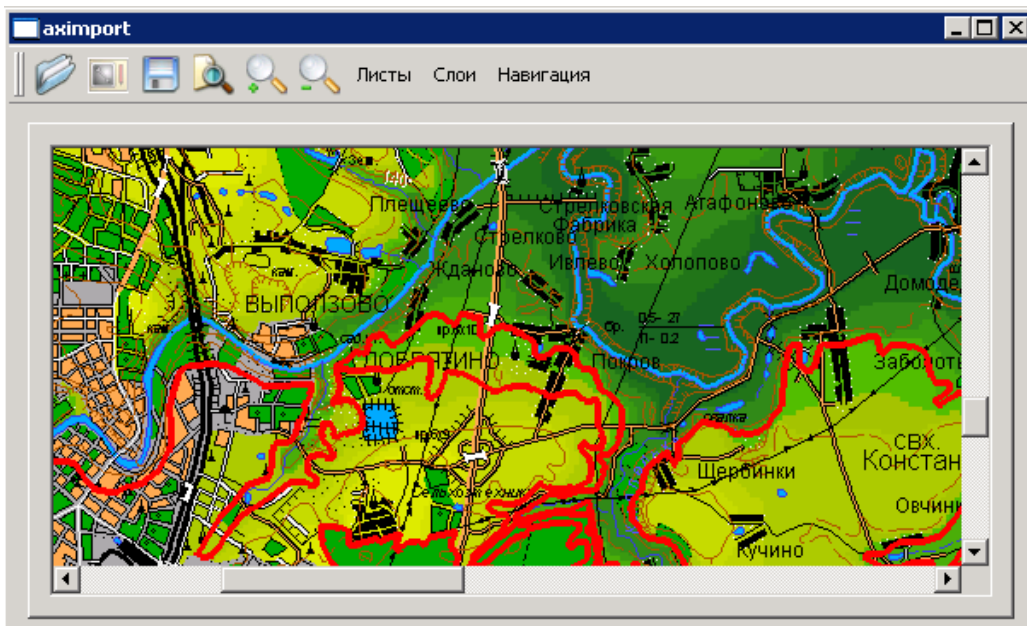


Qt и ActiveX



Недавно мне понадобилось задействовать в моем приложении интерфейс, представленным набором элементов ActiveX и COM объектов. В Qt есть весьма мощные классы для работы с ActiveX и COM-объектами, входящие в специальную подсистему `activeqt-framework`. С помощью этой системы можно создавать как серверы COM-ActiveX, так и использовать интерфейсы, экспортируемые в виде COM другими программами. Если бы мне понадобилось предоставлять API своего приложения для других программ, COM и ActiveX я бы выбрал только в том случае, если бы другие варианты отсутствовали. Среди многих причин, побуждающих меня не использовать ActiveX в своих программах, достаточно назвать трудности с переносом интерфейса на другие платформы. Однако при работе с программами Windows у нас иногда нет другого выбора, кроме использования ActiveX.

Учитывая вышеизложенное, я сосредоточусь на проблеме использования в программе API других программ, основанных на COM-объектах (и не буду рассматривать вопрос создания сервера таких объектов средствами Qt). По сравнению со стандартными Windows API, предназначенными для работы с COM, использование специальных классов Qt дает нам то преимущество, что мы легко можем интегрировать интерфейсы COM объектов в приложения Qt (применять типы данных Qt, встраивать визуальные элементы ActiveX в иерархию виджетов, использовать сигналы и слоты). Для решения этой задачи у нас есть два мощных класса: `QAxObject` и `QAxWidget`. Если вы знакомы с документацией по этим классам, то знаете, что их возможностей в принципе достаточно для работы с объектами ActiveX/COM. Однако, эти средства не очень удобны. Вызывать методы и свойства объектов придется динамически, с использованием мета-информации об объекте, что, во-первых, требует много лишнего кода, а во вторых замедляет быстродействие программы.

В Qt есть решение этой проблемы – программа `dumpscpp`, которая анализирует файлы `*.tlb`, `*.odl`, `*.ocx`, `*.dll` (если DLL является библиотекой ActiveX) и генерирует файлы исходных текстов C++. В этих файлах создаются иерархии классов Qt, соответствующих интерфейсам COM. Эти классы основаны на классах `QAxObject` и `QAxWidget`. В результате решается только проблема удобства

вызова методов интерфейсов. Проблема быстрого действия остается, так как для вызова методов используется все тот же механизм динамического вызова. С этим я еще мог бы смириться, если бы не одно непреодолимое обстоятельство. При попытке создать иерархию классов для той библиотеки ActiveX, с которой мне пришлось иметь дело, утилита `dumpsrcr` просто не справилась с задачей. Не все необходимые классы и структуры данных были сгенерированы. Возможно, причина в том, что библиотека ActiveX отличалась уж очень большим количеством объектов. Проблему можно было бы решить, дописав отсутствующие структуры вручную, но необходимость дописывать большое количество структур сводила на нет все преимущества автоматической генерации классов. Стоит отметить, что утилита `dumpsrcr` появилась в Qt сравнительно недавно, и не все программисты, работающие с Qt, о ней знают.

Так или иначе, я решил поискать другой способ быстрой и удобной интеграции Qt и COM. Дальнейшее описание предполагает, что для разработки программы используется Microsoft VS. Это естественное предположение, поскольку ActiveX – фирменная технология Microsoft и лучше всего с ней должна справляться фирменная IDE.

Что происходит в программе, когда мы включаем в проект библиотеку ActiveX с помощью директивы `#import`, например:

```
#import "axGisToolKit.tlb"
```

Специальный инструмент Visual Studio создает два файла, с расширениями `tlh` и `tli`. Эти файлы содержат все необходимое для работы с интерфейсами COM в нашей программе.

Инструментарий Microsoft, предназначенный для генерации файлов C++ на основе описаний типов ActiveX, делает то же самое, что и утилита `dumpsrcr`, но справляется с теми библиотеками, с которыми `dumpsrcr` не по зубам. Единственное неудобство, связанное с инструментами Microsoft, заключается в том, что они ничего не знают о типах данных, используемых в Qt. Файлы `tlh` и `tli` создаются автоматически, во временной директории, а объявленные в них идентификаторы становятся доступны в том файле исходных текстов, в который мы включили директиву `#import`. Мы можем вообще не заботиться о временных файлах, но, поскольку мы собираемся интегрировать ActiveX и Qt, заглянуть в файл с расширением `tlh` все-таки стоит. Оказывается, что это – обычный заголовочный файл C++, который содержит все определения, необходимые для работы с импортированной библиотекой.

Для упрощения работы с объектами COM определяются два шаблона классов:

```
template<typename T>
class QExtAxObject : public QAxObject
{
public:
    QExtAxObject( QObject * parent = 0 ) : QAxObject( parent )
    {
        mPtr = 0;
    }
    ~QExtAxObject()
    {
        clear();
    }
    T * operator ->() const
    {
        return mPtr;
    }
}
```

```

bool setControl( const QString & control )
{
    if (QAxObject::setControl(control)) {
        queryInterface(QUuid(__uuidof(T)), (void**) &mPtr);
        return true;
    }
    return false;
}
void clear()
{
    if(mPtr)
        mPtr->Release();
    mPtr = 0;
    QAxObject::clear();
}
private:
    T * mPtr;
};

template<typename T>
class QExtAxWidget : public QAxWidget
{
public:
    QExtAxWidget(QWidget * parent = 0, Qt::WindowFlags f = 0) :
    QAxWidget(parent, f)
    {
        mPtr = 0;
    }
    ~QExtAxWidget()
    {
        clear();
    }
    virtual void clear()
    {
        if(mPtr)
            mPtr->Release();
        mPtr = 0;
        QAxWidget::clear();
    }

    bool setControl( const QString & control )
    {
        if (QAxWidget::setControl(control)) {
            queryInterface(QUuid(__uuidof(T)), (void**) &mPtr);
            return true;
        }
        return false;
    }
    T * operator ->() const
    {
        return mPtr;
    }
private:
    T * mPtr;
};

```

Шаблон класса `QExtAxObject` происходит от класса `QAxObject`, а шаблон класса `QExtAxWidget` – от класса `QAxWidget` соответственно. Эти шаблоны упрощают две операции: инициализацию объекта и получение интерфейса COM. При этом с точки зрения семантики вызова, методы и свойства интерфейса становятся методами и свойствами объекта Qt. В качестве параметра

шаблона передается имя интерфейса, затем, с помощью метода `setControl()` инициализируется объект. Схематически это выглядит так:

```
QExtAxObject<IaxMapView> mapView; // Объявление переменной
...
mapView.setControl("axGisToolKit.axcMapView"); // Инициализация объекта
mapView->PlaceOut = PP_PICTURE; // Обращение к свойству интерфейса COM
mapView->MapClose(); // Вызов метода интерфейса COM
mapView.clear(); // Деинициализация объекта
```

Чтобы понять, что тут происходит, рассмотрим подробно функцию `setControl()` класса `QExtAxObject`. Сначала эта функция вызывает одноименную функцию базового класса, в результате чего, если имя объекта задано правильно, инициализируется объект. Затем, с помощью функции `queryInterface()` мы получаем указатель на тот интерфейс, имя которого было задано в качестве параметра шаблона класса. Этот указатель сохраняется в переменной `mPtr`. Благодаря оператору `__uuidof()` мы можем не заботиться об идентификаторе интерфейса.

Доступ к указателю `mPtr` осуществляется с помощью оператора `->()`. Таким образом, при обращении к объекту с помощью оператора прямого выбора «.» мы получаем доступ к методам `QExtAxObject` (например, к методу `setControl()`). Если для обращения к объекту используется оператор косвенного выбора «->» мы получаем доступ к элементам интерфейса COM. Этот принцип похож на принцип работы с интеллектуальными указателями Qt и сам класс `QExtAxObject` можно рассматривать как интеллектуальный указатель на экземпляр соответствующего интерфейса. До вызова функции `setControl()` переменная `mPtr` хранит значение 0, так что вызов оператора `->()` приведет к ошибке. Это же значение записывается в переменную `mPtr` в результате вызова функции `clear()`. Кроме того, функция `clear()` уменьшает счетчик ссылок интерфейса с помощью метода `Release()`. Поскольку деструктор класса вызывает функцию `clear()`, в непосредственном вызове ее зачастую нет необходимости.

Наша функция `setControl()` возвращает `false` и `true` в тех же случаях, что и одноименная функция класса `QAxObject`.

Аналогичным образом можно создать класс `QExtAxWidget` на основе класса `QAxWidget` (оба шаблона классов вы найдете в файле исходных текстов).

Таким образом, предложенное решение представляет собой гибрид из Windows COM API и интерфейса системы Qt, предназначенной для работы с ActiveX. Минус этого подхода, по сравнению с подходом, основанным на `dumprscr`, заключается в том, что нам приходится использовать типы данных Windows API там, где мы могли бы использовать типы данных Qt. При этом не следует забывать, что сама задача возникла из-за того, что `dumprscr` не справился со своей работой (да и преобразовать типы, используемые при работе с COM в типы Qt и обратно совсем несложно). Преимущество этого решения, по сравнению с использованием Windows API, заключается в том, что объекты COM являются так же и объектами Qt. Это означает, например, что с объектом, производным от `QExtAxWidget`, можно работать как с обычным виджетом Qt. А события объектов COM могут обрабатываться как сигналы Qt-объектов:

```

QExtAxWidget<IaxMapWindow> mapWindow;

...

mapWindow.setControl("axGisToolKit.axcMapWindow");
mapWindow.setParent(ui.frame);
ui.horizontalLayout->addWidget(&mapWindow, 1);

connect(&mapWindow, SIGNAL(OnMapMouseDown(int,int,double,double,double)),
this, SLOT(mapMouseDown(int,int,double,double,double)));
    connect(&mapWindow, SIGNAL(OnMapMouseUp(int,int,double,double,double)),
this, SLOT(mapMouseUp(int,int,double,double,double)));
    
```

У объекта `QExtAxWidget` нет сигнала `OnMapMouseDown()`, не определен этот сигнал и в интерфейсе `IaxMapWindow`, рассмотренном здесь в качестве примера, но он появится после инициализации объекта `mapWindow` (об этом сигнале можно узнать из файла `tlh` или с помощью анализа метаинформации объекта `mapWindow` после его инициализации).

И еще немного о типах данных. В Qt существует несколько функций, предназначенных для преобразования типов данных COM в типы данных Qt и обратно. Документация по Qt об этих функциях не упоминает (вероятно, потому, что считает их предназначенными для внутреннего употребления), но при низкоуровневой работе с COM они могут пригодиться.

Функции преобразования типов объявлены в файле `qaxtypes.h`. Ниже приводится их полный список.

```
BSTR QStringToBSTR(const QString &str)
```

Эта функция преобразует строку `QString` в строку типа `BSTR`, который используется объектами COM. Для выделения памяти функция использует функцию Windows API `SysAllocStringLen()`, поэтому память, занятую переменной, следует высвободить с помощью функции `SysFreeString()`. Функции для обратного преобразования не существует, так как преобразование строки `BSTR` в `QString` может быть выполнено простым присваиванием.

```
uint QColorToOLEColor(const QColor &col)
```

Эта функция преобразует тип `QColor` в тип `OLEColor`. Уже из объявления функции можно понять, что она очень проста. Обратное преобразование выполняет функция

```
QColor OLEColorToQColor(uint col).
```

Функции

```
bool QVariantToVARIANT(const QVariant &var, VARIANT &arg, const QByteArray
&typeName = 0, bool out = false);
```

и

```
QVariant VARIANTToQVariant(const VARIANT &arg, const QByteArray &typeName,
uint type = 0);
```

выполняют преобразование типа `QVariant` в `OLE VARIANT` и обратно.

Функция

```
void clearVARIANT (VARIANT *var)
```

высвобождает ресурсы переменной типа `VARIANT`, созданной динамически.

Функции

```
IFontDisp *QFontToIFont (const QFont &font)
```

и

```
QFont IFontToQFont (IFont *f)
```

преобразуют объект `QFont` в соответствующий объект COM. Эти функции не попали даже в `qaxtypes.h`, их можно найти только в файле `qaxtypes.cpp`.

Функции

```
IPictureDisp *QPixmapToIPicture (const QPixmap &pixmap)
```

и

```
QPixmap IPictureToQPixmap (IPicture *ipic)
```

делают тоже самое для объектов класса `QPixmap`, а функции

```
DATE QDateTimeToDATE (const QDateTime &dt)
```

и

```
QDateTime DATEToQDateTime (DATE ole)
```

Для объектов `QDateTime`.