = {'cells': [], 'metadata': {}, 'nbformat': 4, 'nbformat_minor': 4}

pycropml.composition module

Read xml representation of a model composite

class pycropml.composition.**Description**
Bases: object

Model Composition Description.

A description is defined by:

- Title
- Authors
- Institution
- Reference
- Abstract

class pycropml.composition.**ModelComposition**(*kwds*)
Bases: *pycropml.composition.ModelDefinition*

Formal description of a Model Composite.

add_description(*description*)
TODO

class pycropml.composition.**ModelDefinition**(*kwds*)
Bases: object

Model name, id, version and step

class pycropml.composition.**ModelParser**
Bases: *pycropml.composition.Parser*

Read an XML file and transform it in our object model.

Composition(*elts*)

Description(*Title*, *Author*, *Institution*, *Reference*, *Abstract*)

Initialization(*elt*)

Links(*elt*)
Retrieve different types of links

Model(*elt*)
Models

ModelComposition(*elts*)
ModelComposition (Description, Models, Inputlink,Outputlink,externallink)

dispatch(*elt*)

parse(*fn*)

class pycropml.composition.**Models**(*name*, *modelid*, *file*, *package_name=None*)
Bases: *pycropml.composition.ModelComposition*, pycropml.modelunit.ModelUnit

class pycropml.composition.**Parser**
Bases: object

Read an XML file and transform it in our object model.

dispatch(*elt*)

parse(*fn*)

pycropml.composition.**model_parser**(*fn*)
Parse a composite model and return the model.

Returns ModelComposite object of the CropML Model.

pycropml.composition.**retrieve_path**(*fn*)

pycropml.cyml module

Created on Tue Mar 19 22:59:23 2019

@author: midingoy

pycropml.cyml.**ext**(*language*)

pycropml.cyml.**prefix**(*model*)

pycropml.cyml.**transpile_file**(*source*, *language*)

pycropml.cyml.**transpile_package**(*package*, *language*)

pycropml.description module

class pycropml.description.**Description**
Bases: object

Model Unit Description.

A description is defined by:

- Title
- Author
- Institution
- Reference
- Abstract

pycropml.error module

Created on Wed Apr 10 17:01:34 2019

@author: midingoy

exception pycropml.error.**Error**(*message*)
Bases: Exception

pycropml.formater_f90 module

pycropml.formater_f90.**formater**(*code*)

pycropml.formater_f90.**formaterNext**(*line*)

pycropml.function module

class pycropml.function.**Function**(*name*, *language*, *filename*, *type*, *description*)
Bases: object

pycropml.initialization module

class pycropml.initialization.**Initialization**(*name*, *language*, *filename*)
Bases: object

Function

pycropml.inout module

class pycropml.inout.**Input**(*kwds*)
Bases: *pycropml.inout.InputOutput*

class pycropml.inout.**InputOutput**(*kwds*)
Bases: object

class pycropml.inout.**Output**(*kwds*)
Bases: *pycropml.inout.InputOutput*

pycropml.main module

Created on Tue Mar 19 22:59:23 2019

@author: pradal

pycropml.main.**main**()

pycropml.model module

pycropml.modelunit module

Model Description and Model Unit.

class pycropml.modelunit.**ModelDefinition**(*kwds*)
Bases: object

class pycropml.modelunit.**ModelUnit**(*kwds*)
Bases: pycropml.modelunit.ModelDefinition

Formal description of a Model Unit.

add_description(*description*)
TODO

pycropml.package module

from pycropml import composition from pycropml.pparse import model_parser from path import Path import networkx as nx from collections import defaultdict from IPython.display import Image, display from networkx.drawing.nx_pydot import to_pydot from pycropml.render_cyml import signature

class pycropml.package.**AbstractPackageReader**(*filename*)
Bases: object

Abstract class to add a package in the package manager.

register_packages(*pkgmanager*)
Create and add a package in the package manager.

class pycropml.package.**Package**(*name*, *metainfo*, *path=None*)
Bases: *pycropml.package.PackageDict*

A Package is a dictionnary of node factory. Each node factory is able to generate node and their widgets. Meta informations are associated with a package.

add_modelunit(*modelunit*)
Add to the package a factory (node or subgraph)

get_crop2ml_path()
Return the full path of the wralea.py (if set)

get_id()
Return the package id

get_metainfo(*key*)
Return a meta information. See the standard key in the __init__ function documentation. :param key: todo

get_modelunit(*modelid*)
Return the factory associated with id

get_names()
Return all the factory names in a list

get_pkg_files()
Return the list of xml filename of the package. The filename are relative to self.path

get_tip()
Return the package description

is_directory()
New style package. A package is embeded in a unique directory. This directory can not contain more than one package. Thus, you can move, copy or delete a package by acting on the directory without ambiguity. Return True if the package is embeded in a directory.

is_editable()
A convention (for the GUI) to ensure that the user can modify the package.

mimetype = 'pycrop2ml/package'

reload()
Reload all xml file of the package

remove_files()
Remove pkg files

update_modelunit(*old_name*, *modelunit*)
Update factory (change its name)

class pycropml.package.**PackageDict**(**args*)
Bases: dict

Dictionnary with case insensitive key This object is able to handle protected entry begining with an '#'

get(*key*, *default=None*)
Return the value for key if key is in the dictionary, else default.

has_key(*key*)

iter_public_values()
Iterate througth dictionnary value (remove protected value)

nb_public_values()
Return the number of unprotected values

class pycropml.package.**PackageManager**
Bases: object

add_crop2ml_path(*path*, *container*)
Add a search path for wralea files

Parameters

- **path** – a path string
- **container** – set containing the path

add_package(*package*)
Add a package to the pkg manager

clear()
Remove all packages

create_readers(*crop2ml_files*)

find_and_register_packages(*no_cache=False*)
Find all composite model on the system and register them If no_cache is True, ignore cache file

find_crop2ml_dir(*directory*, *recursive=True*)
Find in a directory wralea files, Search recursivly is recursive is True

:return : a list of pkgreader instances

get(**args*)

get_pkgreader(*filename*)
Return the pkg reader corresponding to the filename

has_key(**args*)

init(*dirname=None*, *verbose=True*)
Initialize package manager

If dirname is None, find composition files on the system else load directory

items()

iteritems()

iterkeys()

itervalues()

keys()

load_directory(*dirname*)
Load a directory containing wraleas

rebuild_category()
Rebuild all the category

reload(*pkg=None*)
Reload one or all packages. If the package *pkg* is None reloa all the packages. Else reload only *pkg*.

set_sys_crop2ml_path()
Define the default composition files search path

For that, we look for "composition" entry points and deprecated_wralea entry point if a package is declared as deprecated_wralea, the module is not load

update_category(*package*)
Update the category dictionary with package contents

values()

class pycropml.package.**PseudoGroup**(*name*)
Bases: *pycropml.package.PackageDict*

Data structure used to separate dotted naming (packages, category)

add_name(*name*, *value*)
Add a value in the structure with the key name_tuple

get_id()
todo

get_tip()
todo

mimetype = 'pycrop2ml/package'

new(*name*)
todo

sep = '.'

class pycropml.package.**PyPackageReader**(*filename*)
Bases: *pycropml.package.AbstractPackageReader*

Build packages from wralea file Use 'register_package' function

build_package(*wraleamodule*, *pkgmanager*)
Build package and update pkgmanager

filename_to_module(*filename*)
Transform the filename ending with .py to the module name

get_pkg_name()
Return the OpenAlea (uniq) full package name

register_packages(*pkgmanager*)
Execute model.py

class pycropml.package.**PyPackageReaderModel**(*filename*)
Bases: *pycropml.package.PyPackageReader*

Build a package from a __wralea__.py Use module variable

build_package(*wraleamodule*, *pkgmanager*)
Build package and update pkgmanager

check_exist()

contain_pkg(*pkg*)

get_path(*pkg*, *name*)

class pycropml.package.**PyPackageWriter**(*package*)
Bases: object

Write a wralea python file

get_str()
Return string to write

pkg_template = '\n$PKGNAME\n$METAINFO\n'

wralea_template = '\n# This file has been generated at $TIME\n$PKG_DECLARATION\n'

write_wralea(*full_filename*)
Write the wralea.py in the specified filename

exception pycropml.package.**UnknownNodeError**(*name*)
Bases: Exception

class pycropml.package.**UserPackage**(*name*, *metainfo*, *path=None*)
Bases: *pycropml.package.Package*

Package user editable and persistent

pycropml.package.**get_default_home_dir**()
Return the home directory (valid on linux and windows)

pycropml.package.**get_openalea_home_dir**(*name='.pycrop2ml'*)
Return the crop2ml home directory If it doesn't exist, create it

pycropml.package.**get_userpkg_dir**(*name='user_pkg'*)
Get user package directory (the place where are the wralea.py files). If it doesn't exist, create it

pycropml.package.**is_protected**(*item*)
Return true the item is protected

pycropml.package.**lower**(*item*)

pycropml.package.**protected**(*item*)
Return corresponding protected name for item

pycropml.parameterset module

class pycropml.parameterset.**Parameterset**(*name*, *description*, *uri=None*)
Bases: object

Parameter set

pycropml.parameterset.**parameterset**(*model*, *name*, *kwds*)

pycropml.pparse module

License, Header

class pycropml.pparse.**ModelParser**
Bases: *pycropml.pparse.Parser*

Read an XML file and transform it in our object model.

Algorithm(*elt*)

Description(*Title*, *Author*, *Institution*, *Reference*, *Abstract*)

Function(*elt*)

Initialization(*elt*)

Input(*elts*)

Inputs(*Input*)

ModelUnit(*elts*)
ModelUnit (Description,Inputs,Outputs,Algorithm,Parametersets, Testsets)

Output(*elts*)

Outputs(*elts*)
Ouputs (Output)

Parameterset(*elts*)

Parametersets(*Parameterset*)

Testset(*Test*)

Testsets(*Testset*)

dispatch(*elt*)

param(*pset*, *elt*)
Param

parse(*crop2ml_dir*)

class pycropml.pparse.**Parser**
Bases: object

Read an XML file and transform it in our object model.

dispatch(*elt*)

parse(*crop2ml_dir*)

pycropml.pparse.**model_parser**(*crop2ml_dir*)

Parse a set of models as xml files contained in crop2ml directory and algorithm in src directory This function returns models as python object.

Returns ModelUnit object of the Crop2ML Model.

pycropml.render_R module

Add License, Header.

Use pkglts

Problems: - name of a model unit?

class pycropml.render_R.**Model2Package**(*models*, *dir=None*, *pkg_name=None*)
Bases: object

TODO

DATATYPE = {'BOOLEAN': <class 'bool'>, 'CHARLIST': <class 'list'>, 'DATE': <class 'str

generate_algorithm(*model_unit*)

generate_component(*model_unit*)
Todo

generate_func_test(*model_unit*)

generate_function_doc(*model_unit*)

generate_function_signature(*model_unit*)

generate_package()
Generate a R package equivalent to the xml definition.

Args: - models : a list of model - dir: the directory where the code is generated.

Returns: - None or status

generate_test(*model_unit*)

num = 0

run()
TODO.

write_tests()
TODO: Manage several models rather than just one.

pycropml.render_R.**comment**(*line*)

pycropml.render_R.**generate_doc**(*model*)

pycropml.render_R.**signature**(*model*)

pycropml.render_csharp module

Add License, Header.

Use pkglts

Problems: - name of a model unit?

class pycropml.render_csharp.**Model2Package**(*models*, *dir=None*)
Bases: object

TODO

DATATYPE = {'BOOLEAN': 'bool', 'DATE': 'string', 'DATELIST': 'List<string>', 'DOUBLE':

generate_test(*model_unit*)

num = 0

write_tests()
TODO: Manage several models rather than just one.

pycropml.render_csharp.**signature**(*model*)

pycropml.render_csharp.**transf**(*type_v*, *elem*)

pycropml.render_csharp.**transfDate**(*type*, *elem*)

pycropml.render_csharp.**transfDateList**(*type*, *elem*)

pycropml.render_csharp.**transfDouble**(*type_v*, *elem*)

pycropml.render_csharp.**transfList**(*type_v*, *elem*)

pycropml.render_csharp.**transfString**(*type_v*, *elem*)

pycropml.render_cyml module

Add License, Header. Use pkglts Problems: - name of a model unit?

class pycropml.render_cyml.**Model2Package**(*models*, *dir=None*, *pkg_name=None*)
Bases: object

TODO

generate_algorithm(*model_unit*)

generate_component(*model_unit*)
Todo

generate_func_test(*model_unit*)

generate_function_doc(*model_unit*)

generate_function_signature(*func_name*, *model_unit*)

generate_package()
Generate a Cyml package equivalent to the xml definition. Args: - models : a list of model - dir: the directory where the code is generated. Returns: - None or status

generate_test(*model_unit*)

initialization(*model_unit*)

num = 0

run()
TODO.

write_tests()
TODO: Manage several models rather than just one.

pycropml.render_cyml.**generate_doc**(*model*)

pycropml.render_cyml.**my_input**(*_input*, *defa=True*)

pycropml.render_cyml.**signature**(*model*)

pycropml.render_cyml.**transBool**(*type*, *elem*)

pycropml.render_cyml.**transf**(*type_*, *elem*)

pycropml.render_cyml.**transfDate**(*type*, *elem*)

pycropml.render_cyml.**transfDateList**(*type*, *elem*)

pycropml.render_fortran module

Add License, Header.

Use pkglts

Problems: - name of a model unit?

class pycropml.render_fortran.**Model2Package**(*models*, *directory=None*, *pkg_name=None*)
Bases: object

TODO

DATATYPE = {'BOOLEAN': 'LOGICAL', 'DATE': 'CHARACTER(65)', 'DATELIST': 'CHARACTER(65),

generate_test(*model_unit*)

my_input(*_input*)

num = 0

write_tests()
TODO: Manage several models rather than just one.

pycropml.render_fortran.**generate_doc**(*model*)

pycropml.render_fortran.**signature**(*model*)

pycropml.render_java module

Add License, Header.

Use pkglts

Problems: - name of a model unit?

class pycropml.render_java.**Model2Package**(*models*, *dir=None*)
Bases: object

DATATYPE = {'BOOLEAN': 'boolean', 'DATE': 'String', 'DATELIST': 'Arrays.asList', 'DOUB

generate_test(*model_unit*)

num = 0

write_tests()
TODO: Manage several models rather than just one.

pycropml.render_java.**formatDate**(*elem*)

pycropml.render_java.**formatDateList**(*elem*)

pycropml.render_java.**signature**(*model*)

pycropml.render_java.**transf**(*type_v*, *elem*)

pycropml.render_java.**transfDate**(*categ*, *name*, *elem*)

pycropml.render_java.**transfDateList**(*categ*, *name*, *elem*)

pycropml.render_java.**transfDouble**(*type_v*, *elem*)

pycropml.render_java.**transfList**(*type_v*, *elem*)

pycropml.render_java.**transfString**(*type_v*, *elem*)

pycropml.render_notebook module

License, Header

Use pkglts

Problems: - name of a model unit?

class pycropml.render_notebook.**Model2Nb**(*models*, *dir=None*)
Bases: *pycropml.render_python.Model2Package*

Generate a Jupyter Notebook from a set of models.

generate_notebook()
Generate a Python package equivalent to the xml definition.

Args: - models : a list of model - dir: the directory where the code is generated.

Returns: - None or status

generate_test(*model_unit*)

run()
TODO.

pycropml.render_notebook_csharp module

License, Header

Use pkglts

Problems: - name of a model unit?

class pycropml.render_notebook_csharp.**Model2Nb**(*models*, *dir=None*)
Bases: *pycropml.render_csharp.Model2Package*

Generate a Jupyter Notebook from a set of models in Csharp.

generate_notebook()
Generate a csharp package equivalent to the xml definition.

Args: - models : a list of model - dir: the directory where the code is generated.

Returns: - None or status

generate_test(*model_unit*)

run()
TODO.

pycropml.render_notebook_csharp.**signature**(*model*)

pycropml.render_notebook_csharp.**transf**(*type*, *elem*)

pycropml.render_notebook_csharp.**transfDate**(*type*, *elem*)

pycropml.render_notebook_csharp.**transfDateList**(*type*, *elem*)

pycropml.render_notebook_csharp.**transfDouble**(*type*, *elem*)

pycropml.render_notebook_csharp.**transfSDIList**(*type*, *elem*)

pycropml.render_notebook_csharp.**transfString**(*type*, *elem*)

pycropml.render_notebook_java module

License, Header

Use pkglts

Problems: - name of a model unit?

class pycropml.render_notebook_java.**Model2Nb**(*models*, *dir=None*)
Bases: *pycropml.render_java.Model2Package*

Generate a Jupyter Notebook from a set of models in Java.

generate_notebook()
Generate a java package equivalent to the xml definition.

Args: - models : a list of model - dir: the directory where the code is generated.

Returns: - None or status

generate_test(*model_unit*)

run()
TODO.

pycropml.render_notebook_java.**signature**(*model*)

pycropml.render_python module

Add License, Header.

Use pkglts

Problems: - name of a model unit?

class pycropml.render_python.**Model2Package**(*models*, *dir=None*, *pkg_name=None*)
Bases: object

TODO

DATATYPE = {'BOOLEAN': <class 'bool'>, 'CHARLIST': <class 'list'>, 'DATE': <class 'str

generate_algorithm(*model_unit*)

generate_component(*model_unit*)
Todo

generate_factory(*model*)
Create a Node Factory from CropML model unit.

generate_func_test(*model_unit*)

generate_function_doc(*model_unit*)

generate_function_signature(*model_unit*)

generate_package()
Generate a Python package equivalent to the xml definition.

Args: - models : a list of model - dir: the directory where the code is generated.

Returns: - None or status

generate_test(*model_unit*)

generate_wralea()
Generate wralea factories from the meta-information of the the model units.

num = 0

run()
TODO.

write_tests()
TODO: Manage several models rather than just one.

pycropml.render_python.**generate_doc**(*model*)

pycropml.render_python.**openalea_interface**(*inout*)

pycropml.render_python.**signature**(*model*)

pycropml.test_generator module

pycropml.test_generator.**generate_test_check**(*model*, *dir=None*)

pycropml.test_generator.**generate_test_cs**(*model*, *dir*)

pycropml.test_generator.**generate_test_f90**(*model*, *dir*)

pycropml.test_generator.**generate_test_java**(*model*, *directory=None*)

pycropml.test_generator.**generate_test_openalea**(*model*, *dir=None*)

pycropml.test_generator.**generate_test_py**(*model*, *dir=None*)

pycropml.test_generator.**generate_test_simplace**(*model*, *dir=None*)

pycropml.test_generator.**generate_test_sirius**(*model*, *dir=None*)

pycropml.topology module

class pycropml.topology.**Package**(*name*, *metainfo*, *path=None*)
Bases: object

class pycropml.topology.**PackageManager**(*proj*)
Bases: object

check_exist()

contain_pkg()

get_path(*pkg*)

class pycropml.topology.**Topology**(*name*, *pkg=None*)
Bases: object

algo2cyml()

algorithm()

check_compo(*mc*, *m*)

compotranslate(*language*)

createGraph()

create_edgeInOut()

decl(*defa=True*)

display_wf()

generate_function_signature(*model*)

get_mu_inp(*pkgname*, *varname*)

get_mu_out(*pkgname*, *varname*)

info_inputs_mu(*ppkg*, *mu*, *varname*)

info_minout()

info_outputs_mu(*ppkg*, *mu*, *varname*)

isPackage(*name*)

load_pkge(*name*)

meta_ext(*pkgname*)

meta_inp(*pkgname*)

meta_out(*pkgname*)

minout()

pkg_m(*mc*, *m*)

pkgs = {}

retrive(*pkgname*)

topologicalSort()

translate()

translate_all(*model*)

val_init(*model*)

write_png()

write_xml()

pycropml.version module

Maintain version for this package. Do not edit this file, use 'version' section of config.

pycropml.version.**MAJOR = 0**
(int) Version major component.

pycropml.version.**MINOR = 1**
(int) Version minor component.

pycropml.version.**POST = 1**
(int) Version post or bugfix component.

pycropml.wf2xml module

pycropml.writeTest module

Created on Mon Mar 18 15:46:31 2019

@author: midingoy

class pycropml.writeTest.**WriteTest**(*models*, *language*, *dir*)
Bases: object

write()
Populate and write the test files.

pycropml.writeTest_f90 module

Created on Thu Mar 28 15:39:28 2019

@author: midingoy

Add License, Header.

Use pkglts

Problems: - name of a model unit?

class pycropml.writeTest_f90.**Model2Package**(*models*, *dir=None*)
Bases: object

TODO

DATATYPE = {'BOOLEAN': 'LOGICAL::', 'DATE': 'CHARACTER(65)', 'DATELIST': 'CHARACTER(6

generate_algorithm(*model_unit*)

generate_component(*model_unit*)
Todo

generate_estimation(*model_unit*)

generate_function_doc(*model_unit*)

generate_package()
Generate a csharp package equivalent to the xml definition.

Args: - models : a list of model - dir: the directory where the code is generated.

Returns: - None or status

generate_public_class(*model_unit*)

generate_test(*model_unit*)

num = 0

run()
TODO.

write_tests()
TODO: Manage several models rather than just one.

pycropml.writeTest_f90.**signature**(*model*)

pycropml.writeTest_f90.**transf**(*type*, *elem*)

pycropml.writeTest_f90.**transfDate**(*type*, *elem*)

pycropml.writeTest_f90.**transfDateList**(*type*, *elem*)

pycropml.writeTest_f90.**transfDouble**(*type*, *elem*)

pycropml.writeTest_f90.**transfSDIList**(*type*, *elem*)

pycropml.writeTest_f90.**transfString**(*type*, *elem*)

pycropml.xml2wf module

class pycropml.xml2wf.**XmlToWf**(*xmlwf*, *dir*, *pkg_name*)
Bases: object

compareInterface(*interfaces*)

compoPack(*name*)

compositeNodeInputs()

compositeNodeOutputs()

connectInputs()

connectInternal()

connectOutputs()

createNodes()

retrievePackage(*name*)

run()

Module contents

1.1.6 Usecases

1.1.7 Licence

PyCropML is released under a MIT License.

1.1.8 Usecases

1.1.9 Glossary

Terminology

Model Simplified representation of the crop system within specific objectives.

Overview

1.2 Documentation

- A PDF version of |core| documentation is available.

1.3 History

1.3.1 creation (2018-01-18)

- First release on PyPI.

1.4 Indices and tables

1.5 History

1.5.1 creation (2018-01-18)

- First release on PyPI.

1.6 License

PyCropML is released under a MIT License.

1.7 Welcome to CropML's documentation!

Contents:

1.7.1 Contributing Guide

This is a wiki for anything related to the contributing on [[Crop2ML|https://github.com/AgriculturalModelExchangeInitiative|Crop2ML]] which is a project of the Agricultural Model Exchange Initiative. For more information about this project, please visit CropML documentation [[Crop2ML|https://cropmlformat.readthedocs.io/en/latest/?badge=latest|documentation]]:

1.7.2 Quick Links

- [[Project Git Repository|https://github.com/cython/cython|Git Repository]] (and [[Change Log|https://github.com/cython/cython/blob/master/CHANGES.rst|Change Log]])
- [[Differences between Cython and Pyrex|https://cython.readthedocs.io/en/latest/src/userguide/pyrex_differences.html| Differences between Cython and Pyrex]]
- [[Unsupported Python features|https://cython.readthedocs.io/en/latest/src/userguide/limitations.html|Unsupported Python features]] (aka TODO list)
- [[Hacker-Guide: How to work on the Cython compiler itself|HackerGuide| Hacker-Guide: How to work on the Cython compiler itself]]
- [[Enhancement proposals|enhancements| Enhancement proposals]] (CEPs)
- [[Projects using Cython|projects| Projects using Cython]]
- [[Comparison with SWIG|SWIG| Comparison with SWIG]]
- [[Automatic .pxd/.pyx generation|AutoPxd|Automatic .pxd/.pyx generation]] from C or C++ header files.

Cython Installers

- [[PyPi|http://pypi.python.org/pypi/Cython/|PyPi]] via `easy_install` or pip
- [[Gentoo Ebuild|http://packages.gentoo.org/package/dev-python/cython|Gentoo Ebuild]]
- [[Debian package|http://packages.debian.org/sid/cython|Debian package]] (not always up to date)
- [[Installing Cython on Windows|InstallingOnWindows| Installing Cython on Windows]]

Tips and Tricks

- [[Getting started|http://docs.cython.org/src/quickstart/index.html|Getting started]]
- [[Using early binding techniques to improve speed|http://docs.cython.org/en/latest/src/userguide/early_binding_for_speed.html| Using early binding techniques to improve speed]]
- [[Writing Cython programs in pure Python|http://docs.cython.org/src/tutorial/pure.html| Writing Cython programs in pure Python]]
- [[Helpful notes for wrapping C++ APIs|http://docs.cython.org/en/latest/src/userguide/wrapping_CPlusPlus.html| Helpful notes for wrapping C++ APIs]]
- [[Discussion of all the options how to wrap C/C++ code to Python|WrappingCorCpp| Discussion of all the options how to wrap C/C++ code to Python]]
- [[WritingFastPyrexCode|http://www.sagemath.org:9001/WritingFastPyrexCode|WritingFastPyrexCode]]
- [[Successful creation of a hierarchy of modules in a package|PackageHierarchy| Successful creation of a hierarchy of modules in a package]]
- [[One method for source-level debugging|http://docs.cython.org/en/latest/src/userguide/debugging.html| One method for source-level debugging]]
- [[Dynamic Memory Allocation (malloc, realloc, free)|http://docs.cython.org/en/latest/src/tutorial/memory_allocation.html| Dynamic Memory Allocation (malloc, realloc, free)]]
- [[Profiling]]
- [[Building a Windows Installer|BuildingWindowsInstaller| Building a Windows Installer]]

- [[Embedding Python|EmbeddingCython|Embedding Python]] to create standalone Cython programs.
- [[List Subclass Example|ListExample|List Subclass Example]] Adding mathematical operations to subclassed built-in list.
- Working with Numpy
  - [[Tutorial for NumPy users|http://docs.cython.org/en/latest/src/userguide/numpy_tutorial.html|Tutorial for NumPy users]]
  - [[Accessing a Numpy pointer for passing to C|http://docs.cython.org/en/latest/src/userguide/memoryviews.html#pass-data-from-a-c-function-via-pointer]]

1.7.3 People

[[Stefan Behnel|http://scoder.behnel.de/|Stefan Behnel]], [[Robert Bradshaw|http://www.math.washington.edu/~robertwb/|Robert Bradshaw]], [[Dag Seljebotn|http://heim.ifi.uio.no/dagss/|Dag Seljebotn]], Lisandro Dalcin.

1.7.4 Mailing Lists

Our development mailing list is [[cython-dev|http://mail.python.org/mailman/listinfo/cython-devel|cython-dev]] and user mailing list at http://groups.google.com/group/cython-users.

In the past we also used a [[Google group|http://groups.google.com/group/cython|Google group]] and a list at [[BerliOS Developer|https://lists.berlios.de/mailman/listinfo/cython-dev|BerliOS Developer]]. You can still read [[the archives at Gmane|http://blog.gmane.org/gmane.comp.python.cython.devel|the archives at Gmane]].

1.7.5 Project Goals

- Fully supported easy-to-use test suite, including the normal CPython test suite.
- Easy installation and usage.
- Rich, accessible documentation. Make sure the examples are plenty and can be automatically tested.
- Make Cython part of the standard distribution of Python (like ctypes).
- Compile all Python code except for possibly some obvious exclusions, which will be worked out by developers.
- Very fast when the user explicitly declares types (but we're not going to make promises with type inference). Precise benchmarks.
- Mitigate or eliminate the need for users to invoke the Python/C API directly without sacrificing performance.

1.7.6 Documentation

- See http://docs.cython.org/.
- Official Pyrex [[Language Overview|http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/version/Doc/LanguageOver Overview]] (note the [[changes|http://hg.cython.org/cython|changes]] though).
- [[Extension Types|http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/version/Doc/Manual/extension_types.html|Ex Types]]
- [[Sharing Declarations Between Pyrex Modules|http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/version/Doc/Ma Declarations Between Pyrex Modules]]
- [[FAQ|http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/version/Doc/FAQ.html|FAQ]]

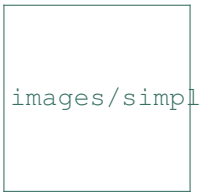
- [[Quick Guide to Pyrex|http://ldots.org/pyrex-guide/|Quick Guide to Pyrex]] from Michael Jason-Smith.
- CategoryCythonDoc lists pages that are related to Cython documentation.
- [[Pure Python mode|pure| Pure Python mode]]
- SAGE Days 4 talk highlighting some of the [[differences between Pyrex and SageX|http://cython.org/talks/SageX.pdf|differences between Pyrex and SageX]] (the predecessor of Cython).

CategoryHomepage

1.8 Indices and tables

1.9 Supported by:


images/siriusquality.png


images/simplace.png

Python Module Index

p

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y