|  |  |  |  |
| --- | --- | --- | --- |
| Title: | Chassis 2023 Package File | Created: | 2023-09-12 |
| Filename: | CH23PACK.docx | Author: | Lion Kimbro |
| Status: | In Development (2023-09-12) | Related: | CH23CORE.docx |

# Intent

This file details the Chassis 2023 Package File contents.

# Progress

|  |  |
| --- | --- |
| 2023-09-12 | started document, by copying data out from the Chassis 2023 Design & Development document. |
| 2022-11-03 | wrote details for pikacall.py & pikaconf.py; added: “reconstructed history”, and “tasks” sections |
|  |  |
|  |  |
|  |  |

# Package File

The “package file” (formerly “program file”) is the JSON file that defines a given package, authoritatively, in Lion's Factory 2023 system, and in chassis2023.

|  |  |  |  |
| --- | --- | --- | --- |
| *fact key* | *fact value* | *information* | *date recorded* |
| ~~program filename:~~ | ~~program.json~~ | ~~filename of the Program File~~ | ~~2023-04-30~~ |
| package definition filename: | package.json | filename for the file containing definitions for the package | 2023-08-01 |
|  |  |  |  |

The package file describes the package’s self-identification, the modules it makes use of, the resources it needs in order to run, and every dimension of configuration of the package.

It does NOT describe the *user’s* configuration of a program. For example, say the program is a computer game, and the user can configure the joystick and screen dimensions and other aspects like that: That’s the user’s configuration, and the program itself will manage the user’s configuration.

Rather, this is the configuration of *the package itself*, and it is intended to be maintained and manipulated by the programmer, his or her own self.

The package file is kept in a file called package.json, and I absolutely do want it to be manipulated via tools. JSON does not support comments, and so “#COMMENT” keys may be used in the dictionary in order to be a place for human readable comments to be kept in the JSON dictionaries. They are to be ignored, systematically, if encountered, unless specifically displaying the comment for the developer, as a comment, is something that is valued.

I am not using TOML, because I want this file to be machine-written, and I suspect comments that are not part of the data itself, will be lost.

# Package File Content Tracking

I am interested in the following information, about each key-value pair in the Package File:

* when was it added
* what is it’s key
* what is it’s logical type
* what is it’s semantic type
* what does it do
* what is leaning on this key-value pair

This is more information than comfortably fits on a page, so I am rendering it in two tables.

# Package File Contents

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date\* | *Key* | *Logical Type* | *Semantic Type* | *Description* |
| *found* | **APPID** | k-v dictionary | APPID dictionary |  |
| *found* | .GUID | str | GUID (typically v4, including dashes) | unique identifier for the package |
| *found* | .TAGURI | str | RFC 4151 TAG URI | unique identifier for the package |
| *found* | .NAME | str | lowercase [\_a-z][\_a-z0-9]\* name | programmer-friendly identifier for the package, and the name that is used in “import” statements; contrast PACKAGE.DISTRIBUTIONNAME |
| *found* | .TITLE | str | human readable title | developer or user facing title string, for the package |
| *found* | .TAGS | str | lowercase [\_a-z][\_a-z0-9]\* space-separated tags | list of tags describing the package; this is local to the programmer, and not intended for tagging on (say) github |
| *found* | .DESC | str | paragraph of human-readable text | a brief description of the package |
|  |  |  |  |  |
| *found; updated 2023-08-01* | **PACKAGE** | k-v dictionary | package definition dictionary | details of the package’s definition that don’t fit anywhere else; for example, the EXECUTIONTYPE  (formerly titled “PROGRAM” – updated to “PACKAGE” 2023-08-01) |
| 2023-08-03 | .DISTRIBUTIONNAME | str | legal distribution package name | the name of the package, per PyPI’s distribution system – it doesn’t need to match the actual package name |
| 2023-08-03 | .VERSION | str | version string, by any system | the version of the package, for PyPI’s distribution system |
| 2023-08-03 | .CLASSIFIERS | list of str | list of PyPI classifier strings | the classifiers that are used to locate and characterize packages in PyPI |
| *~~found~~* | ~~.EXECUTIONTYPE~~ | ~~str~~ | ~~Categorical Values: CLITOOL, WEBSERVER, FILETALKSERVER, TKINTERGUI, INTERACTIVEMENU, LIBRARY~~ | ~~what kind of executable the program is~~ *moved to CHASSIS2023* |
| 2023-08-01 | .DEPENDENCIES | list of str | list of package names, locatable in PyPI | package distribution names that will be installed as dependencies, that this package relies on |
|  |  |  |  |  |
| 2023-08-01 | **CHASSIS2023** | k-v dictionary | chassis2023 definition dictionary | details of the program’s definition that configure chassis2023’s capabilities & use |
| *found* | .LOGRINGLEN | int | length of the log ring | how long the chassis2023 log ring can get |
| 2023-08-13 | .SESSIONFILELOC | str | flags: “S” (system location), “X” (execution directory), “M” (manually specified) | where the session file can be located |
| 2023-08-13 | .SESSIONFILENAME | str | filename of the session file (incl “.json” extension), or None/null | full filename for the session file; must end with “.json” |
| 2023-08-13 | .EXECUTIONTYPE | str | Categorical Values: “CLITOOL”, “WEBSERVER”, “FILETALKSERVER”, … | what kind of executable the program is; full documentation in separate document: “Chassis 2023 Execution Types”. |
|  |  |  |  |  |
| 2023-08-01 | **FACTORY2023** | k-v dictionary | factory definition dictionary | information about how the program shows up in Lion’s factory system |
| 2023-08-01 | .PROJECTID | str | Factory project identity string | an ID, such as “0100”, for the package, in Lion’s factory system |
|  |  |  |  |  |
| *2023-08-03* | **GITHUB** | k-v dictionary or None |  | if the project shows up on Github, use a k-v dictionary, otherwise, just None |
| 2023-08-03 | .NAME | str | legal github project name | name of the project, on Github |
| *found* | **MODULES** | k-v dictionary | module inclusion dictionary | various ways of indicating the modules to include |
| *found* | **RESOURCES** | list of k-v dictionaries | list of resource dictionaries, per resources module | these resources will be automatically loaded and saved |
| 2023-08-01 | **SNOWFLAKES** | list of k-v dicts | list of snowflake definition dictionaries, per snowflakes system | these snowflake systems will be automatically loaded and saved |
| 2023-08-01 | **CONFIG** | list of k-v dicts | package config value dict | a list of options that can be configured for the package by the developer |

(\*) “found” dates – these were entered before 2023-08-01, and likely added on or around 2023-04-30.

# Dependencies

One record is provided per dependency. There can be multiple rows for a key.

If a key exists, but has no known dependencies, it still gets a row, just the “Use” field is left blank.

Keys:

* Began: the date that code that depends on this key, was written
* Noted: the date when this dependency was noted
* Key: the key that is depended on
* Use: the use of the key itself, in English

|  |  |  |  |
| --- | --- | --- | --- |
| *Began* | *Noted* | *Key* | *Use* |
|  |  | **APPID** |  |
|  |  | .GUID |  |
|  |  | .TAGURI |  |
|  |  | .NAME |  |
| *found* | 2023-09-12 | .TITLE | chassis2023.chassis2023.collect\_cli\_data() uses the title to present the title to the user of the CLI --help command |
|  |  | .TAGS |  |
|  |  | .DESC |  |
|  |  |  |  |
|  |  | **PACKAGE** |  |
|  |  | .DISTRIBUTIONNAME |  |
|  |  | .VERSION |  |
|  |  | .CLASSIFIERS |  |
|  |  | .DEPENDENCIES |  |
|  |  |  |  |
|  |  | **CHASSIS2023** |  |
|  |  | .LOGRINGLEN |  |
|  |  | .SESSIONFILELOC |  |
|  |  | .SESSIONFILENAME |  |
| *found* | 2023-09-12 | .EXECUTIONTYPE | chassis2023.chassis2023 critically relies on this information, repeatedly, to run the program |
|  |  |  |  |
|  |  | **FACTORY2023** |  |
|  |  | .PROJECTID |  |
|  |  |  |  |
|  |  | **GITHUB** |  |
|  |  | .NAME |  |
|  |  | **MODULES** |  |
|  |  | **RESOURCES** |  |
|  |  | **SNOWFLAKES** |  |
| *found* | 2023-09-12 | **CONFIG** | chassis2023.chassis2023.collect\_cli\_data() uses the CONFIG to present CLI options to the user, making use of NAME, DESC, TYPE, DEFAULT, and OPTIONS |

# The Module Configuration Value dictionary

Modules can specify that values are to be selected or configured by the developer, in the program configuration.

Each value to be configured is specified in this dictionary.

|  |  |  |  |
| --- | --- | --- | --- |
| *Key* | *Logical Type* | *Semantic Type* | *Description* |
| NAME | str | [\_a-z][\_a-z0-9]\* name | unique identifier (within the module) for the configuration value |
| TITLE | str | human readable title | developer facing title string, for the configuration value |
| DESC | str | paragraph of human-readable text | a brief description of the configuration value |
| TYPE | str | categorical values: STR, INT, FLOAT, BOOL, OPTION | a type for the value that the developer can or must enter |
| DEFAULT | str, int, float, bool, or None | valid value of particular type | default value when the module is added to the program, but the developer has not manually selected a value |
| OPTIONS | list of (str, int, float, bool, or None) | valid values of particular type | if the list is empty, it means that there are no specific options; but if the list is not empty, then the developer must select from among these options |