It's repository with plugins for MASS application.

To use this plugins, download and prepare MASS first: https://gitlab.com/c4rb0n_un1t/MASS

git clone https://gitlab.com/c4rb0n_un1t/MASS

When MASS is loaded open MASS folder and download this repository inside.

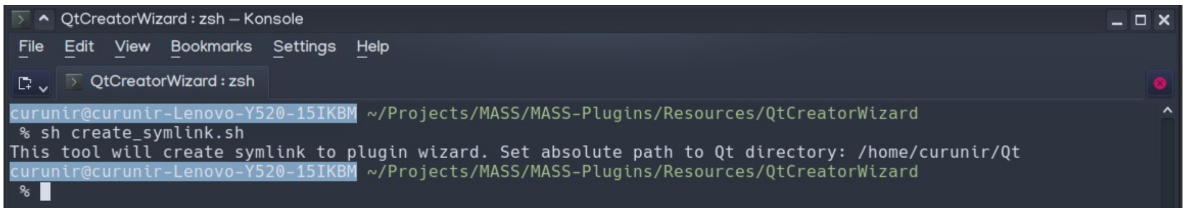
cd MASS

git clone https://gitlab.com/c4rb0n_un1t/MASS-Plugins

After this you're ready to use existing and add new plugins.

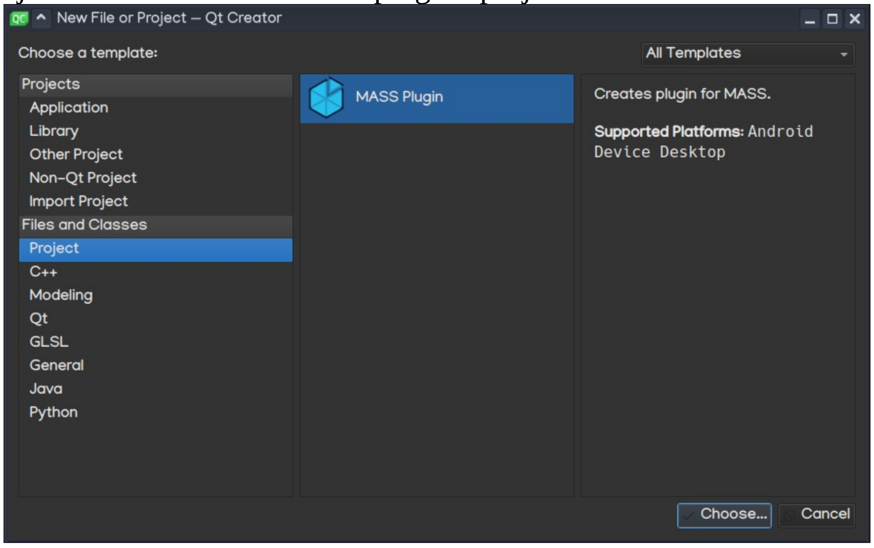
How to create plugin

Create symlink of QtCreator wizard directory "MASS/MASS-Plugins/Resources/QtCreatorWizard/Plugin" to "[your Qt installation directory]/share/qtcreator/templates/wizards" directory.
On Linux you can do it by calling "MASS/MASS-Plugins/Resources/QtCreatorWizard/create_symlink.sh".



```
curunir@curunir-Lenovo-Y520-15IKBM ~/Projects/MASS/MASS-Plugins/Resources/QtCreatorWizard
 % sh create_symlink.sh
This tool will create symlink to plugin wizard. Set absolute path to Qt directory: /home/curunir/Qt
curunir@curunir-Lenovo-Y520-15IKBM ~/Projects/MASS/MASS-Plugins/Resources/QtCreatorWizard
 %
```

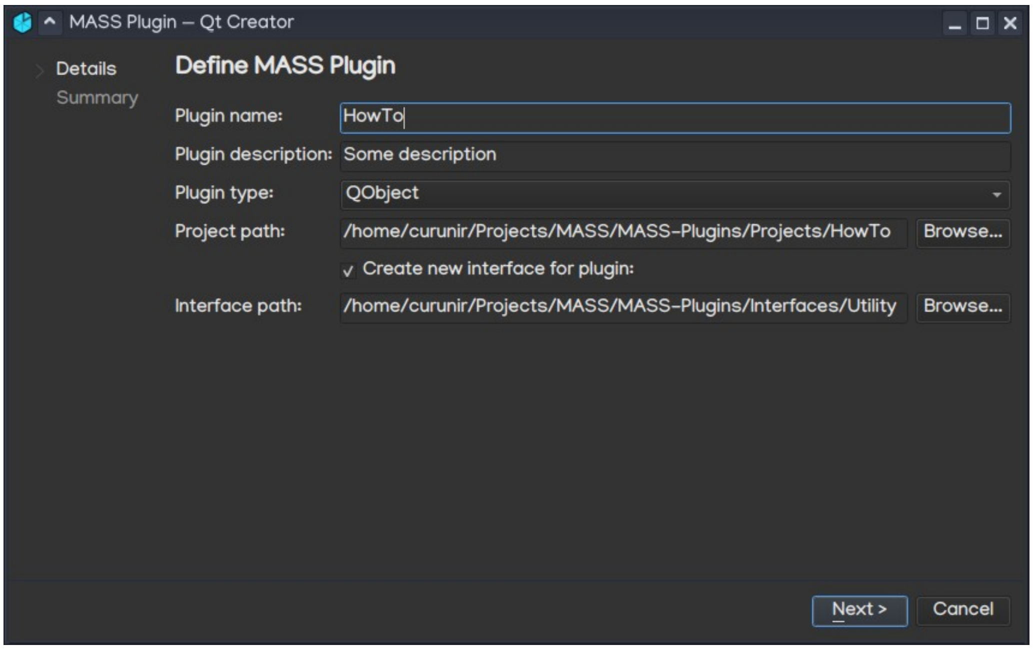
Select «New file or project...» and choose «MASS plugin» project.



Creates plugin for MASS.

Supported Platforms: Android Device Desktop

Set project fields.



MASS Plugin – Qt Creator

Details
Summary

Define MASS Plugin

Plugin name: HowTo
Plugin description: Some description
Plugin type: QObject
Project path: /home/curunir/Projects/MASS/MASS-Plugins/Projects/HowTo
✓ Create new interface for plugin:
Interface path: /home/curunir/Projects/MASS/MASS-Plugins/Interfaces/Utility

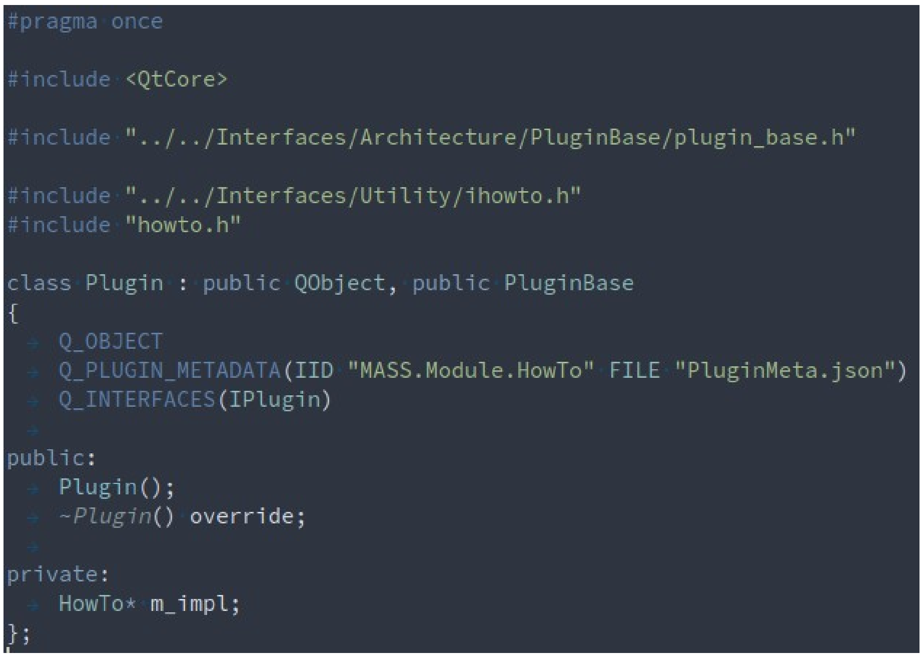
Name	Description
Class name	Name of your plugin
Plugin type	Possible types: • QtObject – it's business logic plugin and it won't contain any UI. • QWidget – it's UI plugin that uses QWidgets. Pick if you want to use QWidgets. • QWidget with QML – it's UI plugin with QWidget that wraps QML. Pick this if you want to use QML.
Project path	Path where plugin files will be created. Create new folder inside "MASS/MASS-Plugins/Projects" directory and set path to this folder.
Create new interface for plugin	Toggle if you want to create new interface for your plugin
Interface path	Path where interface file will be created. Pick one of folders inside "MASS/MASS-Plugins/Interfaces" directory.

Hit "Next" and "Finish".

In result you'll have these files:

- plugin.h
- plugin.cpp
- PluginMeta.json
- ihowto.h (if you've chosen "Create new interface for plugin")
- howto.h (if you've chosen "Create new interface for plugin")
- howto.cpp (if you've chosen "Create new interface for plugin")

Plugin header



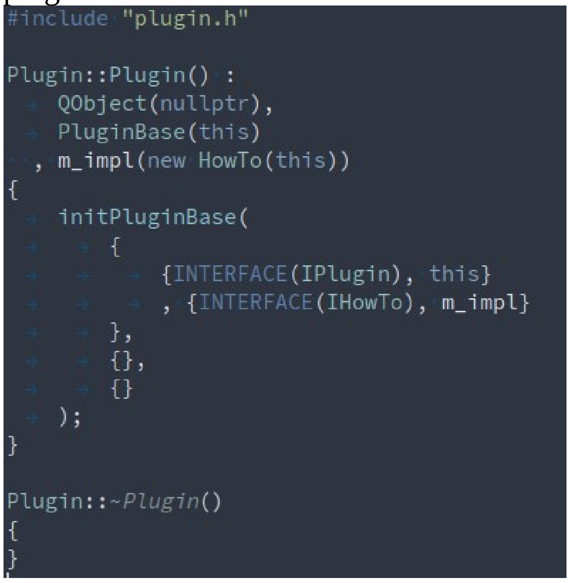
```cpp
#pragma once

#include <QtCore>

#include "../../Interfaces/Architecture/PluginBase/plugin_base.h"

#include "../../Interfaces/Utility/ihowto.h"
#include "howto.h"

class Plugin : public QObject, public PluginBase
{
	Q_OBJECT
	Q_PLUGIN_METADATA(IID "MASS.Module.HowTo" FILE "PluginMeta.json")
	Q_INTERFACES(IPlugin)
	
public:
	Plugin();
	~Plugin() override;
	
private:
	HowTo* m_impl;
};
```

You can see 3 included paths:

- plugin_base.h – file that provides PluginBase class from which your Plugin is inherited. It's purpose is to provide management of references to other plugins (we'll get to this);
- ihowto.h – file with interface that you've created for your plugin. All plugins interact through interfaces, so other plugin will interact with you through yours;
- howto.h – implementation of class that inherits from your plugin interface. You may ask why Plugin class doesn't inherited from this interface – it's made that way because of 2 reasons:
  - if your Plugin class will be inherited from 2 interfaces that have same method name and signature, you'll have no way to implement them separately inside Plugin class;
  - Qt prohibits multiple inheritance of QObject-based classes: https://doc.qt.io/archives/qq/qq15-academic.html

Plugin source

Also you have source file of your plugin.



```cpp
#include "plugin.h"

Plugin::Plugin() :
	QObject(nullptr),
	PluginBase(this)
  , m_impl(new HowTo(this))
{
	initPluginBase(
		{
			{INTERFACE(IPlugin), this}
			, {INTERFACE(IHowTo), m_impl}
		},
		{},
		{}
	);
}

Plugin::~Plugin()
{
}
```

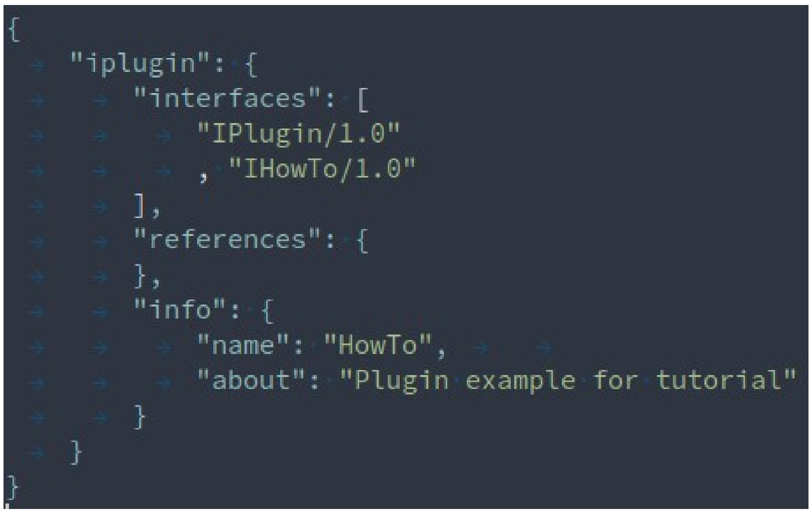
Most important thing here is initPluginBase() method. It initializes all runtime information for your plugin:

- first parameter – QMap with plugin interfaces, where key is Interface instance (INTERFACE is macros that creates Interface instance) and value is instance that inherited from this interface;

- second parameter – QMap with plugin references, where key is Interface instance and value is ReferenceInstancePtr (it's used to handle references to other plugins);

- third parameter – QMap with lists of plugin references, where key is Interface instance and value is ReferenceInstancesListPtr that holds list of ReferenceInstancePtr (it used if your plugin requires more than one reference of specific interface).

Plugin meta data



```
{
	"iplugin": {
		"interfaces": [
			"IPlugin/1.0"
			, "IHowTo/1.0"
		],
		"references": {
		},
		"info": {
			"name": "HowTo",
			"about": "Plugin example for tutorial"
		}
	}
}
```

Plugin meta data is JSON file that describes your plugin. Someone could opened without instantiating your plugin to decide that they need it or not. It has these fields:

- interfaces – list of interfaces that your plugin provides. Number after "/" stands for version of interface;
- references – map of references (we'll get to this);
- info – human readable info about plugin.

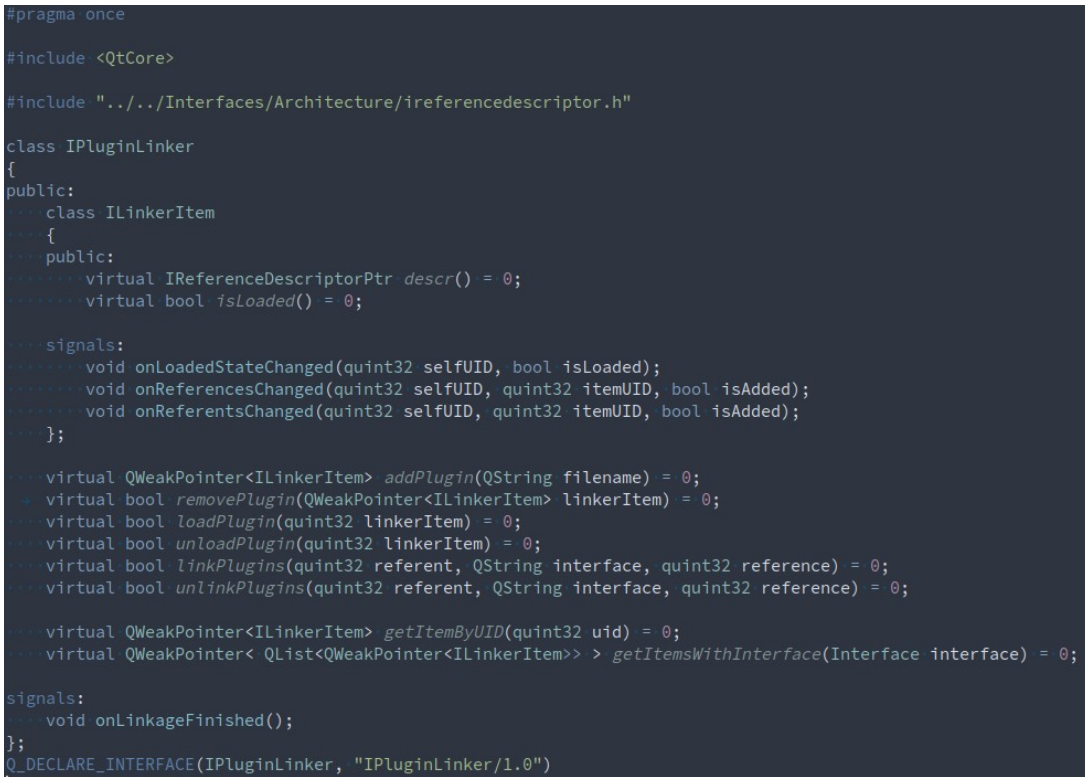
Adding plugin reference

Now, when we've covered basics, we can check how to add reference to our plugin.

First you need to pick interface of reference that you want to use. For now it's manual process – you need to search for it inside "Interfaces" directory, but later on we'll add tool for searching interfaces and plugins based on your requirements.

Let's assume that you've picked IPluginLinker interface – it can provide you with list of plugins that currently loaded in the app.

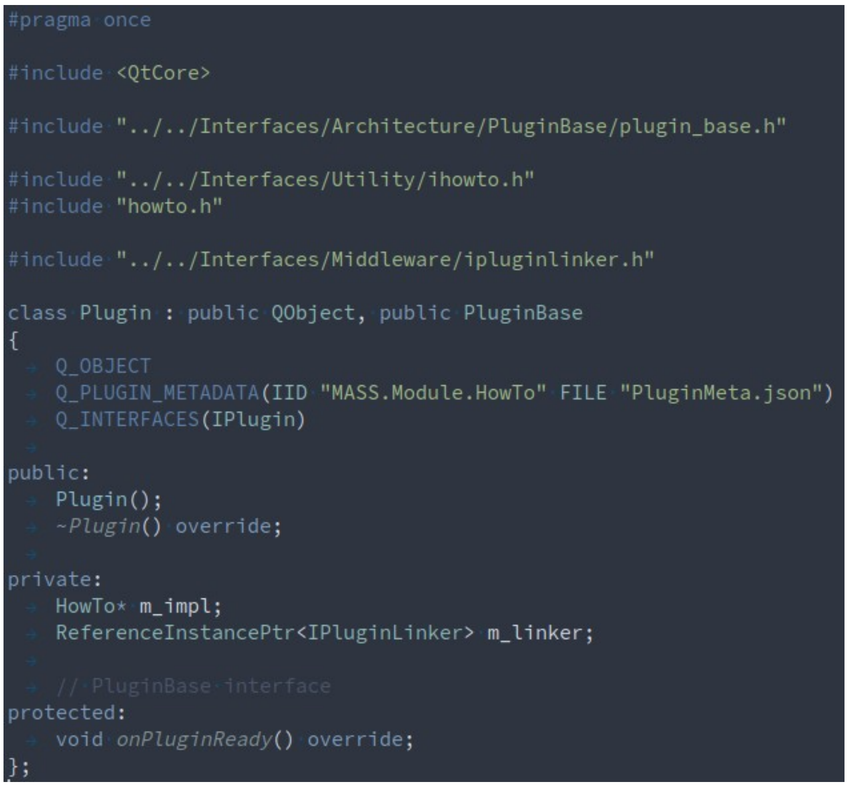
IPluginInterface file is here: "MASS/MASS-Plugins/Interfaces/Middleware/ipluginlinker.h". Here is how it looks like:



```cpp
#pragma once

#include <QtCore>

#include "../../Interfaces/Architecture/ireferencedescriptor.h"

class IPluginLinker
{
public:
    class ILinkerItem
    {
    public:
        virtual IReferenceDescriptorPtr descr() = 0;
        virtual bool isLoaded() = 0;

    signals:
        void onLoadedStateChanged(quint32 selfUID, bool isLoaded);
        void onReferencesChanged(quint32 selfUID, quint32 itemUID, bool isAdded);
        void onReferentsChanged(quint32 selfUID, quint32 itemUID, bool isAdded);
    };

    virtual QWeakPointer<ILinkerItem> addPlugin(QString filename) = 0;
    virtual bool removePlugin(QWeakPointer<ILinkerItem> linkerItem) = 0;
    virtual bool loadPlugin(quint32 linkerItem) = 0;
    virtual bool unloadPlugin(quint32 linkerItem) = 0;
    virtual bool linkPlugins(quint32 referent, QString interface, quint32 reference) = 0;
    virtual bool unlinkPlugins(quint32 referent, QString interface, quint32 reference) = 0;

    virtual QWeakPointer<ILinkerItem> getItemByUID(quint32 uid) = 0;
    virtual QWeakPointer< QList<QWeakPointer<ILinkerItem>> > getItemsWithInterface(Interface interface) = 0;

signals:
    void onLinkageFinished();
};
Q_DECLARE_INTERFACE(IPluginLinker, "IPluginLinker/1.0")
```

To add reference of this interface you need make changes in header, source and meta data of your plugin.

Plugin header changes



```cpp
#pragma once

#include <QtCore>

#include "../../Interfaces/Architecture/PluginBase/plugin_base.h"

#include "../../Interfaces/Utility/ihowto.h"
#include "howto.h"

#include "../../Interfaces/Middleware/ipluginlinker.h"

class Plugin : public QObject, public PluginBase
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID "MASS.Module.HowTo" FILE "PluginMeta.json")
    Q_INTERFACES(IPlugin)

public:
    Plugin();
    ~Plugin() override;

private:
    HowTo* m_impl;
    ReferenceInstancePtr<IPluginLinker> m_linker;

    // PluginBase interface
protected:
    void onPluginReady() override;
};
```

Include interface file in your plugin header and add attribute of ReferenceInterfacePtr to your Plugin class.

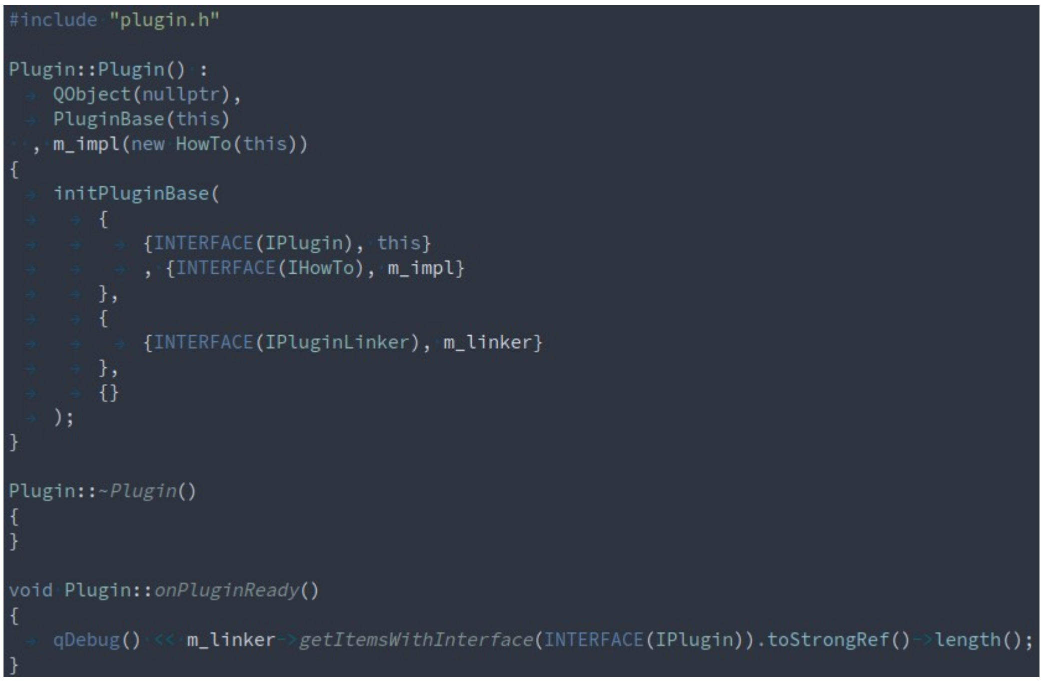
Because references are always set dynamically to plugin, you can’t just use them straightaway. Instead you need special state, when you’ll be able to use them. All states that plugin is passed represented by following methods of PluginBase that you can override:

- onInited – called when your plugin will be instantiated and assigned to id that uniquely identifies it among all other plugins;

- onReferencesSet – called when all references for your plugin is set, but none of them is not ready;
- onReady – called when your references are ready and you can start to call them;
- onReferencesListUpdated – called if your plugin require multiple references of same interface. Each call marks that new plugins were added to your ReferenceInstancesListPtr.

For purpose of this tutorial we’ll use only onReady method, so we’ll call some method of our reference.

Plugin source changes

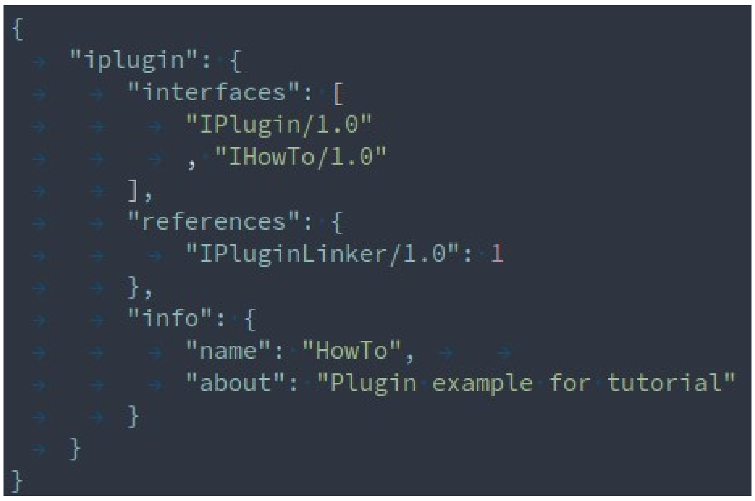


```cpp
#include "plugin.h"

Plugin::Plugin() :
	QObject(nullptr),
	PluginBase(this)
  , m_impl(new HowTo(this))
{
	initPluginBase(
		{
			{INTERFACE(IPlugin), this}
			, {INTERFACE(IHowTo), m_impl}
		},
		{
			{INTERFACE(IPluginLinker), m_linker}
		},
		{}
	);
}

Plugin::~Plugin()
{
}

void Plugin::onPluginReady()
{
	qDebug() << m_linker->getItemsWithInterface(INTERFACE(IPlugin)).toStrongRef()->length();
}
```

 ReferenceInterfacePtr instance into initPluginBase.

Inside overridden onPluginReady method we will output total count of plugins that inherits IPlugin interface (common interface for all plugins).

Meta data changes



```json
{
	"iplugin": {
		"interfaces": [
			"IPlugin/1.0"
			, "IHowTo/1.0"
		],
		"references": {
			"IPluginLinker/1.0": 1
		},
		"info": {
			"name": "HowTo",
			"about": "Plugin example for tutorial"
		}
	}
}
```

Add reference interface to “references” map.

Key must be interface with version, that you can pick from interface file in Q_DECLARE_INTERFACE macro (check at the bottom of the “MASS/MASS-Plugins/Interfaces/Middleware/ipluginlinker.h” file).

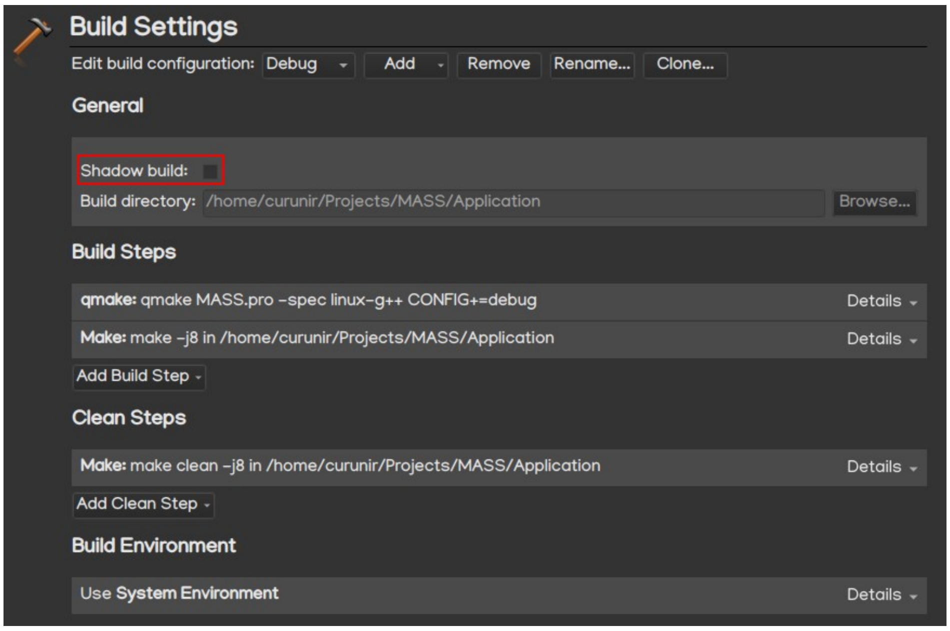
Value is number that represents how many references of this interface your plugin require, it could be:

- 0 – means that references value isn’t specified and that any amount (from zero to 2147483647) of references could be added to your plugin (use ReferenceInstancesListPtr for it);

- 1 – means that your plugin requires only one reference of this interface (use ReferenceInterfacePtr for it);
- from 2 to 2147483647 – means that your plugin requires specific amount of references that you’ve set (use ReferenceInstancesListPtr for it).

Running your plugin

Open “MASS/Application/MASS.pro” project. On opening of “MASS.pro” toggle off “Shadow build” setting.



Build Settings

Edit build configuration: Debug | Add | Remove | Rename... | Clone...

General

Shadow build: ☐

Build directory: /home/curunir/Projects/MASS/Application | Browse...

Build Steps

qmake: qmake MASS.pro -spec linux-g++ CONFIG+=debug | Details

Make: make -j8 in /home/curunir/Projects/MASS/Application | Details

Add Build Step

Clean Steps

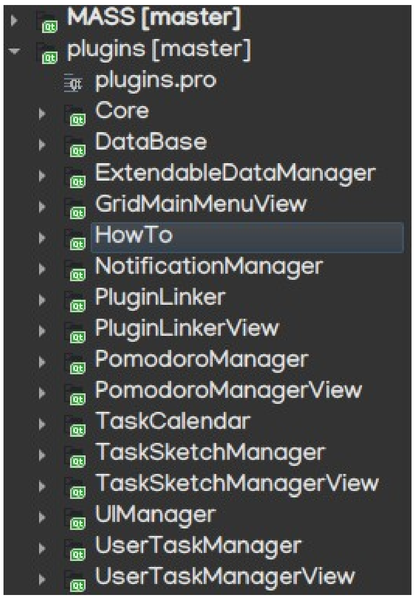
Make: make clean -j8 in /home/curunir/Projects/MASS/Application | Details

Add Clean Step

Build Environment

Use System Environment | Details

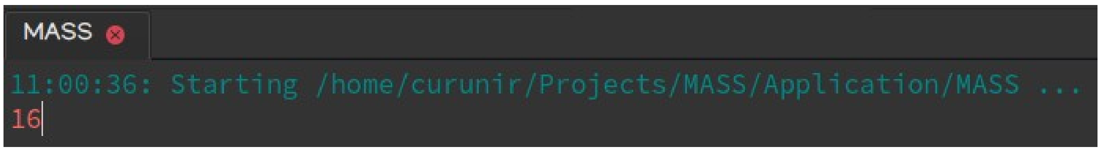
Open “MASS/MASS-Plugins/plugins.pro” project, you’ll see that your plugin is among other plugins.



```
MASS [master]
plugins [master]
    plugins.pro
    Core
    DataBase
    ExtendableDataManager
    GridMainMenuView
    HowTo
    NotificationManager
    PluginLinker
    PluginLinkerView
    PomodoroManager
    PomodoroManagerView
    TaskCalendar
    TaskSketchManager
    TaskSketchManagerView
    UIManager
    UserTaskManager
    UserTaskManagerView
```

Run building of plugins project, you’ll see that plugin libraries will appear inside “MASS/Application/Plugins” directory.

Run MASS application, you’ll see your plugin output.



```
MASS
11:00:36: Starting /home/curunir/Projects/MASS/Application/MASS ...
16
```