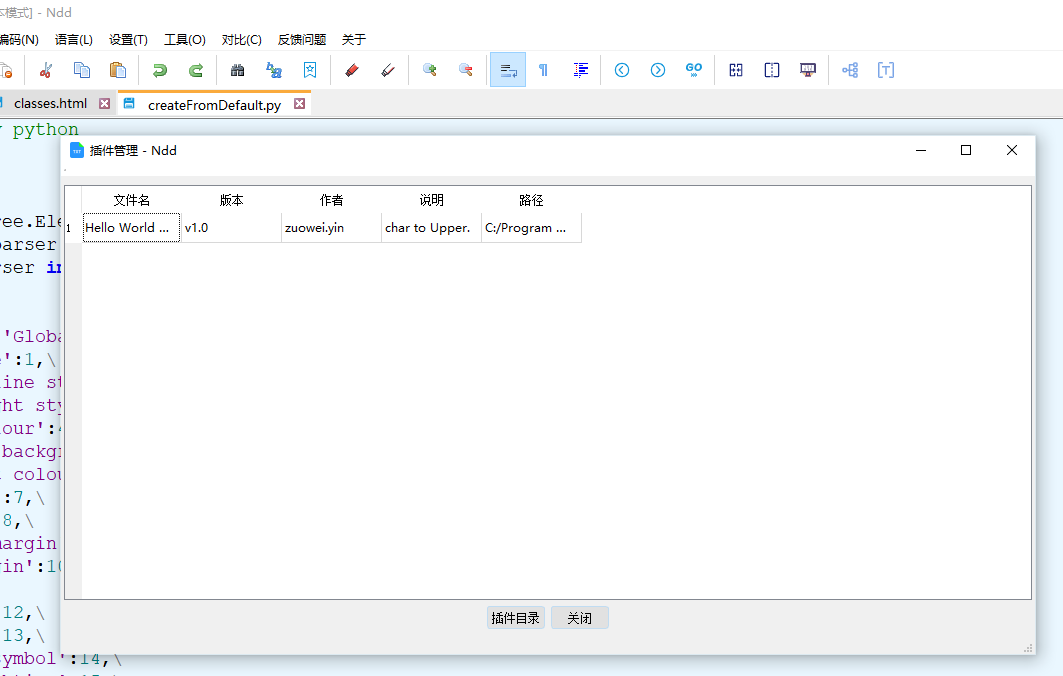
# Ndd 插件编程开发说明

|  |  |  |
| --- | --- | --- |
| 修改记录 | 说明 | 日期 |
| 1.21版本第一次支持插件 | 初次提交 |  |
| 1.22.2 版本扩展插件接口 | 支持插件二级菜单的创建 | 2023/02/13 |
| 2.0 版本扩展插件接口 | 插件可发消息到主程序执行命令 | 2023/05/13 |

从Notepad-- 1.21.0 开始，ndd提供了一个动态编译的版本，可以支持插件开发。如下图所示：



下面我们介绍，如何进行Ndd的插件开发。

## 什么样的功能，适合使用插件来完成。

请看下面的逻辑图，目前插件主要用来做文字处理工作。插件的流程有如下四步：

1. 第一步：从主程序获取当前编辑框对象。
2. 第二步：从当前编辑框对象读取文本。
3. 第三步：插件对文本进行处理，这里也是插件的主要工作。
4. 第四步：插件把处理后的结果问题，回填入当前编辑框对象。



*基于以上流程，什么样的功能适合插件来做？***主要是针对文字处理的功能。**举例来说，要做一个把繁体转换为简体，或者简体转换为繁体的工作，则比较适合插件来完成。再举例，要做一个把全码转换为半码，或者半码转换为全码的文字处理功能，也比较适合使用插件来完成。总体而言，如果与NDD主程序交互的工作相对简单，只有一个来回的获取文字、回填文字的流程工作，都比较适合使用插件来完成。

对于文字的处理，无论其做多么复杂的工作，都是插件本身来完成。

再举一个例子，经常有人说，要针对xml和json做格式化处理，加密解密处理，那么这个功能虽然比较复杂，但是其本身也非常适合使用插件功能来完成。可以武断的说，几乎百分之90以上的小功能，都可以使用插件工作来完成。

哪些工作不适合使用插件来完成？反之，如果您的工作重点不在于文本处理，而在于用户交互，则可能不适合插件来完成，或者可以实现，但是会比较麻烦。

## 二、如何编程开发插件

### 2.1 加载工程库

从gitee下载代码后，分别使用vs或qtcreator打开如下两个pro工程：

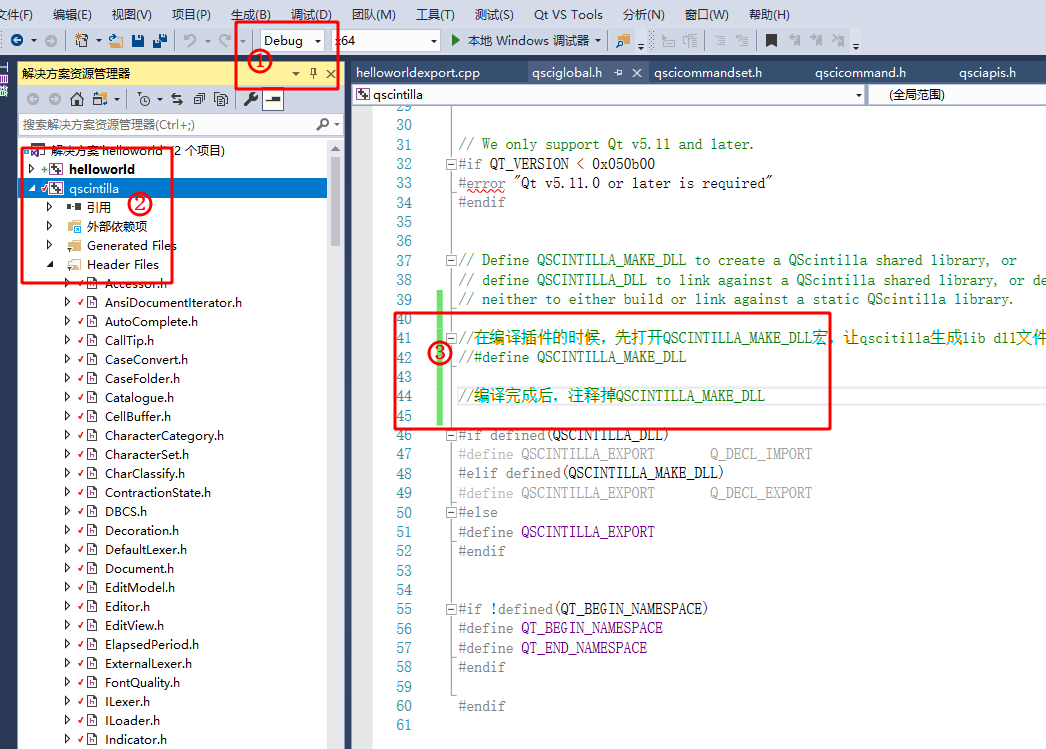
1)notepad--\src\qscint\src\qscintilla.pro

2) notepad--\src\plugin\helloworld\helloworld.pro

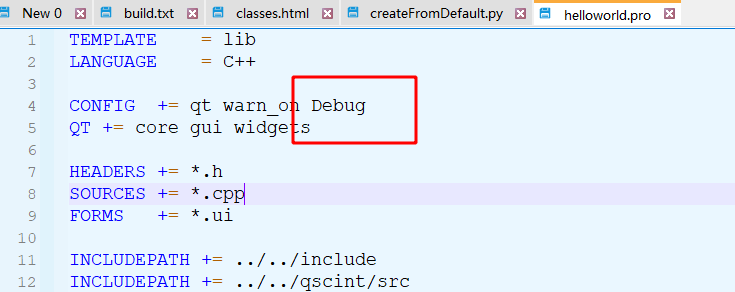
插件只依赖qscintilla库，不依赖ndd的库。

这一步，我们假定你已经对qt的配置环境比较熟悉，对qt中的基本配置环境，比如qtcreator中qt环境的配置，或者vs qtvstool中qt的配置，已经比较熟悉。如果您不熟悉这一步，建议还是自己先写一些qt基本的hello world方面的例子，把这一步补上再说。

下面给出vs加载两个工程后的样子。



按照上图，我们在Debug模式下做开发。如果是release模式，则pro中对应的如下配置，要修改为Release，注意qscintilla.pro 和helloworld.pro两个pro都要修改。我们下面的举例，都是假定在Debug模式下面做开发。



### 推荐的开发环境：

推荐vs2019 + qt515.2。

目前我本是就是使用的vs2019和qt515.2。QT官方到今天（20230107），能够下载的最新的QT5就是QT515.2。而据我所知，有些QT的bug在qt515.7中做了解决，但是官方没有提供qt515.7的在线安装包。如果你也使用QT515.7制作插件，则您的插件，大概率的不用做任何修改，可以直接加载到NDD官方发布的NDD主程序中。

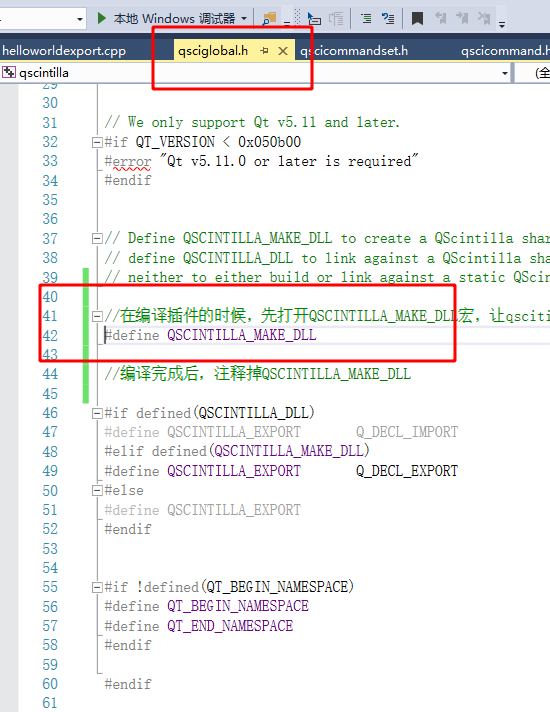
如果你使用的其他QT版本，问题也不大。您只需要手动自己编译NDD程序，即可加载您的插件。只要您的插件开源，而且用户反响较大，我们会在发布NDD的时候，主动打包发布您的插件。所以其实不用太担心QT 版本兼容的问题。

### 编译过程

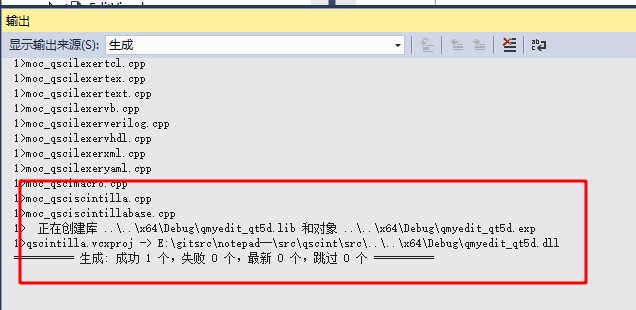
#### 第一步：先编译notepad--\src\qscint\src\qscintilla.pro

修改下面文件，打开宏#define QSCINTILLA\_MAKE\_DLL，然后编译qscintilla.pro

如果你不打开这个宏，则无法得到编译的LIB文件，只有dll文件。

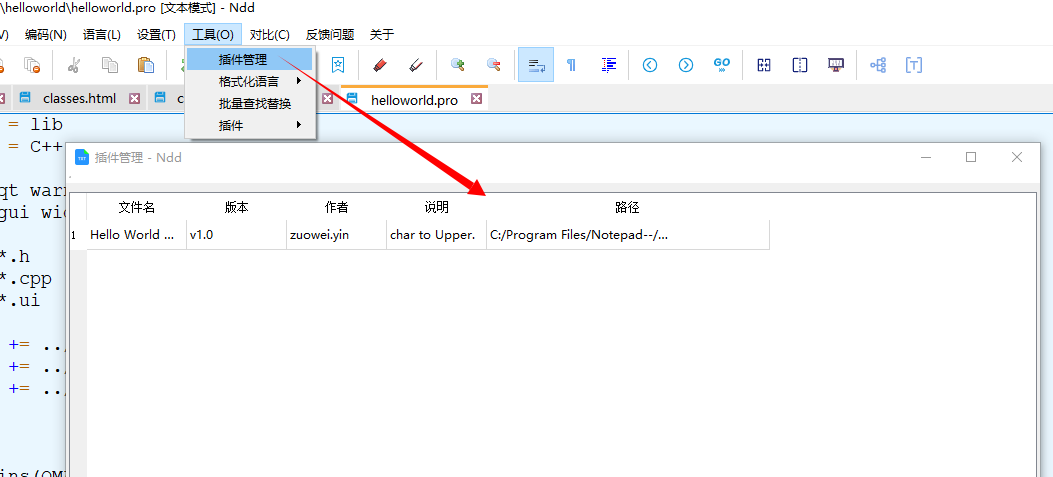


编译完成后，将在如下目录生成出对应的lib 和 dll 库。请到该目录下检查，是否有对应的生成库，如果没有，则说明哪里配置没有对。



### 第二步：编译notepad--\src\plugin\helloworld\helloworld.pro

这个helloworld.pro的功能很简单，就是把当前编辑框的内容，做一个大小写的转换。其加载在NDD中的界面如下：

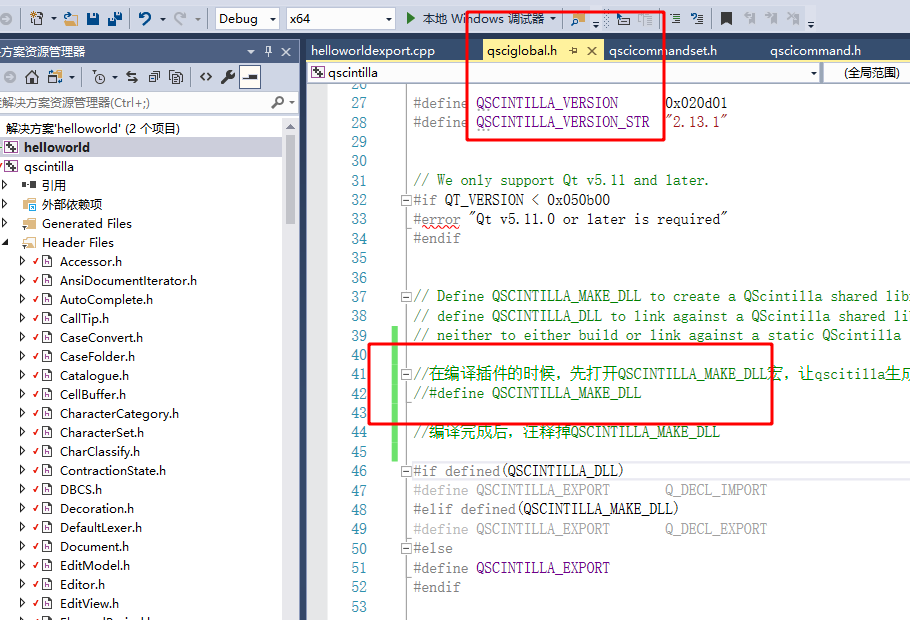


**注意：在编译helloworld.pro，务必要关闭如下文件中的宏//#define QSCINTILLA\_MAKE\_DLL。**

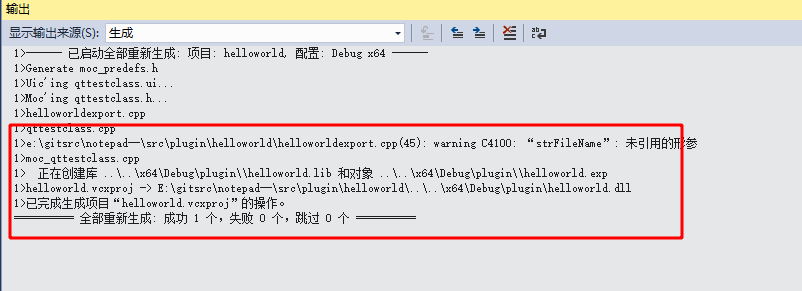
这个宏在编译QSCINTILLA的时候起作用，告诉编译器，要生成导出的LIB文件。

但是其它第三库依赖该头文件的时候，要注释该宏，因为其他库是把该库做一个导入DLL使用的。

***如果不注释，则可能报错。如下图所示：这一步千万不能忘记！***



Helloworld.dll编译成功后，会在如下目录生成对应的dll文件。此时例子插件就已经编译出来了。打开你编译的NDD主程序，则可以加载到该插件。如何编译ndd主程序，在另外一个文档“编译说明”中有讲解。我们建议都在Debug模式下做开发，最后发布的时候，统一切换到Release模式。



如果您熟悉了以上例子，则自己制作新功能的时候，只需要把helloworld的例子拷贝一份，然后改名，则可以在其基础上制作新的插件了。

## 三、对插件代码的分析和讲解

### 3.1 简单插件例子（无需创建二级菜单）

见例子helloword插件的插件文件helloworldexport.cpp中的主函数：

//插件的入口点函数

//则点击菜单栏按钮时，会自动调用到该插件的入口点函数。

//pNotepad:就是CCNotepad的主界面指针

//strFileName:当前插件DLL的全路径，如果不关心，则可以不使用

//getCurEdit:从NDD主程序传递过来的仿函数，通过该函数获取当前编辑框操作对象QsciScintilla

//s\_invokeMainFun: 可以回调NDD主程序中的功能函数，比如创建新文件功能等，根据需要可实时扩展。

//pProcData:如果pProcData->m\_menuType = 0 ,则该指针为空；如果pProcData->m\_menuType = 1，则该指针有值。目前需要关心s\_procData.m\_rootMenu

//开发者可以在该菜单下面，自行创建二级菜单

int NDD\_PROC\_MAIN(QWidget\* pNotepad, const QString &strFileName, std::function<QsciScintilla\*()>getCurEdit, std::function<bool(int, void\*)> pluginCallBack, NDD\_PROC\_DATA\* pProcData)

{

//对于不需要创建二级菜单的例子，pProcData总是nullptr。

//该函数每次点击插件菜单时，都会被执行。

QsciScintilla\* pEdit = getCurEdit();

if (pEdit == nullptr)

{

return -1;

}

//务必拷贝一份pProcData，在外面会释放。

if (pProcData != nullptr)

{

s\_procData = \*pProcData;

}

s\_pMainNotepad = pNotepad;

s\_getCurEdit = getCurEdit;

s\_invokeMainFun = pluginCallBack; //通过该函数给NDD主程序发送消息，如此插件可回调主程序功能。比如创建一个新的文件等。

//如果pProcData->m\_menuType = 1;是自己要创建二级菜单的场景。则通过s\_procData.m\_rootMenu 获取该插件的菜单根节点。

//插件开发者自行在s\_procData.m\_rootMenu下添加新的二级菜单项目

//做一个简单的转大写的操作

QtTestClass\* p = new QtTestClass(pNotepad,pEdit);

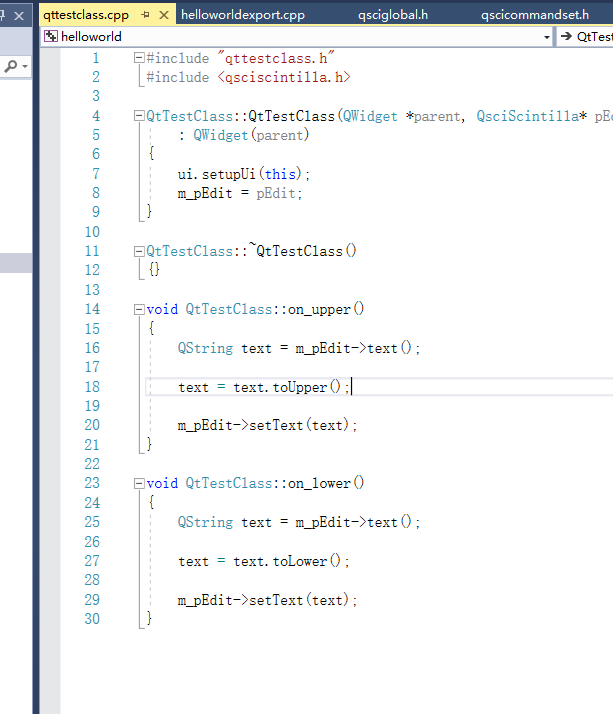
//主窗口关闭时，子窗口也关闭。避免空指针操作

p->setWindowFlag(Qt::Window);

p->show();

return 0;}

而qttestclass.cpp中的代码非常简单，就是一个大小写转换，然后回填编辑器窗口的过程，我们就不再详细讲解，贴图如下：



插件作者、名称、功能、版本号的自定义，如下函数，自行修改为您插件的相应说明就行。

bool NDD\_PROC\_IDENTIFY(NDD\_PROC\_DATA\* pProcData)

{

if(pProcData == NULL)

{

return false;

}

pProcData->m\_strPlugName = QObject::tr("Hello World Plug"); //插件名称

pProcData->m\_strComment = QObject::tr("char to Upper."); //功能说明

pProcData->m\_version = QString("v1.0"); //版本号

pProcData->m\_auther = QString("zuowei.yin"); //作者名称

//对于不需要创建二级菜单的插件，这里需要给0

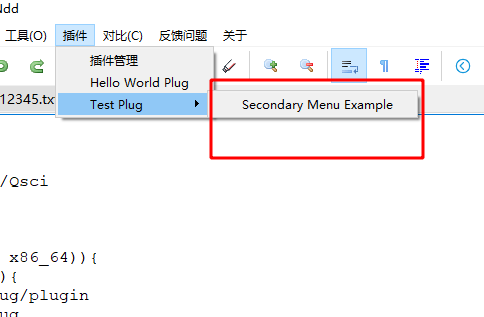
pProcData ->m\_menuType = 0;

return true;

}

### 3.2 二级菜单插件例子

有些插件功能可能比较复杂，需要自定义创建多级菜单。如下图test例子所示：



上图中，除了Test Plug是一个菜单，用户还需要在下面自定义二级子项目，比如图中“Secondary Menu Example”的二级子菜单。Test例子就是这种情况。

这种情况要稍微复杂一点。

入口点函数例子：

//则点击菜单栏按钮时，会自动调用到该插件的入口点函数。

//pNotepad:就是CCNotepad的主界面指针

//strFileName:当前插件DLL的全路径，如果不关心，则可以不使用

//getCurEdit:从NDD主程序传递过来的仿函数，通过该函数获取当前编辑框操作对象QsciScintilla

//s\_invokeMainFun: 可以回调NDD主程序中的功能函数，比如创建新文件功能等，根据需要可实时扩展。

//pProcData:如果pProcData->m\_menuType = 0 ,则该指针为空；如果pProcData->m\_menuType = 1，则该指针有值。目前需要关心s\_procData.m\_rootMenu

//开发者可以在该菜单下面，自行创建二级菜单

int NDD\_PROC\_MAIN(QWidget\* pNotepad, const QString& strFileName, std::function<QsciScintilla\* ()>getCurEdit, std::function<bool(int, void\*)> pluginCallBack, NDD\_PROC\_DATA\* pProcData)

{

//对于需要创建二级菜单的例子，pProcData不能为nullptr。s\_procData.m\_rootMenu是根菜单节点，由主程序进行创建。用户自定义的菜单，都在该根菜单之下自行新增。

//该NDD\_PROC\_MAIN函数只会调用一次。所以在唯一调用之下，务必要把参数中的值保存一份。否则后续再也没有机会执行该函数。

//务必拷贝一份pProcData，函数返回后，在外面会释放。

if (pProcData != nullptr)

{

s\_procData = \*pProcData;

}

else

{

return -1;

}

s\_pMainNotepad = pNotepad;

s\_getCurEdit = getCurEdit;

s\_invokeMainFun = pluginCallBack;

//上面pProcData、pMainNotepad、s\_getCurEdit三个值，务必要保存起来。因为后面不会再有机会执行该函数。如果后续业务功能中需要这些值，只能通过操作保存的副本进行。

//如果pProcData->m\_menuType = 1;是自己要创建二级菜单的场景。则通过s\_procData.m\_rootMenu 获取该插件的菜单根节点。

//插件开发者自行在s\_procData.m\_rootMenu下添加新的二级菜单项目

QAction\* pAction = new QAction(s\_procData.m\_strPlugName, s\_procData.m\_rootMenu);

s\_procData.m\_rootMenu->addAction(pAction);

pAction->setText("Secondary Menu Example");

//后续就是自行绑定pAction的执行槽函数。后面通过点击菜单上面的按钮，进行槽函数的//触发调用。

return 0;

}

static NDD\_PROC\_DATA s\_procData;

static QWidget\* s\_pMainNotepad = nullptr;

std::function<QsciScintilla\* ()> s\_getCurEdit;

bool NDD\_PROC\_IDENTIFY(NDD\_PROC\_DATA\* pProcData)

{

if(pProcData == NULL)

{

return false;

}

pProcData->m\_strPlugName = QObject::tr("Test Plug");

pProcData->m\_strComment = QObject::tr("char to lower.");

pProcData->m\_version = QString("v1.0");

pProcData->m\_auther = QString("zuowei.yin");

pProcData->m\_menuType = 1; //二级菜单插件该值给1。

return true;

}

更加详细的完整例子，请参考插件工程的 jstool例子，该例子移植了vscode中的js 格式化插件的功能。

## 四、关于兼容性和ABI兼容性的说明

如果您编译出来的插件，放在官方发布的NDD插件目录下，无法加载，这多半是兼容性出了问题。有如下两方面的问题。

1 QT兼容性的问题。目前我们发布NDD插件版本，使用的是QT515.2。如果你也使用该QT版本，则可以避免QT不一致问题。如果你使用其他QT版本，则可能存在兼容性问题。

2 QScintilla 库的ABI兼容性问题。qmyedit\_qt5.dll 是QScintilla编译出来的结果库。NDD需要依赖Qscintilla库，而插件也需要依赖Qscintilla库。如果哪天NDD修改了Qscintilla库，比如增加了虚函数或成员变量，则会造成Qscintilla的修改而ABI无法兼容。此时您开发的插件，也需要更新Qscintilla后，同步编译一下。我们会尽量保证Qscintilla的ABI兼容性，避免导致插件的不兼容性。*注意：官方发布NDD插件库的时候，会发布qmyedit\_qt5.dll和qmyedit\_qt5.lib两个库，您其实可以不自己编译qmyedit\_qt5库，直接使用官方发布的qmyedit\_qt5.dll和qmyedit\_qt5.lib就行。*

兼容性其实不是一个很大的问题。只要您的插件反响比较大，NDD在发布程序时，会主动编译并发布您的插件，这样可以避免插件不兼容问题。

## 五、关于Qscintilla的说明

在helloworld的插件例子中，只使用了Qscintilla的两个基本API，一个是获取编辑框内容text;另外一个是回写入编辑框内容setText()。至于Qscintilla提供哪些详细的API，则需要查看官方的手册，代码中的路径是notepad--\src\qscint\doc\Scintilla，其中最主要的文档是notepad--\src\qscint\doc\Scintilla\ScintillaDoc.html。

QsciScintilla类提供了大量的文本操作API接口，比如获取文本，写入文本，替换文本、选择某一部分，删除某一部分等。这些基本的编辑器操作，都是由Qscintilla库来提供的。如果您要编写插件，有些需要操作编辑框的操作，那么需要自己阅读和熟悉Qscintilla库提供的功能。扩展Qscintilla或者调用其提供的API接口，务必要熟悉Qscintilla提供的基本操作API接口，这一步是无法跳过的。为了保证NDD和插件的兼容，Qscintilla里面的函数接口不能随便改动，如果改动，可能会造成ABI不兼容。

## 六、提交插件

如果您开发的插件，用户反响比较大，则我们会主动在插件主页上贴上您的插件项目。后续专门成立一个NDD的插件开源项目，所有插件都会在该项目上进行展示。如果您开发出了功能强大的插件，一定要通知我们，插件的所有权归您。