OxOOL Module v4 Compiling HOWTO

1. Preparing The Environment

- Follow the instructions of OxOOL Community Compiling HOWTO to compile oxool-community -- English version coming soon
  - Be noticed that you should use `autogen.sh` and don't directly run configure here. `autogen.sh` by default would add `--enable-debug` and load ModuleTesting module for developing and test modules.
  - Generate `oxool` and `oxool-dev` rpm/deb packages by `rpmbuild` or `debuild`.
- Install `oxool` and `oxool-dev` you just generated into your development environment. oxool-dev contains the headers necessary for building modules, module templates and `oxool-module-maker` which will make it a lot easier to start a new module.
  - Notice that you also need to install `oxoffice*` packages before installing `oxool`. This is described in the documents of OxOOL Community Compiling HOWTO

2. Create an basic module repository by oxool-module-maker

- Run `oxool-module-maker` to create a module git repository from a module template. The parameters are:
  - `--module-name=name` : Module name, used for the git repository name. It must match the regular expression: `^[\w\@#]+$` which means all the upper and lower case letters, number 0-9, underline _ and # @ symbols. Notice that dash '-' is not allowed due to C++ class naming issue, and underline '_' is better not used.
  - `--serviceURI=serviceURI` : URI used by this module. If it ends with a slash '/' it means this module can handle a series of related commands. For example, "/oxool/user/" means this module is used to handle requests such as /oxool/user/add, /oxool/user/del, /oxool/user/update, and so on. Without the ending slash means it is a fixed URI.
  - `--version=version` : Version number. Default "0.0.1"

- `--summary=summary` : Specify Summary field, used in Summary fields in generated rpm file.
- `--description=Description` : Specify Description field, used in Description fields in generated deb file.
- `--author=author` : Module author. By default it will use the user.name and user.email in global git settings.
- `--license=license` : Module license. Default "MPLv2.0".
- `--adminPrivilege=true/false` : Does this module URI needs admin privileges? Default "false".
- `--adminIcon=icon` : The icon used in backend administration page (See getbootstrap.com for references.) Default "bug-fill".
- `--adminItem=text` : The text/title used in backend administration page.
- `--template-path=path` : Assign the module template path. The new created module will be generated by copying files from this path. By default the module template is in `/usr/share/oxool-devel/module-template`.
- `--output-path=path` : The path of generated module. By default it will be in user home directory. Notice that it may cause problems if you use symbols like ~ so better use a full path name.
- Example:

```
oxool-module-maker --module-name="samplemod" --serviceURI="/oxool/samplemod/" --summary="A sample moudle service on OxOOL" --description="A sample module service on OxOOL" --output-path="/home/oxool/git"
```

- A new git repository will be generated in the specified output-path. Inside the folder are all the files from the specified template-path folder.

3. Compile and Test Modules

- Use `autogen.sh` to generate `configure`, then run `configure` to check the environment and generate Makefile.
- Compile with `make`.
- Before testing the module, go to oxool-community folder and run `make run` to activate a testing oxool environment. Notice that when you install `oxool` packages in your environment, it will activate a systemd service `oxool.service` and occupy the default port like 9980. So you may need to deactivate this systemd service by `sudo systemctl stop oxool` before running the testing oxool environment.
- After starting the testing environment, go back to the module folder and run `./test.sh <XML file for this module>`. `test.sh` will call ModuleTesting module in oxool and sent the XML path to it. ModuleTesting will then notify oxool to load the .so file of this module (under .libs/ of the module folder). Now you can call and run the module by calling the URI specified in `serviceURI`. For example, open your browser with `http://127.0.0.1:9980/oxool/samplemod/` .
- If you change something in the module, you need to re-compile the module and re-run `test.sh` to load the new .so into oxool to make it effect.

Revision #3
Created 2 August 2023 05:34:54 by Jeff Huang
Updated 2 August 2023 06:43:58 by Jeff Huang