Core Packages

Davide Rizzi

Version 1.0, 2016-04-13

Table of Contents

Core Packages

Overview

A Core package is ??

A package can include:

1. Messages
2. Nodes
3. Node Configurations
4. Libraries or other code

Anatomy of a Core package

From a Core user point of view a package is nothing but a directory that meets the following requirements:

1. It contain a file named `CORE_PACKAGE.json`
2. It contain a license file `LICENSE`
3. It have a well defined structure

CORE_PACKAGE.json

The `CORE_PACKAGE.json` file describes the package, and it is used by the Core build system either to manage the dependencies and to generate the makefiles.

The file must be valid according to an Apache Avro [1: Apache Avro LINK] schema.

Example: `CORE_PACKAGE.json` *file for* `led` *package*

```
{
  "name": "led", ①
  "description": "LED Nodes", ②
  "namespace": "@", ③
  "sources": [ ④
    "PublisherNode.cpp",
    "SubscriberNode.cpp"
   ]
}
```

① the `name` of the message field must match the name of the package directory

② a brief description

③ by default ('@') the namespace will be the name of the package; `namespace` can be used to override it

④ list of source files that will be compiled and linked with the target

Package directory structure

There are two kinds of packages, depending on their content: messages or non-messages.

It is surely possible to have messages in a non-messages package, but this is strongly discouraged.

The reason for such a division resides in the fact that it is possible (it is likely) that we don’t need the nodes (and their configurations) all the times.
When we only need the messages it is easier to avoid the MOTIVARE

Messages packages

Messages package must contain nothing but message definition files.

These files must be inside a directory called `messages`.

```
led_msgs
├─ CORE_PACKAGE.json
├─ LICENSE
└─ messages
   └─ Led.json
```

Figure 1. `led_msgs` Package

Non-Messages packages

Non-Messages package must not contain any message definition files.

Source code must be put inside the `include` and `src` directories, for header and source respectively. Nodes definitions must reside in `nodes` directory, while node configuration definitions must be in `configurations`.

```
led
├─ CORE_PACKAGE.json
├─ LICENSE
├─ include
│  ├─ PublisherNode.hpp
│  └─ SubscriberNode.hpp
├─ src
│  ├─ PublisherNode.cpp
│  └─ SubscriberNode.cpp
├─ nodes
│  ├─ PublisherNode.json
│  └─ SubscriberNode.json
└─ configurations
   ├─ PublisherNodeConfiguration.json
   └─ SubscriberNodeConfiguration.json
```

Figure 2. `led` Package

The CorePackage.py command

`CorePackage.py` is a command line tools that permits to:

1. list the content of a package

2. generate the code of a package

It is possible to use `TAB` to invoke sub-command or package name completion.

ls

Example 1. Example

```
$ CorePackage.py ls
```

```
 ╒════════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════╕
 │ PACKAGE                                                                       │
 ╘════════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════╛
Name: led
Description: LED Nodes
Root: packages/led



=== CONFIGURATIONS =============================================================
NS: led
Name: PublisherConfiguration
Description: LED Publisher node configuration
Source: configurations/PublisherConfiguration.json

NS: led
Name: SubscriberConfiguration
Description: LED Subscriber node configuration
Source: configurations/SubscriberConfiguration.json
```