Module Manifests

Manifest

The manifest file serves to declare a python package as an Odoo module and to specify module metadata.
It is a file called `__manifest__.py` and contains a single Python dictionary, where each key specifies module metadatum.

```
{
    'name': "A Module",
    'version': '1.0',
    'depends': ['base'],
    'author': "Author Name",
    'category': 'Category',
    'description': """
    Description text
    """,
    # data files always loaded at installation
    'data': [
        'views/mymodule_view.xml',
    ],
    # data files containing optionally loaded demonstration data
    'demo': [
        'demo/demo_data.xml',
    ],
}
```

Available manifest fields are:

`name` (`str` , required)
the human-readable name of the module

`version` (`str`)
this module's version, should follow semantic versioning (https://semver.org) rules

`description` (`str`)
extended description for the module, in reStructuredText

`author` (`str`)
name of the module author

`website` (`str`)
website URL for the module author

`license` (`str` , defaults: `LGPL-3`)
distribution license for the module. Possible values:

`GPL-2`

`GPL-2 or any later version`

`GPL-3`

`GPL-3 or any later version`

`AGPL-3`

`LGPL-3`

`Other OSI approved licence`

`OEEL-1` (Odoo Enterprise Edition License v1.0)

`OPL-1` (Odoo Proprietary License v1.0)

`Other proprietary`

`category` **(** **`str`** **, default:** **`Uncategorized`** **)**

classification category within Odoo, rough business domain for the module.

Although using existing categories (https://github.com/odoo/odoo/blob/14.0/odoo/addons/base/data/ir_module_category_data.xml is recommended, the field is freeform and unknown categories are created on-the-fly. Category hierarchies can be created using the separator `/` e.g. `Foo / Bar` will create a category `Foo` , a category `Bar` as child category of `Foo` , and will set `Bar` as the module's category.

`depends` **(** **`list(str)`** **)**

Odoo modules which must be loaded before this one, either because this module uses features they create or because it alters resources they define.

When a module is installed, all of its dependencies are installed before it. Likewise dependencies are loaded before a module is loaded.

`data` **(** **`list(str)`** **)**

List of data files which must always be installed or updated with the module. A list of paths from the module root directory

`demo` **(** **`list(str)`** **)**

List of data files which are only installed or updated in *demonstration mode*

`auto_install` **(** **`bool`** **, default:** **`False`** **)**

If `True` , this module will automatically be installed if all of its dependencies are installed.

It is generally used for "link modules" implementing synergic integration between two otherwise independent modules.

For instance `sale_crm` depends on both `sale` and `crm` and is set to `auto_install` . When both `sale` and `crm` are installed, it automatically adds CRM campaigns tracking to sale orders without either `sale` or `crm` being aware of one another

`external_dependencies` **(** **`dict(key=list(str))`** **)**

A dictionary containing python and/or binary dependencies.

For python dependencies, the `python` key must be defined for this dictionary and a list of python modules to be imported should be assigned to it.

For binary dependencies, the `bin` key must be defined for this dictionary and a list of binary executable names should be assigned to it.

The module won't be installed if either the python module is not installed in the host machine or the binary executable is not found within the host machine's PATH environment variable.

`application` (`bool`, default: `False`)

Whether the module should be considered as a fully-fledged application (`True`) or is just a technical module (`False`) that provides some extra functionality to an existing application module.

`css` (`list(str)`)

Specify css files with custom rules to be imported, these files should be located in `static/src/css` inside the module.

`images` (`list(str)`)

Specify image files to be used by the module.

`installable` (`bool` default: `True`)

Whether a user should be able to install the module from the Web UI or not.

`maintainer` (`str`)

Person or entity in charge of the maintenance of this module, by default it is assumed that the author is the maintainer.

`{pre_init, post_init, uninstall}_hook` (`str`)

Hooks for module installation/uninstallation, their value should be a string representing the name of a function defined inside the module's `__init__.py` .

`pre_init_hook` takes a cursor as its only argument, this function is executed prior to the module's installation.

`post_init_hook` takes a cursor and a registry as its arguments, this function is executed right after the module's installation.

`uninstall_hook` takes a cursor and a registry as its arguments, this function is executed after the module's uninstallation.

These hooks should only be used when setup/cleanup required for this module is either extremely difficult or impossible through the api.

`active` (`bool`)

This indicates whether this module must install automatically or not.