

# Brief Introduction

---

In Deepin Desktop Environment, the Dock not only has highly customizable appearance, but also provided API document. Every community developer can extend it by your own interest to enrich its function.

A plugin is made by icon, tooltip, popup and menu, etc.

The combination of each part can make a full functional plugin: a standard and well-compatible dock plugin not only provides the functions user needed, but also get the same level in user experiences like native plugin (battery, sound, network, etc.). The recommended interactions are: \* showing popup when left-clicked at the icon; \* showing menu when right-clicked at the icon; \* showing tooltip when the mouse stopped at the icon, etc.

The following article is about how to develop a high quality plugin.

# Preparation

---

Before developing the Dock plugin, you need to install some of the package and tools to assist the developing work. Execute the following command in terminal:

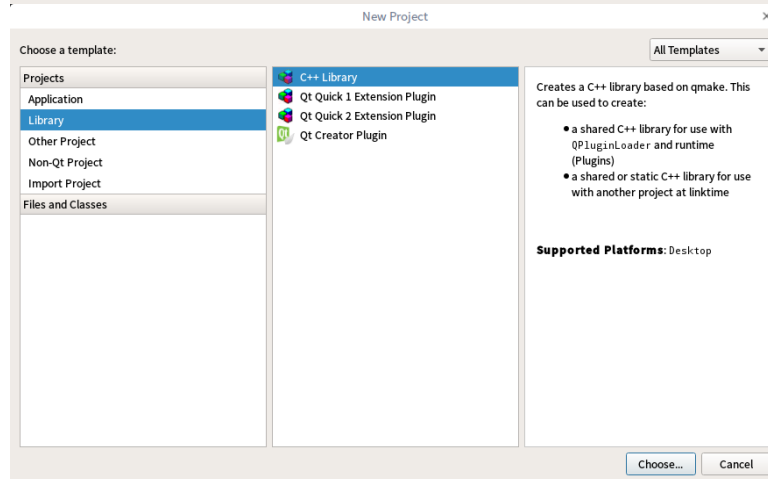
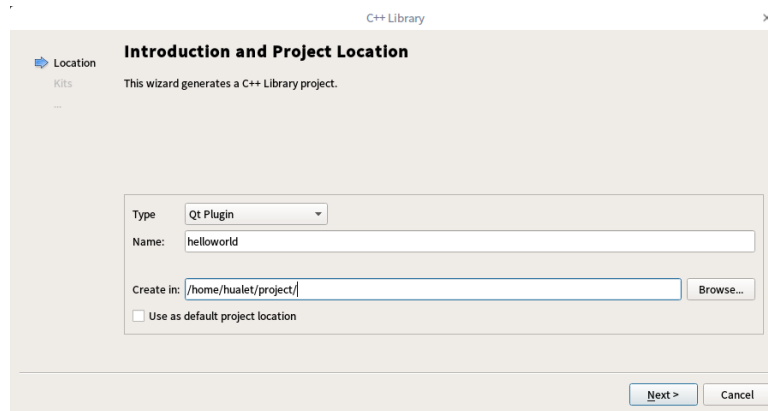
```
sudo apt-get install dde-dock-dev build-essential qt5-  
qmake qt5-default qtcreator
```

In these packages, the `qt5-default` is optional, mainly configure qt5 instead of qt4 as the default development environment. If you don't understand this, install `qt5-default` directly.

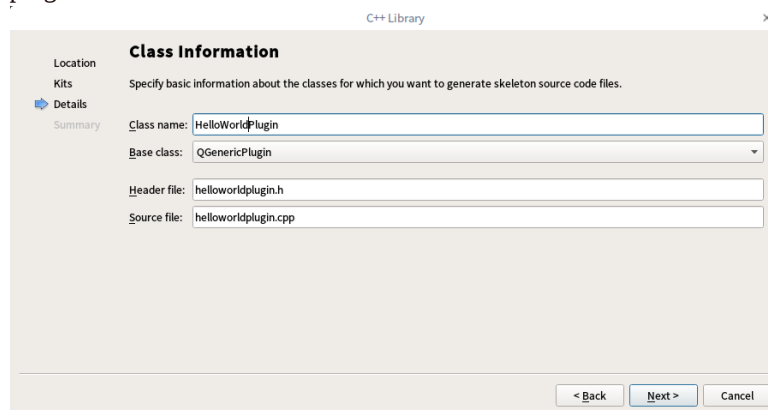
# Introduction with Images

---

1. Open QtCreator, created a new "Qt Plugin" project, for example, named "helloworld".



2. Created a class named "HelloWorldPlugin" as the entry point of the plugin.



3. After created the project, you need to modify some of the created project config.

First, open helloworldplugin.h and helloworldplugin.cpp to change the inheritance relation, from QGenericPlugin to QObject and PluginsItemInterface,

```

#include <dde-dock/constants.h>
#include <dde-dock/pluginsiteminterface.h>
class HelloWorldPlugin : public QObject,
PluginsItemInterface
{
    Q_OBJECT
#ifdef QT_VERSION >= 0x050000
    Q_INTERFACES(PluginsItemInterface)
    Q_PLUGIN_METADATA(IID
"com.deepin.dock.PluginsItemInterface" FILE
"helloworld.json")
#endif // QT_VERSION >= 0x050000

```

*Notice: Here added `Q_INTERFACES(PluginsItemInterface)`, and modified IID to `"com.deepin.dock.PluginsItemInterface"`.*

Then, delete `DESTDIR` in `helloworld.pro`, it was pointed to the system directory, if you don't delete, it will show errors when compiling.

Now it's time for the plugin to implement all the method in `PluginItemInterface` interface. Because all need now is `pluginName` and `init`, `itemWidget`, `itemTipsWidget` method, other function can simply implement as empty or return empty:

```

const QString HelloWorldPlugin::pluginName() const
{
    return "helloworld";
}

```

`pluginName` returns the name of the plugin, you need to confirm its uniqueness:

```

QWidget *HelloWorldPlugin::itemWidget(const QString
&itemKey)
{
    Q_UNUSED(itemKey);
    return m_icon;
}

```

`itemWidget` is used to provide the icon of the dock plugin, now returns `QWidget` for flexibility; here assigned `itemKey` is because one plugin can provide unlimited combinations of icon, popup and menu, so each combination can get different `itemKey`, this strategy is for the plugin too complicated (for example, the system tray), we won't use this field for now (the same below):

```

QWidget *HelloWorldPlugin::itemTipsWidget(const QString
&itemKey)
{
    Q_UNUSED(itemKey);
    return m_tooltip;
}

```

itemTipsWidget is used to provide the tooltip of the dock plugin, returns QLabel in general, but here we return the QLabel "Hello World" for demonstration:

```

HelloWorldPlugin::HelloWorldPlugin() :
    QObject(nullptr),
    m_icon(new QLabel),
    m_tooltip(new QLabel("Hello World"))
{
    QPixmap pix(":/deepin.png");
    m_icon->setPixmap(pix);
    m_icon->setMargin(6);
    m_icon->setAlignment(Qt::AlignCenter);
    m_icon->setScaledContents(true);

    m_tooltip->setStyleSheet("color:white;"
                            "padding:5px 10px;");
}

```

Last, the init function is to initialize the dock plugin, notify the dock to add the icon by PluginProxy:

```

void HelloWorldPlugin::init(PluginProxyInterface
*proxyInter)
{
    proxyInter->itemAdded(this, "");
}

```

After successfully compiled the plugin, put the generated libhelloworld.so to /usr/lib/dde-dock/plugins/ directory, relaunch the dock to see the newly developed plugin. move the mouse point to the custom icon, is there shows the "Hello World" tooltip? :)

## Add popup

Popup provided a new interaction way more easily and more pragmatically than normal desktop applications. Besides, the dock plugin is different from the system tray icon, the dock plugin can get its own popup.

The following is about how to add a popup:

Create a new class named "HelloPopup" and inherited from QWidget, here take a simple function for example: showing a Label says "click me ! ", when user clicked it, it will show a non-stop lengthened string "deepin".

```
HelloWorldPlugin::HelloWorldPlugin() :  
  
    QObject(nullptr),  
    m_icon(new QLabel),  
    m_tooltip(new QLabel("Hello World")),  
    m_applet(new HelloPopup)
```

Create a instance named HelloPopup and return the string in itemTipsWidget.

```
QWidget *HelloWorldPlugin::itemTipsWidget(const QString  
&itemKey)  
{  
    Q_UNUSED(itemKey);  
    return m_tooltip;  
}
```

Save and recompile the libhelloworld.so file, relaunch the dock to see the non-stop lengthened string "deepin".

## Add right context menu

The dock has provided the specified API for plugin to add menus. the two key functions are: itemContextMenu and invokedMenuItem.

```

const QString HelloWorldPlugin::itemContextMenu(const
QString &itemKey)
{
    Q_UNUSED(itemKey)

    QList<QVariant> items;
    items.reserve(1);

    QMap<QString, QVariant> open;
    open["itemId"] = MenuIdOpenEyes;
    open["itemText"] = "Open Eyes";
    open["isActive"] = true;
    items.push_back(open);

    QMap<QString, QVariant> close;
    close["itemId"] = MenuIdCloseEyes;
    close["itemText"] = "Close Eyes";
    close["isActive"] = true;
    items.push_back(close);

    QMap<QString, QVariant> menu;
    menu["items"] = items;
    menu["checkableMenu"] = false;
    menu["singleCheck"] = false;

    return QJsonDocument::fromVariant(menu).toJson();
}

void HelloWorldPlugin::invokedMenuItem(const QString
&itemKey, const QString &menuId, const bool checked)
{
    Q_UNUSED(itemKey);
    Q_UNUSED(checked);
    if (menuId == MenuIdOpenEyes) {
        QProcess::startDetached("xeyes");
    } else if (menuId == MenuIdCloseEyes) {
        QProcess::startDetached("killall xeyes");
    }
}

```

The first function provides the content of the menu, the second function is used to get the notify of the plugin and start process. This time the menu has added the two items: "Open Eyes" and "Close Eyes" to open and close xeyes application.

If you don't have this application, please execute the following command in terminal:

```
sudo apt-get install x11-apps
```

Only implement these two functions won't register the menu for your plugin, you should also call the requestContextMenu method of PluginProxy in a certain situation (for the right context menu of the icon):

```
bool HelloWorldPlugin::eventFilter(QObject *watched,
QEvent *event)
{

    if (watched == m_icon && event->type() ==
QEvent::MouseButtonPress) {
        QMouseEvent *mouse = static_cast<QMouseEvent*>
(event);
        if (mouse->button() == Qt::RightButton) {
            m_proxyInter->requestContextMenu(this, "");
            return true;
        }
    }

    return false;
}
```

*Notice: the structure m\_proxyInter need to assign its value in init function. After recompile and replace the libhelloworld.so file, then relaunch the dock, the right context menu will show. you can check the right context menu by your own.*

## Conclusion

---

It is not easy to develop an outstanding plugin. Although an easy plugin can run normally, but how to adapt the two modes and size of the dock needs patient of the developer to make the plugin perfect.