Программный проект и иерархия каталогов

Содержание

Для операционных систем типа Linux принят стандарт FHS («стандарт иерархии файловой системы»), унифицирующий местонахождение файлов и каталогов с общим назначением в файловой системе. Полная текущая версия стандарта находится здесь.

Типы расположения проекта

В соответствии с данным стандартом, а также принятыми в ведущих дистрибутивах правилами размещения исполняемых файлов в каталогах пользователей, можно выделить следующие типы расположения:

- системная иерархия в каталоге `/usr` используется для установки бинарных пакетов для данного дистрибутива;
- системная иерархия в каталоге `/usr/local` используется для установки программного обеспечения системным администратором без использования пакетов (не рекомендуется для использования из-за проблем поддержки в актуальном состоянии);
- системная иерархия в каталоге `/opt` используется для установки стороннего программного обеспечения. В рамках данной иерархии предполагается, что каждый программный продукт располагается в собственном каталоге. При таком типе сборки обычно используются дополнительные методы (статическая компоновка, включение в состав пакета своего набора динамических библиотек) для обеспечения работы пакета в операционных системам с отличающимся составом библиотек и другим циклом обновления;
- системная иерархия в домашнем каталоге пользователя не имеет определённого

стандарта, обычно производители дистрибутивов предлагают использовать для исполняемых файлов каталоги `$HOME/bin` или `$HOME/.local/bin`.

Система автоматизации сборки программного обеспечения CMake позволяет организовать окружение подобное перечисленным выше. На этапе сборки проекта можно создать структуру каталогов, которая будет отвечать требованиям по логическому разделению файлов на исполняемые, заголовочные, библиотеки, файлы настроек и т.д.

Автоматическая адаптация к текущему окружению

Для обеспечения единообразной работы вне зависимости от варианта иерархии каталогов, в которой находится исполняемый файл, можно выполнять автоматическую настройку на работу в текущем окружении.

В библиотеке myxlib реализован класс, который анализирует расположение и окружение исполняемого файла и предоставляет методы для получения имён каталогов, соответствующих текущему окружению. Названия методов и описания возвращаемых значений приведены в таблице.

Таблица 1. Имена методов и описания

Метод	Описание
`homeDirectory()`	Полный путь к домашнему каталогу текущего пользователя
`tempDirectory()`	Полный путь к каталогу с временными файлами
`userConfigDirectory()`	Полный путь к пользовательскому каталогу с файлами настройки
`userConstDataDirectory()`	Полный путь к пользовательскому каталогу с неизменяемыми файлами
`userVarDataDirectory()`	Полный путь к пользовательскому каталогу с изменяемыми файлами
`userLogDirectory()`	Полный путь к пользовательскому каталогу с журналами работы
`executableFilePath()`	Полный путь к исполняемому файлу
`systemConfigDirectory()`	Полный путь к системному каталогу с файлами настройки
`systemConstDataDirectory()`	Полный путь к системному каталогу с неизменяемыми файлами
`systemVarDataDirectory()`	Полный путь к системному каталогу с изменяемыми файлами
`systemLogDirectory()`	Полный путь к системному каталогу с журналами работы
`executableFileDirectory()`	Полный путь к каталогу с исполняемым файлом
`executableFileName()`	Имя исполняемого файла

Метод	Описание
`projectName()`	Имя подкаталога для проекта

Пример использования:

```
#include <myx/filesystem/paths.hpp>
namespace MF = myx::filesystem;

MF::Paths& paths = MF::Paths::instance();
paths.init();
qDebug() << paths.systemConstDataDirectory().path();
```

Правила выбора типа окружения

Класс `myx::filesystem::Paths` реализован в виде синглтона, чтобы повторно не выполнять проверку окружения в разных частях программы. Сначала определяются имена пользовательского и временного каталогов с помощью вызовов функций `QDir::homePath` и `QDir::tempPath`, затем имена пользовательских каталогов для настроек, постоянных и изменяемых данных и журналов. Эти значения не зависят от расположения исполняемого файла, а определяются в соответствии со значениям переменных окружения `HOME`, `TMPDIR`, `XDG_CONFIG_HOME` и `XDG_DATA_HOME`, либо устанавливаются значения, принятые в стандартах. Пример имён каталогов для пользователя `user`, названия организации `org`, названия выполняемой работы `theme` и проекта `project` приведён в таблице.

Таблица 2. Стандартные каталоги для текущего пользователя

Назначение каталога	Метод	Значение
Домашний каталог	`homeDirectory()`	`/home/user`
Временные файлы	`tempDirectory()`	`/tmp`
Файлы настройки	`userConfigDirectory()`	`/home/user/.config/org-theme/project`
Неизменяемые файлы	`userConstDataDirectory()`	`/home/user/.local/share/org-theme/project/share`
Изменяемые файлы	`userVarDataDirectory()`	`/home/user/.local/share/org-theme/project/var`
Журналы работы	`userLogDirectory()`	`/home/user/.local/share/org-theme/project/log`

Общая проверка

Для определения типа текущего окружения используется полный путь к исполняемому файлу, если он находится в каталоге `bin`, то выполняются проверки работы в одной из возможных вариантов иерархий, иначе делается заключение о том, что файлы всех типов находятся в одном каталоге с исполняемым и дальнейшие проверки не выполняются.

ВАЖНО	При проверке типов иерархии всегда проверяется наличие всех необходимых каталогов, при отсутствии хотя бы одного будет принято решение, что файлы всех типов находятся в одном каталоге с исполняемым.

Проверка на работу в иерархии `/opt`

Если полный путь к исполняемому файлу начинается с `/opt` и содержит в себе название текущего проекта, например `/opt/org-theme/project/bin/application`, то выполняется проверка на наличие сопутствующих системных каталогов. Если они присутствуют, то принимается решение, что окружение в иерархии `/opt` сформировано правильно, иначе делается заключение о том, что файлы всех типов находятся в одном каталоге с исполняемым и дальнейшие проверки не выполняются. Пример правильной структуры каталогов для данной иерархии приведён в таблице.

Таблица 3. Каталоги в иерархии `/opt`

Назначение файла / каталога	**Метод**	**Значение**
Исполняемый файл	`executableFilePath()`	`/opt/org-theme/project/bin/application`
Файлы настройки	`systemConfigDirectory()`	`/opt/org-theme/project/etc`
Неизменяемые файлы	`systemConstDataDirectory()`	`/opt/org-theme/project/share`
Изменяемые файлы	`systemVarDataDirectory()`	`/opt/org-theme/project/var`
Журналы работы	`systemLogDirectory()`	`/opt/org-theme/project/log`

Проверка на работу в иерархии `/usr/local`

Если полный путь к исполняемому файлу начинается с `/usr/local`, например `/usr/local/bin/application`, то выполняется проверка на наличие сопутствующих системных каталогов. Если они присутствуют, то принимается решение, что окружение в иерархии `/usr/local` сформировано правильно, иначе делается заключение о том, что файлы всех типов находятся в одном каталоге с исполняемым и дальнейшие проверки не выполняются. Пример правильной структуры каталогов для данной иерархии приведён в таблице.

Таблица 4. Каталоги в иерархии `/usr/local`

Назначение файла / каталога	**Метод**	**Значение**
Исполняемый файл	`executableFilePath()`	`/usr/local/bin/application`
Файлы настройки	`systemConfigDirectory()`	`/usr/local/etc/project`
Неизменяемые файлы	`systemConstDataDirectory()`	`/usr/local/share/project`
Изменяемые файлы	`systemVarDataDirectory()`	`/var/lib/project`

Назначение файла / каталога	Метод	Значение
Журналы работы	`systemLogDirectory()`	`/var/log/project`

Проверка на работу в иерархии `/usr`

Если полный путь к исполняемому файлу начинается с `/usr`, например `/usr/bin/application`, то выполняется проверка на наличие сопутствующих системных каталогов. Если они присутствуют, то принимается решение, что окружение в иерархии `/usr` сформировано правильно, иначе делается заключение о том, что файлы всех типов находятся в одном каталоге с исполняемым и дальнейшие проверки не выполняются. Пример правильной структуры каталогов для данной иерархии приведён в таблице.

Таблица 5. Каталоги в иерархии `/usr`

Назначение файла / каталога	Метод	Значение
Исполняемый файл	`executableFilePath()`	`/usr/bin/application`
Файлы настройки	`systemConfigDirectory()`	`/etc/project`
Неизменяемые файлы	`systemConstDataDirectory()`	`/usr/share/project`
Изменяемые файлы	`systemVarDataDirectory()`	`/var/lib/project`
Журналы работы	`systemLogDirectory()`	`/var/log/project`

Проверка на работу в домашнем каталоге

Если полный путь к исполняемому файлу начинается с `/home/user/bin` или `/home/user/.local/bin`, например `/home/user/bin/application`, то выполняется проверка на наличие сопутствующих системных каталогов. Если они присутствуют, то принимается решение, что окружение в домашнем каталоге сформировано правильно, иначе делается заключение о том, что файлы всех типов находятся в одном каталоге с исполняемым и дальнейшие проверки не выполняются. Пример правильной структуры каталогов для данной иерархии приведён в таблице.

Таблица 6. Каталоги при работе в домашнем каталоге

Назначение файла / каталога	Метод	Значение
Исполняемый файл	`executableFilePath()`	`/home/user/bin/application`
Файлы настройки	`systemConfigDirectory()`	`/home/user/.config/org-theme/project`
Неизменяемые файлы	`systemConstDataDirectory()`	`/home/user/.local/share/org-theme/project/share`
Изменяемые файлы	`systemVarDataDirectory()`	`/home/user/.local/share/org-theme/project/var`
Журналы работы	`systemLogDirectory()`	`/home/user/.local/share/org-theme/project/log`

Проверка на работу в окружении для разработки

Если исполняемый файл находится в каталоге `bin` и при этом окружение не совпадает ни с одним из перечисленных выше, то делается предположение, что исполняемый файл запускается из окружения, сформированного системой управления проектом, и в данный момент идёт разработка (отладка) приложения. В этом случае целесообразно считать системными каталогами те, которые находятся внутри иерархии каталогов программного проекта. Если присутствуют каталоги, созданные системой управления проекта, то принимается решение, что окружение сформировано правильно, иначе делается заключение о том, что файлы всех типов находятся в одном каталоге с исполняемым и на этом проверки заканчиваются.

Пример правильной структуры каталогов для данной иерархии приведён в таблице.

Таблица 7. Каталоги при работе в окружении для разработки

Назначение файла / каталога	**Метод**	**Значение**
Исполняемый файл	`executableFilePath()`	`/home/user/work/project/_build/debug/bin/application`
Файлы настройки	`systemConfigDirectory()`	`/home/user/work/project/_build/debug/etc`
Неизменяемые файлы	`systemConstDataDirectory()`	`/home/user/work/project/_build/debug/share`
Изменяемые файлы	`systemVarDataDirectory()`	`/home/user/work/project/_build/debug/var`
Журналы работы	`systemLogDirectory()`	`/home/user/work/project/_build/debug/log`

Расположение в одном каталоге

Если в ходе перечисленных выше проверок не удалось найти правильно сформированное окружение, то применяется настройка по умолчанию, которая соответствует ситуации, когда все типы файлов расположены в одном каталоге с исполняемым файлом. Пример для такого случая приведён в таблице.

Таблица 8. Каталоги в неопределённой иерархии

Назначение файла / каталога	**Метод**	**Значение**
Исполняемый файл	`executableFilePath()`	`/home/user/work/project/application`
Файлы настройки	`systemConfigDirectory()`	`/home/user/work/project`
Неизменяемые файлы	`systemConstDataDirectory()`	`/home/user/work/project`
Изменяемые файлы	`systemVarDataDirectory()`	`/home/user/work/project`
Журналы работы	`systemLogDirectory()`	`/home/user/work/project`