

Plugins TerraView

Versão TerraLib/TerraView: 3.3.1

Requisitos

- Código fonte completo da TerraLib na estrutura de diretórios sugerida no site da TerraLib¹.
- Código completo do TerraView na estrutura de diretórios sugerida no site do TerraView².
- Um compilador C++ apropriado para a plataforma escolhida:
 - Microsoft Visual Studio .NET 2003 para o ambiente Windows;
 - GNU GCC 3.2.3 ou superior para o ambiente Linux (veja o site do GNU para maiores informações³).
- QT 3.X.X (binários e ambiente de desenvolvimento - veja o site do Qt⁴ para maiores informações).

Construindo um Plugin “Hello World”

Nos tópicos a seguir serão mostrados os passos para a construção de um plugin que efetua uma interação simples com o banco de dados aberto pelo TerraView para adquirir a lista dos planos de informação nele presentes. Paralelamente aos passos mostrados serão apresentadas opções adicionais disponíveis para a construção de plugins mais complexos.

¹ Website TerraLib: www.terralib.org

² Website TerraView: www.dpi.inpe.br/terraview

³ GNU Website: <http://www.gnu.org/>

⁴ QT Website: <http://www.trolltech.com/>

Criando o projeto do Plugin

O suporte a plugins para o TerraView usa uma biblioteca especialmente voltada para o gerenciamento de plugins chamada LibSPL (disponibilizada na estrutura de diretórios da TerraLib - veja o site da LibSPL⁵ para maiores informações), que resolve questões de portabilidade e de codificação. Todos os plugins para o TerraView devem ser *linkados* com a SPL. Este *link* é feito automaticamente utilizando um arquivo de projeto QT base.

A utilização do projeto base implica que a estrutura de diretórios esteja disposta como mostrado na Figura 1 – Estrutura de diretórios da TerraLib, TerraView e Plugins – que é a mesma sugerida nos sites da TerraLib/TerraView. Os módulos “**terralibx**”, “**terralibw**”, “**terraView**” e “**src**” podem ser obtidos no *site* pelos arquivos compactados em ZIP ou TAR.GZ.

O módulo de plugins “**terraViewPlugins**” deve ser obtido pela utilização do CVS. No sistema Linux o pacote do CVS geralmente já é instalado por padrão, bastando utilizar o executável a partir do *shell* ou utilizar uma interface gráfica (como o **Cervisia**⁶ integrado ao KDE). No sistema windows pode-se instalar um cliente gráfico como o **TortoiseCVS**⁷, que se integra ao *Windows Explorer*. Para acessar o repositório da TerraLib utilizando algum destes clientes devem ser fornecidas as seguintes informações (sensíveis a letras maiúsculas):

- **Endereço do servidor:** cvs.dpi.inpe.br
- **Usuário:** anonymous
- **Diretório do repositório:** /home/terralib
- **Nome do módulo:** terraViewPlugins
- **Tag/Revisão:** v-3-3-1

⁵ Website LibSPL: <http://www.unitedbytes.de/go.php?site=spl>

⁶ Website Cervisia: <http://cervisia.kde.org/>

⁷ Website TortoiseCVS: <http://www.tortoise cvs.org/>

NOTA: É importante que as *tags* utilizadas ao obter os módulos do CVS (*src*, *terralibw*, *terralibx*, *terraView* e *terraViewPlugins*) sejam as mesmas. Caso contrário o projeto de plugins poderá gerar erros de compilação.

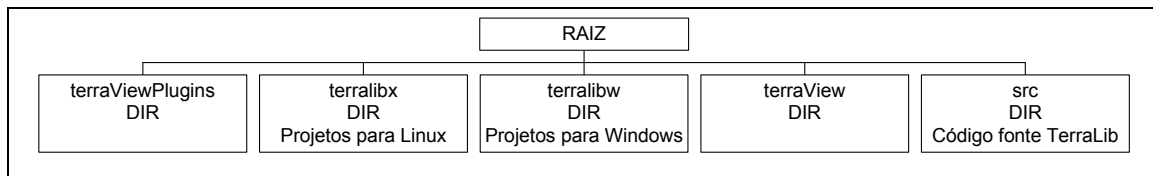


Figura 1 – Estrutura de diretórios da TerraLib, TerraView e Plugins

O arquivo do projeto base “**base.pro**” está incluído dentro do módulo “**terraViewPlugins**” no subdiretório “**base**”, como mostrado na Figura 3 - Diretório de código de plugins. Este arquivo deve ser referenciado pelo arquivo de projeto do novo plugin (**HelloWorld.pro**). Assim todas as questões relativas a dependências e inclusão de arquivos são resolvidas e os únicos arquivos que devem ser incluídos são aqueles específicos do plugin (arquivos de cabeçalho, arquivos de implementação de código, arquivos de interface, bibliotecas adicionais utilizadas, etc). O arquivo de projeto **HelloWold.pro** de geração do plugin, mostrado na Figura 2 pode ser criado com um editor de textos comum. Ele deve ser salvo no subdiretório “**HelloWorld**” que deve estar no mesmo nível do sub-diretório “**base**”, como mostrado na Figura 3 - Diretório de código de plugins.

NOTA: É importante observar que o nome do diretório onde o arquivo de projeto do plugin está deve ser igual ao do arquivo de projeto, observando-se inclusive os caracteres em maiúsculas e minúsculas. O mesmo vale para todos os outros nomes de arquivos citados dentro do projeto.

```
include( ../base/base.pro )
FORMS += HelloWorldPluginWindow.ui
HEADERS += HelloWorldPluginCode.h
SOURCES += HelloWorldPluginCode.cpp
```

Figura 2 – O arquivo HelloWorld.pro

Além da referência ao arquivo **base.pro** devem ser citados os demais arquivos do projeto do plugin, detalhados nos próximos tópicos. São eles:

- **HelloWorldPluginWindow.ui** – Arquivo com as definições da interface gráfica QT do plugin.
- **HelloWorldPluginCode.h** – Arquivo que contém as definições do plugin gerado, como nome, versão, etc.
- **HelloWorldPluginCode.cpp** – Contém a implementação das funções de entrada para o plugin.

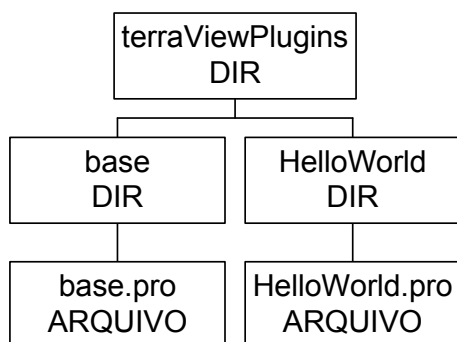


Figura 3 - Diretório de código de plugins

NOTA: Utilizando bibliotecas de Terceiros: Caso o plugin utilize outras bibliotecas de terceiros estas não deverão permanecer no diretório de plugins, pois poderão ocorrer problemas no momento da carga de bibliotecas. Preferencialmente elas deverão ser instaladas no devido diretório do sistema operacional ou, quando isto não for possível:

1. Para plataforma Windows:
 - a. Quando compilando/executando em modo *debug*: As bibliotecas de terceiros deverão estar presentes no mesmo do projeto do TeraView (mesmo diretório onde se localiza o arquivo de solução do TerraView – **terraView.sln**).

- b. Quando gerando/executando a versão de distribuição (*release*):
Deverão estar presentes no mesmo diretório do executável do TerraView.
2. Para plataforma Linux – Atualizar a variável de ambiente **LD_LIBRARY_PATH** para também apontar para o diretório das bibliotecas utilizadas além delas, preferencialmente, estar presentes no mesmo diretório do executável do TerraView (*release* ou *debug*).

Criando o arquivo das definições de interface (HelloWorldPluginWindow.ui)

Para a criação do arquivo de interface pode-se utilizar o aplicativo **QTDesigner**, fornecido no kit de desenvolvimento QT. Esta ferramenta possibilita não só a criação de interfaces como também a gerência do arquivo de projetos como um todo. Como o arquivo de projeto **HelloWorld.pro** já foi criado, esta ferramenta será utilizada somente para a criação da interface.

NOTA: Nunca permita que o aplicativo sobrescreva o arquivo **base.pro** pois seu conteúdo poderá ser alterado e problemas no momento da compilação poderão ocorrer.

O primeiro passo é criar uma nova janela que deverá ser derivada do tipo **QMainWindow** como mostrado na Figura 4- Criando uma nova janela no QTDesigner.

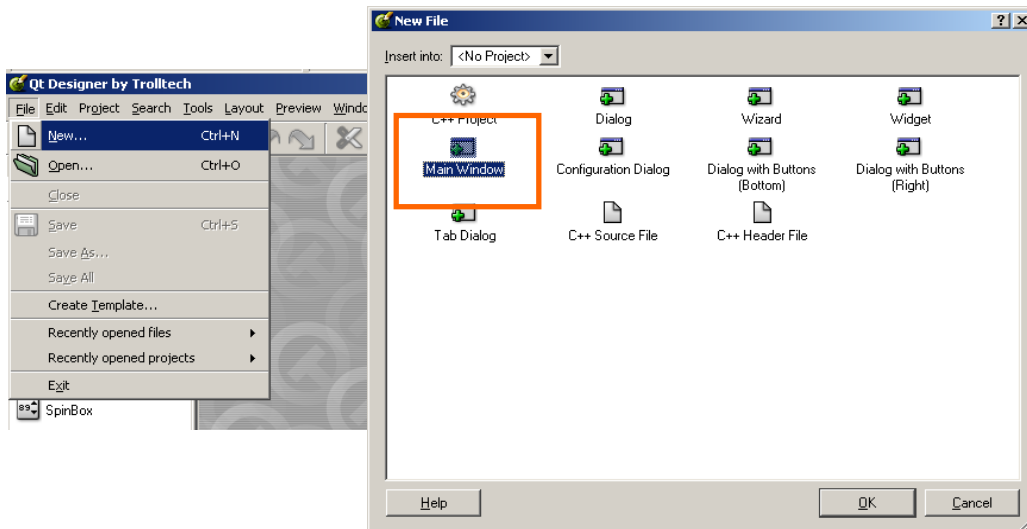


Figura 4- Criando uma nova janela no QTDesigner

Como queremos criar apenas uma janela vazia pode-se desativar as opções de criação automática de menus e barras de ferramentas que o aplicativo oferece, como mostrado na Figura 5 - Desativando as opções padrão.

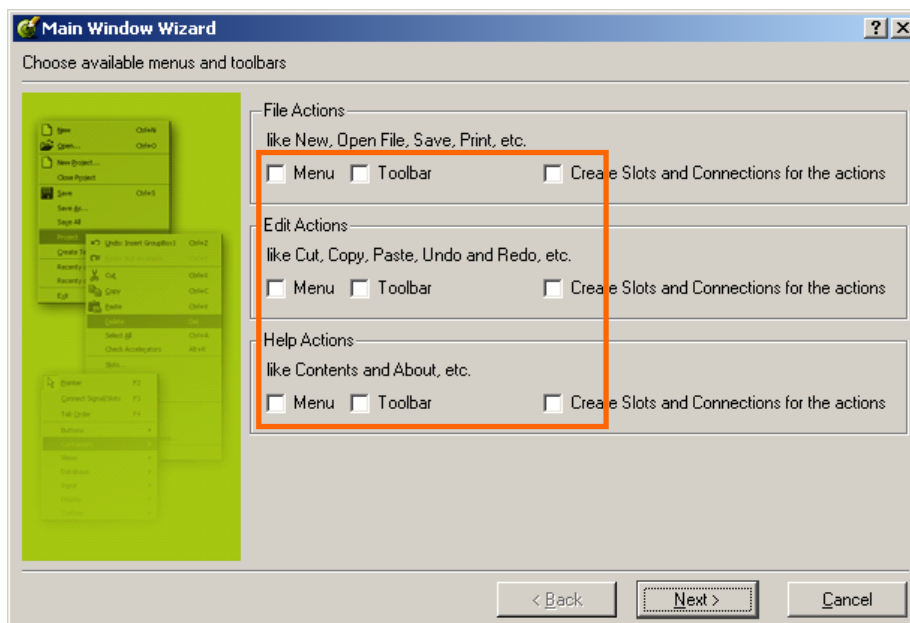


Figura 5 - Desativando as opções padrão

O nome da instância desta janela deverá ser modificado para **HelloWorldPluginWindow** como mostrado na Figura 6 - Renomeando a instância.

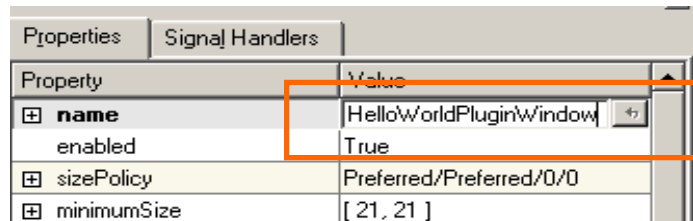


Figura 6 - Renomeando a instância

A seguir deve-se inserir uma **QcomboBox** que será preenchida e apresentará os nomes dos planos de informação lidos do banco de dados. Para tal basta selecionar “**ComboBox**” na caixa de objetos comuns à esquerda e em seguida clicar sobre a janela que foi criada em branco como mostrado na Figura 7 - Criando um QComboBox.

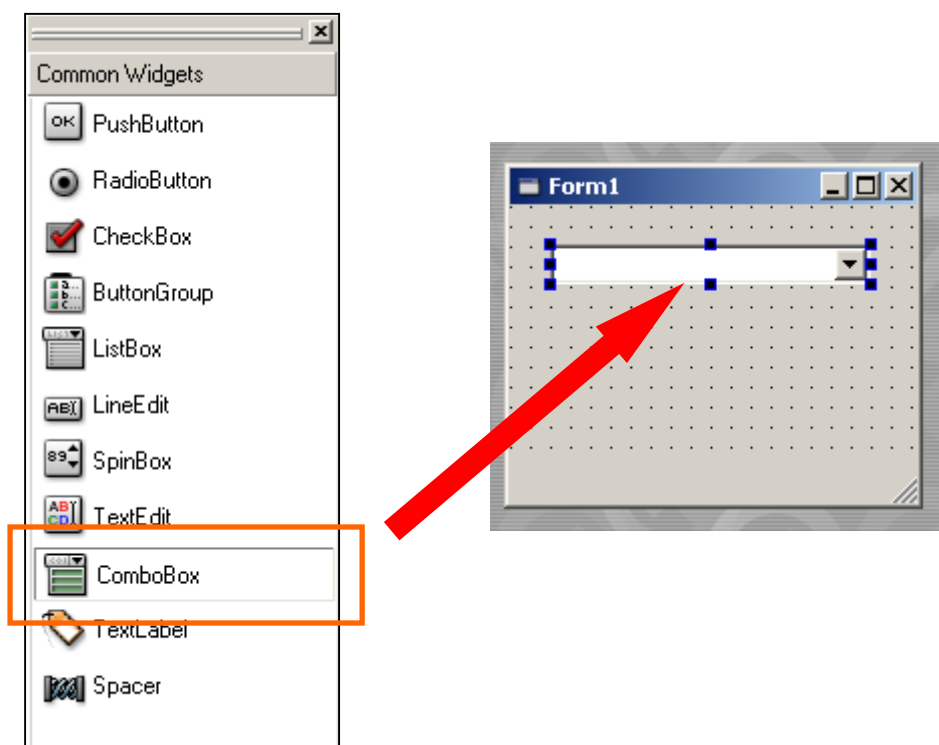


Figura 7 - Criando um QComboBox

Para que seja fácil identificar este novo objeto criado dentro da janela deve-se renomear a instância para “q_db_layers_” (nome sugerido), para tal basta selecionar o objeto criado na janela e alterar o campo de nome como mostrado na Figura 8 - Alterando o nome da instância QComboBox.

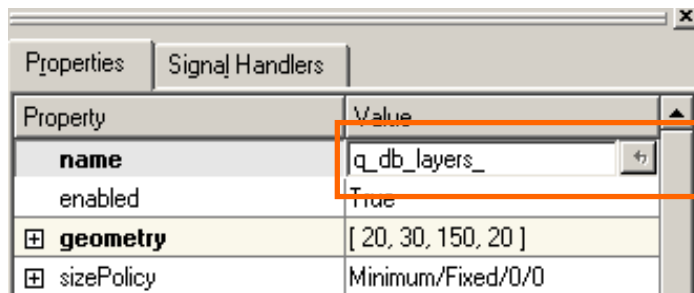


Figura 8 - Alterando o nome da instância QComboBox

A janela criada deve ser salva sob o nome de **HelloWorldPluginWindow.ui** no mesmo diretório onde está o arquivo de projeto **HelloWorld.pro** como mostrado na Figura 9 - Local onde salvar HelloWorldPluginWindow.ui.

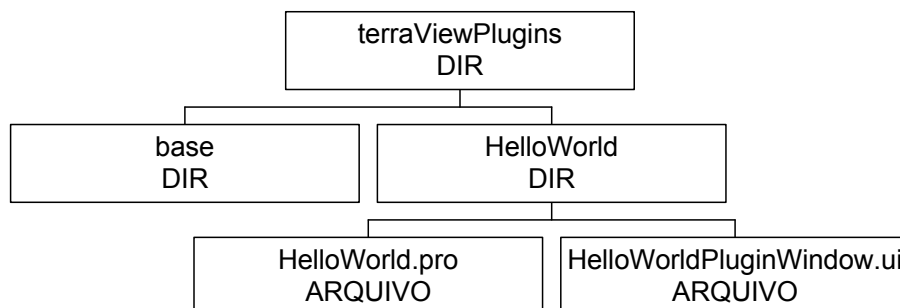


Figura 9 - Local onde salvar HelloWorldPluginWindow.ui

Criando o arquivo de cabeçalho do Plugin (HelloWorldPluginCode.h)

O arquivo **HelloWorldPluginCode.h** contém as informações necessárias para que o TerraView saiba que se trata de um plugin e que ele deverá ser carregado no momento em que ele for iniciado. A maior porção deste arquivo é a mesma para todos os plugins. A única porção de código que se diferencia de um plugin para outro é aquela definida pela macro **SPL_DEFINE_PLUGIN_INFO**.

Um editor de texto simples pode ser utilizado para criar este arquivo e o código deverá ser como o mostrado na Figura 10 – Código do arquivo HelloWorldPluginCode.h. Ele deverá ser salvo na localização indicada pela Figura 11 - Local para salvar HelloWorldPluginCode.h.

```
#include <spl.h>
#if SPL_PLATFORM == SPL_PLATFORM_WIN32
    // Include windows.h - Needed for Windows projects.
    #include <windows.h>
#endif
#include <PluginParameters.h>

SPL_DEFINE_PLUGIN_EXPORTS();

SPL_DEFINE_PLUGIN_INFO(
    1, // Build number (xx.1.0)..
    0, // Major version (1.xx.0).
    1, // Minor version (0.1.xx).
    true, // Always "true".
    "HelloWorld", // The plugin's name.
    "INPE-DPI", // The plugin's author.
    "A HelloWorld Plugin", // The plugin's general description.
    PLUGINPARAMETERS_VERSION, // Always "PLUGINPARAMETERS_VERSION".
    "http://www.dpi.inpe.br", // The plugin's homepage.
    "terraview@dpi.inpe.br", // The plugin author's email.
    "TerraViewPlugin.Generic Plugins" ); //Plugins menu disposition

SPL_DEFINE_PLUGIN_DLLMAIN();
SPL_IMPLEMENT_PLUGIN_GETINFO();
```

Figura 10 – Código do arquivo HelloWorldPluginCode.h

NOTA: O uso da macro **PLUGINPARAMETERS_VERSION** neste arquivo é necessário pois evita inconsistências entre os parâmetros de interface fornecidos pelo TerraView e os parâmetros de interface esperados pelo Plugin. Caso as versões não sejam as mesmas o plugin não será carregado.

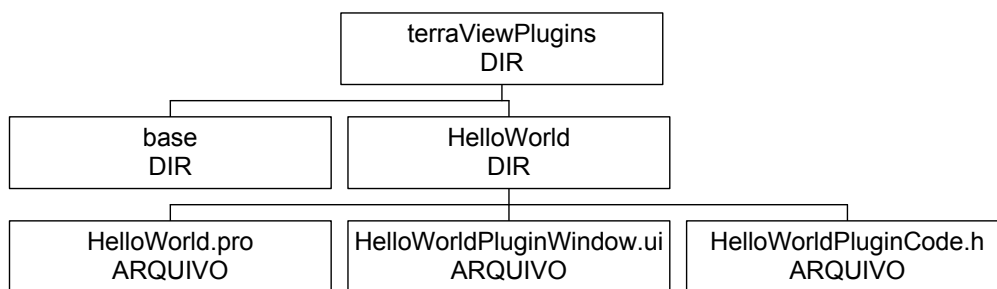


Figura 11 - Local para salvar HelloWorldPluginCode.h

Criando o arquivo com a implementação das funções do Plugin (HelloWorldPluginCode.cpp)

O arquivo **HelloWorldPluginCode.cpp** contém a implementação das funções que o TerraView espera que todo plugin possua. É através destas funções que o plugin recebe os parâmetros passados pelo TerraView e a partir deles tem acesso ao banco de dados e a interface gráfica do TerraView. Da mesma forma que no arquivo **HelloWorldPluginCode.h** estas funções são implementadas pela utilização de três macros:

- **SPL_INIT_NAME_CODE** – A função definida por esta macro será executada uma única vez quando o TerraView carregar a DLL/SO do plugin. Uma boa utilização desta função é a de inicializar/instanciar estruturas/objetos que estarão ativos durante todo o período em que o plugin estiver carregado.
- **SPL_RUN_NAME_CODE** – A função definida por esta macro será executada todas as vezes que o usuário acionar o nome do plugin no menu de plugins na interface principal do TerraView. Uma possível utilização é a de mostrar uma janela para o usuário todas as vezes que requisitado.
- **SPL_SHUTDOWN_NAME_CODE** - A função definida por esta macro será executada uma única vez momentos antes de a DLL/SO do plugin ser descarregada pelo TerraView. Uma boa utilização para esta função é

a de remover/apagar todas as estruturas/objetos que foram criados pelo plugin.

NOTA: Sempre remova todos os objetos e estruturas que foram criados durante a execução do plugin antes que ele seja descarregado da memória. Esta regra vale principalmente para objetos derivados de classes QT como a janela do plugin. Um bom local para tal remoção é na função definida pela macro SPL_SHUTDOWN_NAME_CODE.

No exemplo a seguir uma instância da janela do plugin é criada na função definida pela macro SPL_INIT_NAME_CODE. O ponteiro para esta instância é guardado em uma variável global “window_ptr” (no contexto do plugin). Todas as vezes que o usuário acionar o plugin no menu de plugins do TerraView a função definida por SPL_RUN_NAME_CODE é executada e a janela do plugin é atualizada com os nomes dos layers do banco de dados e mostrada. Ao final a janela do plugin é destruída na função definida por SPL_SHUTDOWN_NAME_CODE. O código é mostrado na Figura 12 – Código do arquivo HelloWorldPluginCode.cpp.

```
#include <HelloWorldPluginCode.h>
#include <HelloWorldPluginWindow.h>
#include <PluginParameters.h>

#include <qcombobox.h>

HelloWorldPluginWindow* window_ptr;

SPL_PLUGIN_API bool SPL_INIT_NAME_CODE( slcPluginArgs* a_pPluginArgs )
{
    /* Retriving parameters */

    void* arg_ptrs[ 1 ];
    a_pPluginArgs->GetArg( 0, arg_ptrs );
    PluginParameters* plug_pars_ptr =
        ( PluginParameters* ) arg_ptrs[ 0 ];

    /* Creating the plugin window instance */

    window_ptr = new HelloWorldPluginWindow( plug_pars_ptr->qtmain_widget_ptr_,
        "HelloWorldPluginWindow", Qt::WType_TopLevel | Qt::WShowModal );

    return true;
}

SPL_PLUGIN_API bool SPL_RUN_NAME_CODE( slcPluginArgs* a_pPluginArgs )
{
    /* Retriving parameters */
```

```

void* arg_ptrs[ 1 ];
a_pPluginArgs->GetArg( 0, arg_ptrs );
PluginParameters* plug_pars_ptr =
    ( PluginParameters* ) arg_ptrs[ 0 ];

/* Retriving the layers names */

window_ptr->q_db_layers_->clear();

if( plug_pars_ptr->getCurrentDatabasePtr() ) {
    TeLayerMap::iterator map_it =
        plug_pars_ptr->getCurrentDatabasePtr()->layerMap().begin();
    TeLayerMap::iterator map_it_end =
        plug_pars_ptr->getCurrentDatabasePtr()->layerMap().end();

    /* Updating the combo box */

    while( map_it != map_it_end ) {
        window_ptr->q_db_layers_->insertItem(
            map_it->second->name().c_str() );

        ++map_it;
    }
}

window_ptr->show();

return true;
}

SPL_PLUGIN_API bool SPL_SHUTDOWN_NAME_CODE( slcPluginArgs* a_pPluginArgs )
{
    delete window_ptr;

    return true;
}

```

Figura 12 – Código do arquivo HelloWorldPluginCode.cpp.

NOTA: A cada execução de uma das três funções definidas pelas macros anteriores um ponteiro para uma instância para um objeto que contém parâmetros do Plugin (**PluginParameters***) é passado. Esse objeto contém todos os métodos e propriedades que permitem a interação com a interface do TerraView e com o banco de dados. No tópico a seguir cada um deles é mostrado.

NOTA: Da mesma forma que na função **SPL_INIT_NAME_CODE** também é possível adquirir os parâmetros de plugins passados pelo TerraView nas funções **SPL_RUN_NAME_CODE** e **SPL_SHUTDOWN_NAME_CODE**.

Parâmetros do Plugin

Os parâmetros disponibilizados pelo TerraView para os plugins permitem a comunicação bidirecional entre a aplicação principal (o TerraView) e os plugins. Os parâmetros são (veja o código do arquivo **PluginParameters.h** para uma documentação detalhada):

Métodos disponíveis:

- **getCurrentDatabasePtr()** – Retorna um ponteiro para a instância do banco de dados ativo. Se nenhum banco está ativo um valor nulo será retornado.
- **getCurrentLayerPtr()** – Retorna um ponteiro para a instância do *layer* ativo dentro do banco de dados ativo. Se nenhum *layer* está ativo um valor nulo será retornado.
- **getCurrentViewPtr()** - Retorna um ponteiro para a instância da vista ativa dentro do banco de dados ativo. Se nenhuma vista está ativa um valor nulo será retornado.
- **getCurrentThemeAppPtr()** - Retorna um ponteiro para a instância do tema ativo dentro do banco de dados ativo. Se nenhum tema está ativo um valor nulo será retornado.
- **updateTVInterface()** – Atualiza a interface do TerraView recarregando o banco de dados ativo.
- **plotTVData ()** – Força o TerraView a redesenhar a vista corrente.
- **loadTranslationFile(const std::string& filename)** – Carrega um arquivo de tradução do formato suportado pelo QT (arquivos .QM) em memória. A tradução carregada será adicionada a lista de traduções já existentes no TerraView e quando a janela do plugin for mostrada ela será traduzida automaticamente.

São fornecidos ponteiros para os elementos de interface do TerraView (mostrados na Figura 13). Estes ponteiros são sempre válidos:

- **QMainWindow* qtmain_widget_ptr_** - Ponteiro para a instância da janela principal do TerraView que pode ser utilizada como janela pai para a janela principal do plugin. Através deste ponteiro também é possível a criação de novas barras de ferramentas e botões que podem ser inseridos na interface do TerraView e associados ao plugin.
- **TeQtDatabasesListView* teqtdatabaseslistview_ptr_** - Ponteiro para a lista de bancos de dados abertos e as respectivas camadas de informação. Através desta instância também é possível ter acesso ao ponteiro das instâncias dos bancos de dados e as respectivas camadas de informação. Instância representada pelo número 1 na Figura 13 – Elementos de interface do TerraView.
- **TeQtViewsListView* teqtviewslistview_ptr_** - Ponteiro para a instância da lista de vistas e temas para o banco de dados ativo. Instância representada pelo número 2 na Figura 13 – Elementos de interface do TerraView.
- **TeQtGrid* teqtgrid_ptr_** - Ponteiro para a instância da grade de informações para o tema ativo. Instância representada pelo número 3 na Figura 13 – Elementos de interface do TerraView.
- **TeQtCanvas* teqtcanvas_ptr_** - Ponteiro para a instância da área de desenho do TerraView. Instância representada pelo número 4 na Figura 13 – Elementos de interface do TerraView.

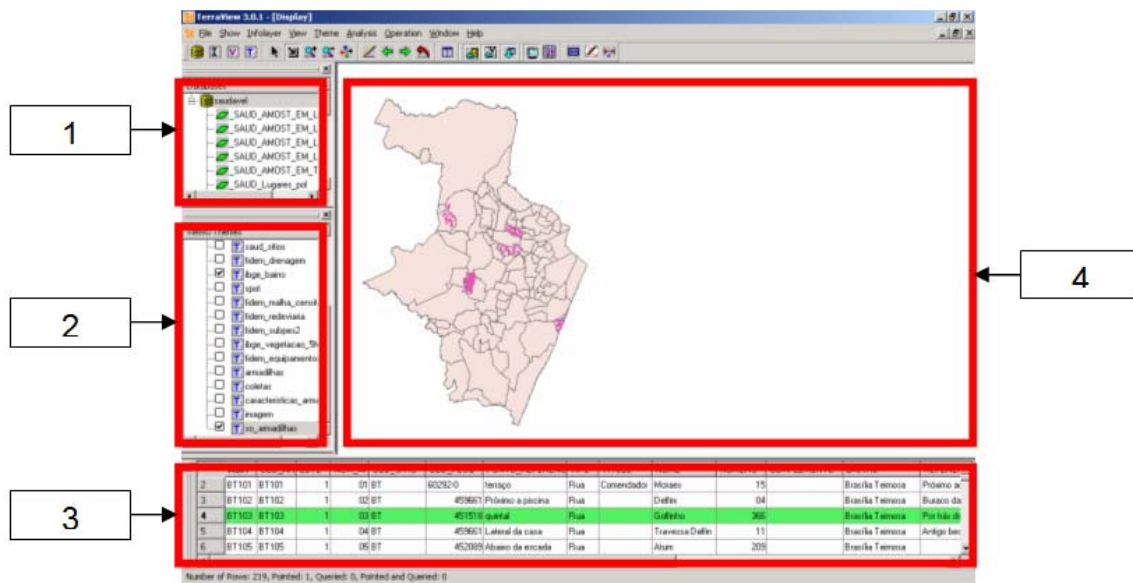


Figura 13 – Elementos de interface do TerraView

Gerando o arquivo de projeto e compilando

O arquivo de projeto do plugin (**HelloWorld.pro**) pode ser usado para gerar o arquivo de projeto para o MS Visual Studio ou Makefiles para Linux através da utilização da ferramenta “**qmake**” fornecida no kit de desenvolvimento QT. Para maiores informações consulte o site do QT.

Na Figura 14 - Geração do arquivo de projeto para Visual Studio - é mostrada a utilização no ambiente Windows: utilizando uma janela do *prompt* de comando do sistema o aplicativo qmake é chamado a partir do diretório onde está o arquivo de projeto do plugin **HelloWorld.pro**.

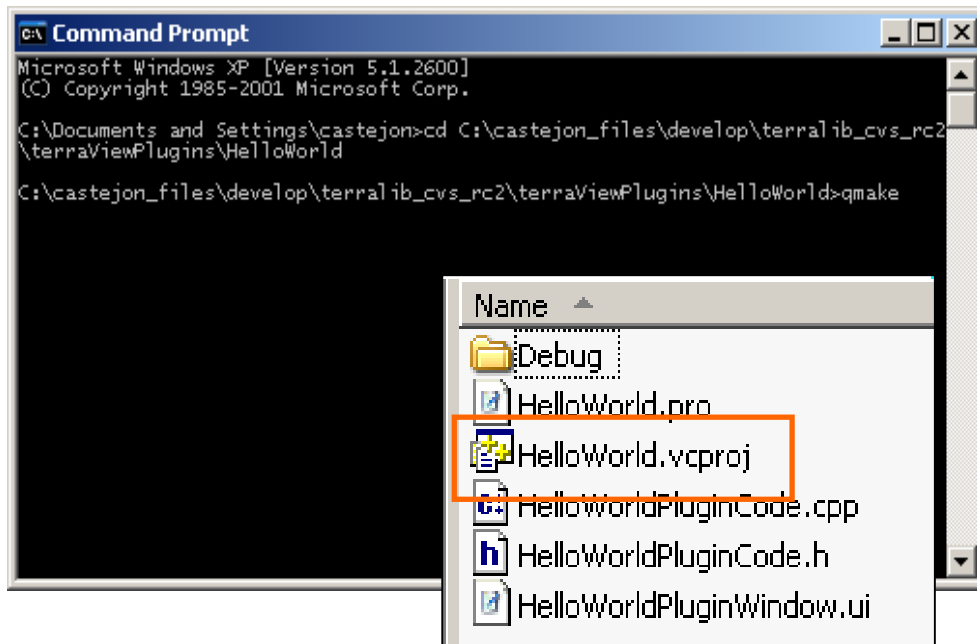


Figura 14 - Geração do arquivo de projeto para Visual Studio

Utilizando o arquivo de projeto do Visual Studio ou o Makefile gerado é possível compilar a biblioteca do plugin. Se o projeto base de plugins (**base.pro**) foi utilizado as bibliotecas já serão geradas nos lugares corretos, a partir dos quais o TerraView será capaz de encontra-los, como mostrado na Figura 15 - Localização das bibliotecas geradas.

NOTA: Para ser capaz de debugar o código do plugin gerado no ambiente do Visual Studio é necessário adicionar o arquivo de projeto do plugin (HelloWorld.vcproj) no arquivo de solução do TerraView (terraView.sln) - Figura 16 - Inserindo o projeto do plugin no solution do TerraView. O arquivo terraView.sln pode ser encontrado no diretório mostrado na Figura 17 - Localização do arquivo de solução do TerraView.

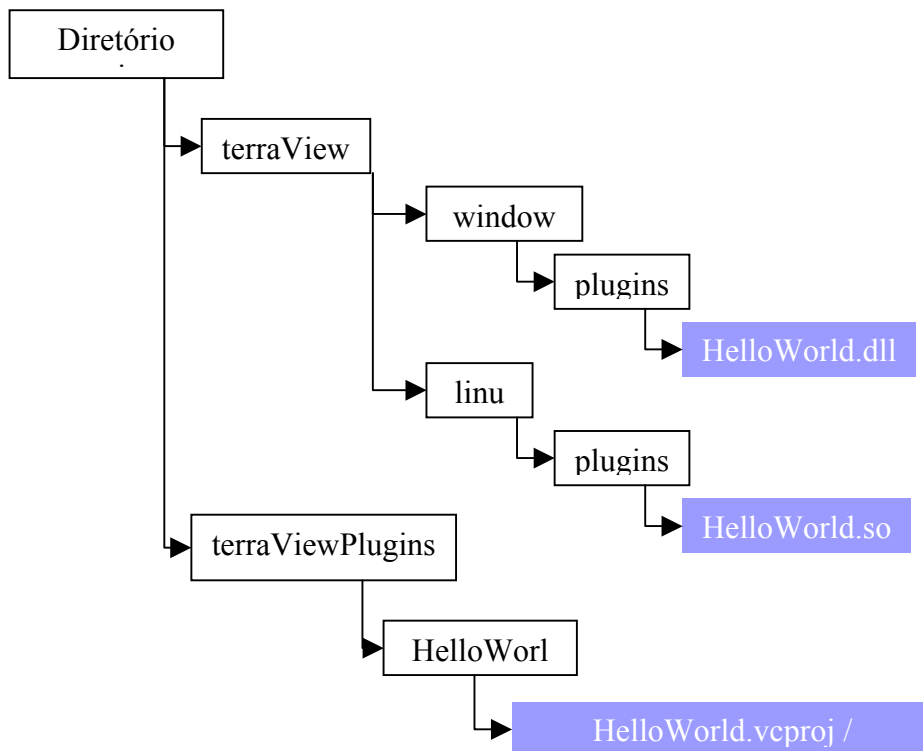


Figura 15 - Localização das bibliotecas geradas

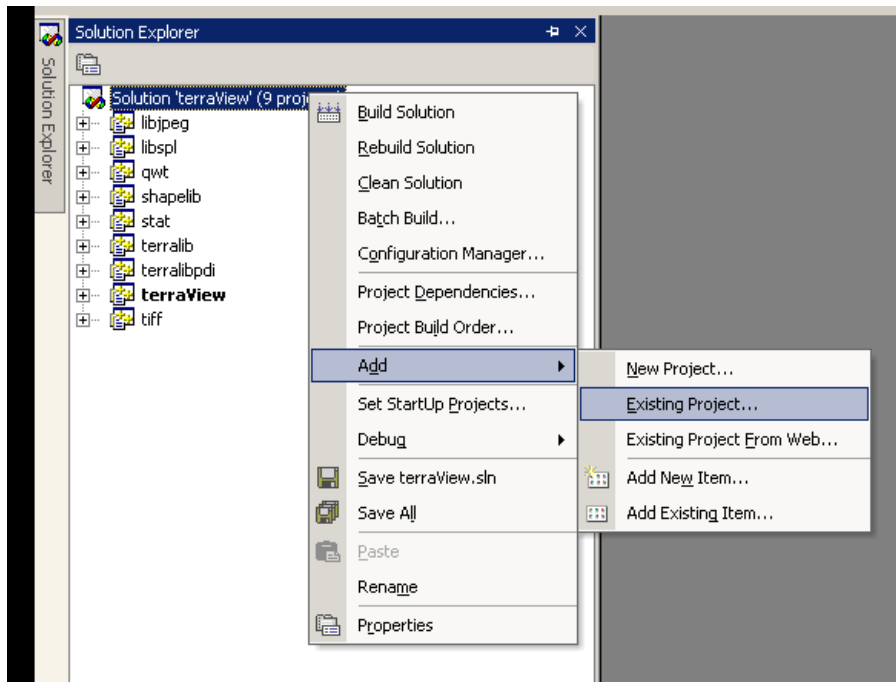


Figura 16 - Inserindo o projeto do plugin no solution do TerraView

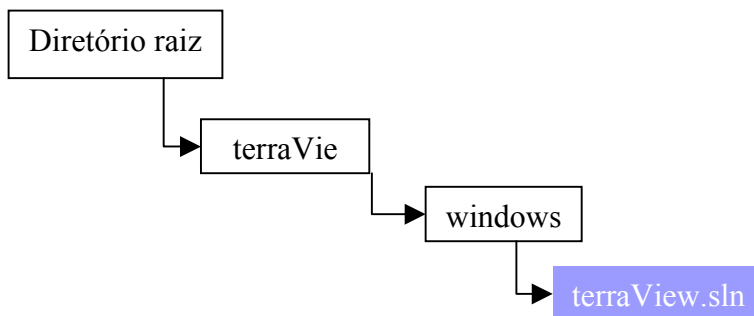


Figura 17 - Localização do arquivo de solução do TerraView

Compilando para a distribuição

Para que qualquer *plugin* possa ser carregado por uma versão de distribuição do TerraView (*release*) oferecida no *site* é necessário que o projeto do *plugin* seja alterado e que ele seja compilado no **modo release**. Para tal o arquivo **base.pro** mostrado na Figura 3 - Diretório de código de plugins – deve ser editado de forma que a linha:

```
CONFIG += dll thread warn_on debug rtti exceptions
```

Seja alterada para:

```
CONFIG += dll thread warn_on release rtti exceptions
```

Depois de modificado e salvo o arquivo **base.pro** o próximo passo é re-gerar o arquivo de projeto **HelloWorld.vcproj** (para ambiente Windows) ou o **Makefile** (para ambiente Linux) do plugin utilizando a ferramenta **qmake** - Figura 14 - Geração do arquivo de projeto para Visual Studio. O novo arquivo de projeto/Makefile deve ser re-compilado e ao final do processo a biblioteca do plugin será gerada no mesmo local indicado pela Figura 15 - Localização das bibliotecas geradas – mas estará compilada em modo *release*.

NOTA: Para que *plugins* de terceiros sejam carregados pela versão oficial do TerraView distribuída no *site* outro requisito, além de ter sido compilado em modo *release*, é o de que ele também tenha sido compilado utilizando a mesma versão do *toolkit* QT utilizado pelo TerraView.

NOTA: É aconselhável que todos os arquivos gerados em modo *debug* sejam apagados antes de compilar o plugin em modo *release*. No ambiente Windows basta acionar a opção de apagar os arquivos do projeto, disponível no menu acionado ao se clicar com o botão direito do *mouse* sobre o nome do projeto. No ambiente Linux basta executar o comando “*make distclean*” a partir do diretório do projeto do plugin.

Questões freqüentes

3. Meu plugin é compilado corretamente, mas o TerraView não o reconhece. O que está errado?

Verifique se a versão do plugin é compatível com a versão do TerraView executado.

Verifique se existem referências indefinidas durante a compilação. O carregador de códigos dinâmicos do sistema operacional não carrega bibliotecas onde faltam implementações de funções.

4. Meu plugin funcionava com a versão anterior do TerraView, mas agora a nova versão não o reconhece. O que posso fazer?

A estrutura de Plugins do TerraView verifica se a versão de interface é a mesma para o Plugin e para o TerraView. Se essa verificação falha, o plugin não será carregado. Recompile o código do seu plugin, atualizando-o com a nova interface fornecida pelo TerraView.

5. Tudo está correto, mas quando eu fecho o TerraView (depois de ter usado meu plugin) ocorre um erro de falha de segmentação. O que está errado?

Possivelmente o TerraView está tentando destruir um objeto criado pelo Plugin. Verifique se todos os objetos criados durante a execução do plugin foram destruídos antes do momento de sua descarga da memória.