Table of Contents

How to declare OpenAlea component package

edit	1372259159	[How to declare OpenAl	section	1-57

OpenAlea packages

Openalea components are placed in a **OpenAlea package**. An OpenAlea package is a particular *Python package*.

It is a **directory** with :

a `__init__`.py : : containing python package declaration
a `__wralea__`.py : containing openalea component declaration
python modules

edit	1372259159	[OpenAlea packages]	section	58-391

OpenAlea components

OpenAlea *logical components* are special classes which **wrap** functionalities to be used in the dataflow. They are represented graphically by a box with input and output ports.

A component is composed by:

Inputs and Outputs port (to exchange data with other components)
A function to execute (executed when the graph is evaluated)
A graphical widget for user interaction

Each input or output has an interface which defines the *type* of the expected data. This interface allow to do checks or automatic conversions, but also is used to generate default visualization widgets.

edit	1372259159	[OpenAlea components]	section	

392-1023

Component and package declaration : __wralea__.py

The `__wralea__.py` file contains

package meta informations
component declaration

Important

For memory management reason, we separate the *node declaration* and the *node implementation*. The node implementation contains the function to execute. The node declaration is a simple description of the node and contains meta informations.

This `__wralea__.py` file **MUST NOT CONTAIN NODE FUNCTION OR CLASSES**, but just their description. The nodes classes will be placed in an other file. It should also avoid to load unnecessary modules in order to save memory.

An example of __wralea__.py

```
from openalea.core import *

__name__ = "arithmetics"
__version__ = '0.0.1'
__license__ = 'CECILL-C'
__authors__ = 'OpenAlea consortium'
__institutes__ = 'INRIA/CIRAD'
__description__ = 'Arithmetic nodes.'
__url__ = 'http://openalea.gforge.inria.fr'

__editable__ = 'False'

__alias__ = ["old_arithmetics"] # Aliases for compatibitity

__all__ = ['linearmodel',]

linearmodel = Factory( name="linearmodel",
                       description="Linear Model",
                       category="Model,Math",

                       inputs=(dict(name='X', interface=IFloat, value=., desc='X
variable'),
                               dict(name='A', interface=IFloat, value=., desc=
'Coefficent'),
                               dict(name='B', interface=IFloat, value=., desc=
'Offset'),
```

```
                    ),
                outputs=(dict(name='Y', interface=IFloat),),

                nodemodule="model",
                nodeclass="LinearModel",

                widgetmodule=None, # optional
                widgetclass=None,  # optional
                lazy=True,         # optional
            )


Alias(linearmodel, "linear") # alias for linearmodel factory
```

Package meta-informations

OpenAlea packages have meta informations :

name
version
license
authors
institutes
url

They are used to identify the origin of the package.

alias : a list of alias name for the package
editable : allows or not to edit the package
?

Meta informations are declared in the `__wralea__.py`

Components declaration : the factories

`Factory` allows to declare components. The `__wralea__.py` file can instantiate several `Factory`. They are identified with the `__all__` field :

`__all__` is a global variable containing the list of variable name pointing to a factory.

A `Factory` contains different fields:

`name` : Component string id
`description` : Description string
`category` : String to classify the components, the category are separated with coma ex (?Types, Simulation?)
`nodemodule` : the name of the python module (without .py) containing the component function/class
`nodeclass` : the name of the component function/class (as a string)
`inputs`[optional] : the input ports description (a list of dict)

`outputs`[optional] : the output ports description (a list of dict)
`widgetmodule`[optional] : the name of the python module (without .py) containing the widget class
`widgetclass`[optional] : the name of the widget class (as a string)
`lazy`[optional] : The node support lazy evaluation (default is True)
`alias` [default=None] : list of alias names

Nota :

inputs and *outputs* can be set to None. The system will then try to guess values.
widgetmodule and *widgetclass* can be set to None if you don't provide any widget.

Inputs and Outputs ports

Inputs and **outputs** are described by a list of dictionaries. Each dictionary represent a *port*. The port order corresponds to the declaration order.

The accepted parameters are :

`name` : the port name
`interface`[default=None] : interface class (or instance) to describe the port.
`value`[default=None]: default value
`desc`: help string
`label`: port widget title
`showwidget`[default=True]: display the corresponding interface widget
`hide` [default=False] : hide the port

edit	1372259159	[Component and packag	section	a_

1024-5353

Component Implementation

Component implementation can be :

a simple python function
a functor class (a class with a __call__ method)
a class which inherits from Node

Node classes are not written in `__wralea__.py`, but in an other file. This is important to avoid importing all the modules at the same time.

Simple Function

The node can be defined with a simple function in the file `model.py`:

```
def linearmodel(x, a, b):
        return a*x + b
```

Functor

Implementation of a LinearModel functor class in the file `model.py`:

```
class LinearModel(object):
    """
    y = Ax + B model
    Inputs : x, a, b
    Output : y
    """

    def __init__(self):
        pass

    def __call__(self, x, a, b):
        y = a*x + b
        return y
```

edit	1372259159	[Component Implementa	section	

5354-6225

Widget Declaration

Widget are not mandatory, you can declare a node without a widget. By default, a composite widget will be created depending of the Interface of input ports.

edit	1372259159	[Widget Declaration]	section	

6226-6420

Widget Implementation

Widgets allow to interact with nodes in the OpenAlea graphical user interface. We distinguish :

Node Widgets
Interface Widgets

Node Widget are arbitrary graphical box which are displayed when double clicking on the node. The widget can be a viewer, a configuration box or everything else.

Interface Widget are box to display and/or edit data corresponding to a particular Interface. If a node doesn't have an associated widget, a compound widget will be creating containing the interface widgets which corresponds to the Interface of the node input ports. Interface widget can be reuse for different type of node. This article doesn't deal with *interface widget*. See how_to_declare_interfaces_and_adapters

Node Widget

Node widget are associated to a node in the `wralea.py` file, in the factory declaration with the fields `widgetclass` and `widgetmodule`.

A NodeWidget has a `node` property which holds the associated *Node* instance.
The `obj` property holds the associated functor object (if the node class inherits form *Node* then `obj == node`).

When a node is modified, it notifies its associated widget with `notify` function.

Notify function

`sender` : the object which send the message
`event` : a tuple

`(?input_modified?, input_index)` : an input of the node has been modified
`(?data_modified?,)` : an internal data has been modified
`(?status_modified?, state)` : the node status has been modified

edit	1372259159	[Widget Implementation]	section	6421-