# pycropml Documentation

***Release 0.0.2***

**Cyrille Ahmed Midingoyi**

**May 02, 2018**

# CONTENTS

CHAPTER

# ONE

# CONTENTS

**Summary**

**Version** 0.0.2

**Release** 0.0.2

**Date** May 02, 2018

**Author** See **'authors'**_ section

**ChangeLog** See **'changelog'**_ section

## 1.1 What is PyCropML?

**PyCropML** 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.

### 1.1.1 What is CropML ?

**CropML** is a language based on XML format that allows to represent different biological processes involved in the crop models.
CropML project is intented to provide a common framework for defining and exchanging descriptions of crop growth models between crop simulation frameworks.

### 1.1.2 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

### 1.1.3 Context

Nowadays, we observe the emergence of plant growth models which are built in differents 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 modellers to define a standard framework based on the development of declarative language and libraries to improve exchange model components between platforms.

### 1.1.4 Motivation

- 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.

### 1.1.5 Vision

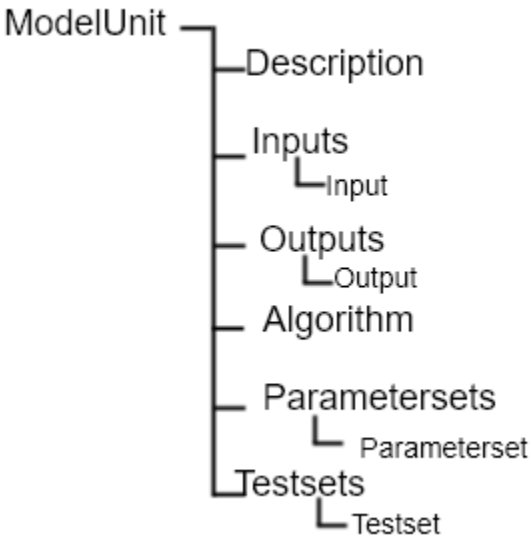
- Use modular modelling to share knowledge and rapidly develop operational tools.
- Reuse model parts to leverage the expertise of third parties;
- Renovate legacy code.
- Realize the benefit of sharing and complementing different expertise.

## 1.2 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.
These models have a specific formal definition in CropML.

### 1.2.1 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 schema.
This structure MAY be described by the below tree:




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=" "/>
      <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="float" min="" max=""  unit="" uri=""/>
      <Output name="" description="" datatype="float" 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>
      <Testset name="" parameterset = "" description="" uri = ""/>
      <Testset name="" parameterset = "" description="" >
         <Test name="">
            <InputValue name="">value</InputValue>
            ...
            <OutputValue name="" precision ="">value</OutputValue>
            ...
         </Test>
         ...
      </Testset>
      ...
  </Testsets>
  ...
</ModelUnit>
```

### 1.2.2 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 composite model. 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.

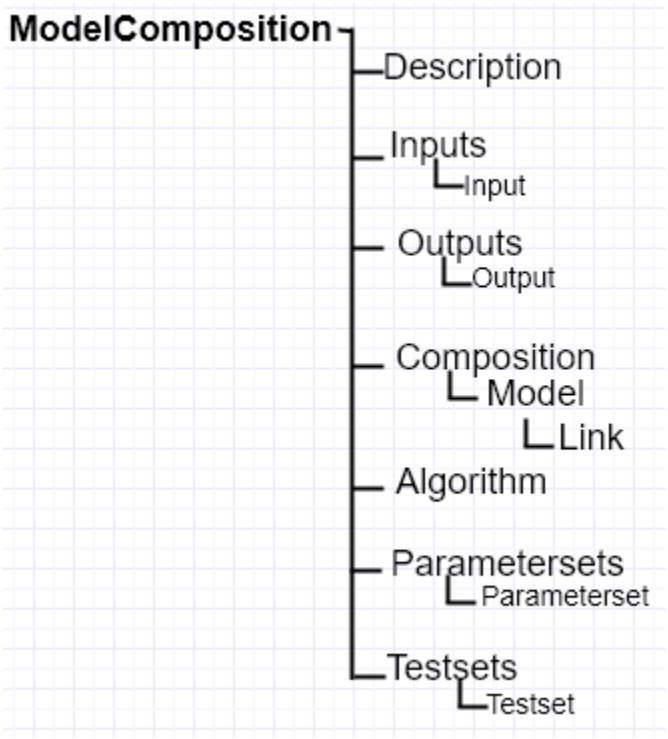
The structure of a Composite Model in CropML MUST be conform to a specific Document Type Definition named ModelComposition.dtd .

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.

## 1.3 PyCropML User Guide

**Version** 0.0.2

**Release** 0.0.2

**Date** Apr 30, 2018

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.

### 1.3.1 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

## 1.4 src

### 1.4.1 pycropml package

#### Submodules

#### pycropml.algorithm module

**class** pycropml.algorithm.**Algorithm**(*language*, *development*)
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.description module

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

Model Unit Description.

**A description is defined by:**

- Title
- Author
- Institution
- Reference
- Abstract

### 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.modelunit module

**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.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*)

**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**(*fn*)

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

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

**dispatch**(*elt*)

**parse**(*fn*)

pycropml.pparse.**model_parser**(*fn*)
Parse a set of models as xml files and return the models.

Returns ModelUnit object of the CropML Model.

### 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_python module

License, Header

Use pkglts

Problems: - name of a model unit?

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

TODO

**DATATYPE = {'DOUBLEARRAY': <built-in function array>, 'Double': <type 'float'>, 'int'**

**generate_algorithm**(*model_unit*)

**generate_component**(*model_unit*)
Todo

**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*)

**num = 0**

**run**()
TODO.

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

### 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 = 0**
(int) Version minor component.

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

#### Module contents

## 1.5 Usecases

## 1.6 Licence

PyCropML is released under a MIT License.

## 1.7 Usecases

## 1.8 Glossary

Terminology

**Model** Simplified representation of the crop system within specific objectives.

CHAPTER

TWO

# CREDITS

**1. Development Lead**

- Cyrille Ahmed Midingoyi, <cyrille.midingoyi@inra.fr>
- Christophe Pradal, <christophe.pradal@cirad.fr>

**2. Contributors**

None yet. Why not be the first?

## 2.1 History

### 2.1.1 creation (2018-01-18)

- First release on PyPI.

## 2.2 Indices and tables

- genindex
- modindex
- search

## 2.3 Supported by:




simplace
modelling framework

# PYTHON MODULE INDEX

## p

# INDEX