Dave's Development Blog
Software Development using Borland / Codegear / Embarcadero RAD Studio

The Open Tools API using C++ Builder

By David | December 5, 2016 0 Commen

Overview

A week or so ago Alex Bassi was asking on the CPP Slack channel how to do Open Tools API stuff in C++ Builder. I had started looking into this a while ago but found a few hicups along the way and put it all on the back burner until I had more time. His request made me pick it up again and solve most of the problems.

I'll say here that I'm not a seasoned C++ Builder developer but with my knowledge of Delphi and it's RTL and VCL frameworks I'm a bit more than a novice however I've probably approached some of the code in a very Object Pascal way so apologies to all C++ developers reading this.

So I've spent the last week or so coding up an example which will end up being a C++ equivalent to the OTA Template. At the moment it's no where near there and only contains the following example code in the source:

- Basic Menu Wizard code;
- Package creation code;
- DLL creation code;
- Splash screen;
- About Box;
- IDE Notifier;
- Auto Save code;
- IDE Options frame for the Auto Save Options.

What I'm not going to do is explain the functionality of the OTA as this has already been done but I will provide references to those articles. I'm also not going to explain C++ unless I feel it's necessary in the context of implementing the Open Tools APIs. Also I've created this with RAD Studio Berlin but I'm hoping the code is backwardly compatible to circa XE. I haven't got around to creating multiple version for different

compilers using IFDEFs yet.

References

I can't claim to have worked all this out myself as I've used a number of Borland / Codegear / Embarcadero references to help me along.

The first reference is a section in the help entitled Extending the IDE Using the Tools API. It's part of the Components Writer's Guide. It contains a lot of the information you will need. It first appeared in the C++ Builder 6 Developer's Guide which you can still download from http://docs.embarcadero.com/products/rad_studio/. It's right at the bottom of the page. I still have my Delphi 5 and C++ Builder 5 manuals but this chapter does not appear in either.

The above provides a lot of good information but there are some little bits that are missing and you can find them in Code Central here: ID: 17305, Developer's Guide Tools API Examples. It seems that this was all written by Ray Lischner, author of the original OTA book.

Notifer / Interfaced Object

First create a new package in the IDE and then create a C++ unit in that package (I'll leave the structure of your source up to you but there are suggestions here (Chapter 1: Starting an Open Tools API Project)).

The first place we'll start is with a class to handle interface references. Now I've followed the examples in the code here and used a class to act as a based class for notifiers and interfaced objects. I think in later blogs I will break this down into 2 objects, one for interfaced objects and another derived from that for notifier objects.

The first thing to understand is that C++ does not have native interfaces as Delphi does so interfaces are implemented in C++ as pure virtual classes which must have ALL their methods overridden. Now some of you will say why not use the `TInterfaceObject` which is part of the RTL. I can't be certain but I suspect that it would not work for the above reason.

Declaration

So we will start with the declaration of the notifier object class we will use for all our objects:

```
#ifndef CPPOTATemplateNotifierObjectH
#define CPPOTATemplateNotifierObjectH

#include <ToolsAPI.hpp>

class PACKAGE TDGHNotifierObject : public IOTANotifier {
  private:
    long ref_count;
    String FObjectName;
  protected:
    void __fastcall DoNotification(String strMessage);
    // IOTANotifier
    void __fastcall AfterSave();
    void __fastcall BeforeSave();
    void __fastcall Destroyed();
    void __fastcall Modified();
    // IInterface
    virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
    virtual ULONG __stdcall AddRef();
    virtual ULONG __stdcall Release();
  public:
    __fastcall TDGHNotifierObject(String strObjectName);
    virtual __fastcall ~TDGHNotifierObject();
};
#endif
```

There are a number of things to consider here. The first is that all the header is defined between the `#ifndef` and `#endif` statements. With me not being a C++ guru it took me sometime to work out why this should be the case. For non-C++ people the header files are not quite the same as Delphi's interface sections and the `#include` statements are not the same as Delphi s `uses` statements. If you don't place your header declarations in the `#ifndef` section you will eventually end up with the compiler saying you are declaring some of your objects mulitple times, especially with the above base class which will be used multiple times.

The next thing was to include the `ToolsAPI.hpp` file however this is not enough to get your code to compile. You need to added $(BDSINCLUDE)\windows\vcl\design\ to your project's include list.

Next I've declared a field for the reference counting and methods I need to override for `IInterface` and `IOTANotifier`. One thing to be careful of is NOT to simply copy the methods from the `ToolsAPI.hpp` file as there are some modifiers that should not be present in your declarations like `Virtual` and `HIDEBASE`.

Finally I've added a number of of extra methods and fields so that the object can output messages to the message window and know what the

class implementing this base class is called.

Implementation

Below is the implementation of the above object:

```
#pragma hdrstop
#include "CPPOTATemplateNotifierObject.h"
#include "CPPOTATemplateMacros.h"
#include "Forms.hpp"
#pragma package(smart_init)

/* Notifer Implementation */

__fastcall TDGHNotifierObject::TDGHNotifierObject(String strObjectName) {
  ref_count = 0;
  FObjectName = strObjectName;
}

__fastcall TDGHNotifierObject::~TDGHNotifierObject() {
  // Do nothing destructor
}

ULONG __stdcall TDGHNotifierObject::AddRef() {
  return InterlockedIncrement(&ref_count);
}

ULONG __stdcall TDGHNotifierObject::Release() {
  ULONG result = InterlockedDecrement(&ref_count);
  if (ref_count == 0)
    delete this;
  return result;
}

HRESULT __stdcall TDGHNotifierObject::QueryInterface(const GUID& iid, void** obj) {
  QUERY_INTERFACE(IInterface, iid, obj);
  QUERY_INTERFACE(IOTANotifier, iid, obj);
  return E_NOINTERFACE;
}

void __fastcall TDGHNotifierObject::AfterSave() {
  DoNotification(FObjectName + "::AfterSave");
};

void __fastcall TDGHNotifierObject::BeforeSave() {
  DoNotification(FObjectName + "::BeforeSave");
};

void __fastcall TDGHNotifierObject::Destroyed() {
  DoNotification(FObjectName + "::Destroyed");
};

void __fastcall TDGHNotifierObject::Modified() {
  DoNotification(FObjectName + "::Modified");
};

void __fastcall TDGHNotifierObject::DoNotification(String strMessage) {
  _di_IOTAMessageServices MsgServices;
  if (BorlandIDEServices->Supports(MsgServices)) {
    // Make sure messages are not added when the IDE is being destroyed.
    if (Application->MainForm->Visible) {
      _di_IOTAMessageGroup MsgGrp = MsgServices->AddMessageGroup("C++ OTA Template");
      MsgServices->ShowMessageView(MsgGrp);
      MsgServices->AddTitleMessage(FObjectName + strMessage, MsgGrp);
    }
  }
}
```

So in the above the constructor initialises the reference counting to zero and stores the name passed to the constructor for later use in outputting messages so that you know which object is outputting messages.

The method `AddRef()` calls the windows function `InterlockedIncrement()` to increment the reference counter in a thread safe manner and returns the resulting incremented value. Likewise `Release()` decrements the reference count in a similar manner. If the count gets to zero the `Release()` frees the object.

`QueryInterface` is where the most work is done. Embedded in this method are 2 calls to a macro which determines whether the passed object implements a specific interface. The difference here is that you MUST implement this function with the checks for the interfaces you are implementing for you code to work. Secondly, the macro is used as this is a pattern that will be used numerous times in the application. The macro is defined as below (Note: I've put my macros in their own unit):

```
#define QUERY_INTERFACE(T, iid, obj)   \
  if ((iid) == __uuidof(T)) {          \
    *(obj) = static_cast<T*>(this);    \
    static_cast<T*>(*(obj))->AddRef(); \
    return S_OK;                       \
  }
```

This checks that the passed interface is of the type given and if so returns the cast object reference. Note: Macros in C++ are replaced in-line and are not the same as functions.

The `BeforeSave()`, `AfterSave()`, `Destroyed()` and `Modified()` methods simply return a notification message if they are called.

Finally the `DoNotification()` method outputs a message to the message view in a new group. Here is where C++ OTA code differs from Object Pascal. Since C++ does not have interfaces you have to implement them using macros that are defined in the `ToolsAPI.hpp` file. So for instance I need a `IOTAMessageServices` interface from the `BorlandIDEServcies` interface. First I declare my variable using the macro `_di_IOTAMessageServcies` where the _di_ stands for Delphi Interface. I can then ask `BorlandIDEServices` if it supports the interface and if so I can then call the appropriate methods.

Basic Wizard

So now we need to declare our wizard so we can do something. Create a new unit in your package.

Declaration

So here I've implemented a menu wizard. I've done this before here (Chapter 1: Starting an Open Tools API Project) so if you want to understand the OTA interfaces please have a read.

Here's my C++ declaration below:

```
 #ifndef CPPOTATemplateMainWizardH
 #define CPPOTATemplateMainWizardH

 #include <ToolsAPI.hpp>
 #include <CPPOTATemplateNotifierObject.h>

class PACKAGE TCPPOTATemplateWizard : public TDGHNotifierObject, public IOTAMenuWizard {
  typedef TDGHNotifierObject inherited;
  private:
  protected:
    // IOTAWizard
    virtual UnicodeString __fastcall GetIDString();
    virtual UnicodeString __fastcall GetName();
    virtual TWizardState  __fastcall GetState();
    virtual void __fastcall Execute();
    // IOTAMenuWizard
    virtual UnicodeString __fastcall GetMenuText();
    // IOTANotifer
    void __fastcall BeforeSave();
    void __fastcall AfterSave();
    void __fastcall Destroyed();
    void __fastcall Modified();
    // IInterface
    virtual HRESULT __stdcall QueryInterface(const GUID& iid, void** obj);
    virtual ULONG __stdcall AddRef();
    virtual ULONG __stdcall Release();
    // Non-interface methods
  public:
    __fastcall TCPPOTATemplateWizard(String strObjectName);
    __fastcall ~TCPPOTATemplateWizard();
};
 #endif
```

So here we create a class that is inherited from our notifier object and `IOTAMenuWizard`. In C++ you do not need to inherited from all the chain of interfaces like you do in Delphi as the interfaces are implemented as classes and therefore this is happening anyway.

Implementation

Below is the implementation of the wizard:

```
#pragma hdrstop
#include <CPPOTATemplateMainWizard.h>
#include <Dialogs.hpp>
#include <CPPOTATemplatePkgDLLInit.h>
#include "CPPOTATemplateMacros.h"
#pragma package(smart_init)

/* TCPPOTATemplateWizard Implementation */

__fastcall TCPPOTATemplateWizard::TCPPOTATemplateWizard(String strObjectName) :
  TDGHNotifierObject(strObjectName) {
  // Do nothing constructor
}

__fastcall TCPPOTATemplateWizard::~TCPPOTATemplateWizard() {
  // Do nothing destructor
}

ULONG __stdcall TCPPOTATemplateWizard::AddRef() {
  return inherited::AddRef();
}

ULONG __stdcall TCPPOTATemplateWizard::Release() {
  return inherited::Release();
}

HRESULT __stdcall TCPPOTATemplateWizard::QueryInterface(const GUID& iid, void** obj) {
  QUERY_INTERFACE(IOTAMenuWizard, iid, obj);
  QUERY_INTERFACE(IOTAWizard, iid, obj);
  return inherited::QueryInterface(iid, obj);
}

UnicodeString __fastcall TCPPOTATemplateWizard::GetIDString() {
  return "CPP.OTA.Template.Wizard";
}

UnicodeString __fastcall TCPPOTATemplateWizard::GetName() {
  return "CPP OTA Temaple";
}

TWizardState  __fastcall TCPPOTATemplateWizard::GetState() {
  TWizardState result;
  result << wsEnabled;
  return result;
}

void __fastcall TCPPOTATemplateWizard::Execute() {
  DoNotification("... Hello Dave!");
  MessageDlg("Hello Dave... How are you...", mtInformation, TMsgDlgButtons() << mbOK, 0);
}

UnicodeString __fastcall TCPPOTATemplateWizard::GetMenuText() {
  return "My CPP OTA Template Menu";
}

void __fastcall TCPPOTATemplateWizard::BeforeSave() {
  inherited::BeforeSave();
}

void __fastcall TCPPOTATemplateWizard::AfterSave() {
  inherited::AfterSave();
}

void __fastcall TCPPOTATemplateWizard::Destroyed() {
```

```
  inherited::Destroyed();
}

void __fastcall TCPPOTATemplateWizard::Modified() {
  inherited::Modified();
}
```

Most of the methods just call their inherited versions or the `DoNotification()` method to output a message to the message view however the `QueryInterface()` method checks for the `IOTAWizard` and `IOTAMenuWizard` interfaces before calling the inherited method.

Implementing a Package

Now for the fun bit and getting something working. My sugguest to all of you is DO NOT install the package into you IDE but debug it in a second IDE using a commandline of `-rCPPOTATemplatePkg` so that if you get something wrong you don't blow up your IDE.

So we need a new unit in our package to contain the registration code. It will also contain the DLL code later on.

I found when I first tried this that I could not get a package to be recognised by the IDE and its because I had not defined the namespace correctly in which the `Register()` function is declared. So the declaration is as follows:

```
pragma hdrstop

#include <CPPOTATemplatePkgDLLInit.h>
#include <CPPOTATemplateMainWizard.h>
#pragma package(smart_init)

#ifndef DLL
// For Packages...
// We need to declare for a package a Register procedure.
// The NAMESPACE MUST BE the same name as unit Register is declared in and be lower case except
// for first letter.
namespace Cppotatemplatepkgdllinit {
  void __fastcall PACKAGE Register() {
    RegisterPackageWizard(new TCPPOTATemplateWizard("TCPPOTATemplateWizard"));
  }
}
#else
#endif
```

It took me a while to work out what was wrong. The namespace above needs to be ALL in lowercase letters except for the first letter and MUST be the name of the CPP unit in which it appears NOT the name of the Package (this is described in the C++ Builder Developer's Guide reference above). The `Register()` function must also be with a capital letter. The function simply creates an instance of our wizard and passes it to the `RegisterPackageWizard()` method of the IDE.

You should now be able to compile and run the package in a secondary IDE (Note: you will have to install the package once the secondary IDE has loaded using the Installed Packages menu item). You should find that there is a new menu item under Help | Help Wizards for the package entitled My CPP OTA Template Menu. By pressing the menu you should get a message in the message view and a dialogue box with a message.

Implementing a DLL

So for a DLL create a new blank DLL in C++ builder and add to the project the above files (Notifier, Wizard, Macros and the unit containing the package registration code). You will note that the package code contained an `#ifndef DLL` statement. We need different code for the DLL so in your DLL's project options add a compiler definition for DLL. Also make it use the packages VCL, RTL and DesignIDE. I'll come back to the topic before the end.

Now in the empty part of the `#ifndef DLL` in our unit for the package code added the below code for the DLL as follows:

```
// For DLLs...
// We need to declare a local variable to accept the BorlandIDEServices reference from the
// Wizard creation method below
_di_IBorlandIDEServices LocalIDEServices;

// We also need to delcare the wizard entry point that is called by the IDE on loading a DLL
extern "C" bool __stdcall __declspec(dllexport) INITWIZARD0001(
  const _di_IBorlandIDEServices Service,
  TWizardRegisterProc RegisterWizard,
  TWizardTerminateProc&)
{
  LocalIDEServices = Service; // get reference to the BorlandIDEServices
  RegisterWizard(new TCPPOTATemplateWizard("TCPPOTATemplateWizard"));
  return true;
```

```
}
```

Here we need to define a very specific function which the IDE looks for when loading DLLs. The function has 3 parameters. The first is a reference to the IDE's `BorlandIDEServices` variable. The second is a function we must use to register our wizard(s) and the third is a method for unregistering wizards.

We need to use the second parameter to register our wizard with the IDE (it will be unregistered for us when the IDE closes). The function must return `true`.

Now you're thinking I'll compile this and run it in a secondary IDE (using the command line `-rCPPOTATemplateDLL`) and all will be wonderful. It will compile BUT it will NOT link! According to the documentation referred to above you should be able to compile the DLL with run-time packages including the DesignIDE package which has the external references required however I have not been able to do this and the original examples did not do this either so I'm wondering whether this is possible. The package does have this problem as it does link to the DesignIDE package.

So we need a work around using a macro as follows:

```
#ifdef DLL
#define BorlandIDEServices LocalIDEServices
extern _di_IBorlandIDEServices LocalIDEServices;
#endif
```

I've placed my macros in their own unit so they can be made available to all units in the project. The above macro essentially fixes the missing reference by binding the local `BorlandIDEServices` reference in the macro to the `LocalIDEServices` variable defined in the DLL code above.

Once you've added this to your project then you should be able to compile and run you DLL with the same results as the package.

Adding a Splash Screen

Below is the code for displaying an item in the splash screen. Its implemented in a method which is also declared in the header file so it can be accessed from the main wizard. Then place this method call in the wizard's constructor.

Note: You can only use this in a package not a DLL as you will get an unresovled external reference for the `IOTASplashScreenServices` variable. Thus you will have to `#ifdef` the below and the call in the wizard's constructor. We cannot patch this missing variable as we are not passed this information in the DLLs intialisation method.

The below also uses constants defined in a the `CPPOTATemplateConstants.h` file along with a function to get the package/DLL build information defined in the `CPPOTATemplateFunctions.h` file.

```
pragma hdrstop
#include "CPPOTATemplateSplashScreen.h"
#include "windows.h"
#include "CPPOTATemplateConstants.h"
#include "SysInit.hpp"
#include <ToolsAPI.hpp>
#include "SysUtils.hpp"
#include "Forms.hpp"
#include "CPPOTATemplateFunctions.h"
pragma package(smart_init)

ifndef DLL
void __fastcall AddSplashScreen() {
  int iMajor;
  int iMinor;
  int iBugFix;
  int iBuild;
  HBITMAP bmSplashScreen;
  BuildNumber(iMajor, iMinor, iBugFix, iBuild);
  bmSplashScreen = LoadBitmap(HInstance, L"CPPOTATemplateSplashScreenBitMap24x24");
  _di_IOTASplashScreenServices SSServices;
  if (SplashScreenServices->Supports(SSServices)) {
    String strRev = strRevision[iBugFix];
    SSServices->AddPluginBitmap(
      Format(strSplashScreenName, ARRAYOFCONST((iMajor, iMinor, strRev, Application->Title))),
      bmSplashScreen,
      False,
      Format(strSplashScreenBuild, ARRAYOFCONST((iMajor, iMinor, iBugFix, iBuild)))
    );
    Sleep(1000); //: @debug Here to pause splash screen to check icon
  }
}
```

```
#endif
```

Note: You will need to add the `CPPOTATemplateSplashScreenIcons.rc` file to your project and place the provided icons in an appropriate sub-directory. See the download section below on details about getting the source code.

For a reference for creating splash screen please refer to Chapter 9: Aboutbox Plugins and Splash Screens.

Adding an About Box Entry

Unlike a splash screen entry an about box entry should be installed and uninstalled so there are 2 methods here. Since these are installed and uninstalled using thr `BorlandIDEServices` variable there are no restrictions on whether this appears in a package or DLL. Each method below is also declared in the header to provide access to the methods in the wizard.

To install the about box entry add a call to the below method to the wizard's constructor.

```
#pragma hdrstop
#include "CPPOTATemplateAboutBoxPlugin.h"#
#include <ToolsAPI.hpp>
#include "CPPOTATemplateFunctions.h"#
#include "CPPOTATemplateConstants.h"#
#include "CPPOTATemplateMacros.h"#
#pragma package(smart_init)

int __fastcall AddAboutBoxPlugin() {
  int iMajor;
  int iMinor;
  int iBugFix;
  int iBuild;
  HBITMAP bmAboutBoxPlugin;
  int iAboutBoxPlugin;
  BuildNumber(iMajor, iMinor, iBugFix, iBuild);
  bmAboutBoxPlugin = LoadBitmap(HInstance, L"CPPOTATemplateSplashScreenBitMap48x48");
  _di_IOTAAboutBoxServices ABServices;
  if (BorlandIDEServices->Supports(ABServices)) {
    String strRev = strRevision[iBugFix];
    iAboutBoxPlugin = ABServices->AddPluginInfo(
      Format(strSplashScreenName, ARRAYOFCONST((iMajor, iMinor, strRev, Application->Title))),
      strAboutBoxDescription,
      bmAboutBoxPlugin,
      false,
      Format(strSplashScreenBuild, ARRAYOFCONST((iMajor, iMinor, iBugFix, iBuild))),
      Format("SKU Build %d.%d.%d.%d", ARRAYOFCONST((iMajor, iMinor, iBugFix, iBuild)))
    );
    return iAboutBoxPlugin;
  }
  return -1;
}
```

To uninstall the about box entry add a call to the below method to the wizard's destructor.

```
void __fastcall RemoveAboutBoxPlugin(int iPluginIndex) {
  _di_IOTAAboutBoxServices ABServices;
  if (BorlandIDEServices->Supports(ABServices)) {
    if (iPluginIndex > -1)
      ABServices->RemovePluginInfo(iPluginIndex);
  }
}
```

For a reference for creating abpout box entries please refer to Chapter 9: Aboutbox Plugins and Splash Screens.

Adding an IDE Notifier

I didn't try and pick an awkward notifier to implement, I just pick one that I thought would be most interesting however as you can see from below the notifier has been added to over the years so I've implemented to most recent, `IOTAIDENotifier80` so that I get all the functionality.

Declaration

Below is the declaration which is similar to the wizard that has gone before. Note also that I've declared 2 methods to install and remove the notifier.

```
#ifndef CPPOTATemplateIDENotifierH
```

```
#define CPPOTATemplateIDENotifierH

#include <ToolsAPI.hpp>
#include <CPPOTATemplateNotifierObject.h>

class PACKAGE TCPPOTATemplateIDENotifier : public TDGHNotifierObject, public IOTAIDENotifier80 {
  typedef TDGHNotifierObject inherited;
  public:
    __fastcall TCPPOTATemplateIDENotifier(String strObjectName);
    __fastcall  TCPPOTATemplateIDENotifier();
    // IOTAIDENotifier
    void __fastcall FileNotification(TOTAFileNotification NotifyCode,
      const System::UnicodeString FileName, bool &Cancel);
    void __fastcall BeforeCompile(const _di_IOTAProject Project, bool &Cancel);
    void __fastcall AfterCompile(bool Succeeded);
    // IOTAIDENotifier50
    void __fastcall BeforeCompile(const _di_IOTAProject Project,
      bool IsCodeInsight, bool &Cancel);
    void __fastcall AfterCompile(bool Succeeded, bool IsCodeInsight);
    // IOTAIDENotifier80
    void __fastcall AfterCompile(const _di_IOTAProject Project,
      bool Succeeded, bool IsCodeInsight);
    // IOTANotifer
    void __fastcall BeforeSave();
    void __fastcall AfterSave();
    void __fastcall Destroyed();
    void __fastcall Modified();
  protected:
    // IInterface
    virtual HRESULT __stdcall  QueryInterface(const GUID&, void**);
    virtual ULONG __stdcall  AddRef();
    virtual ULONG __stdcall  Release();
  private:
};

int __fastcall AddIDENotifier();
void __fastcall RemoveIDENotifier(int iIDENotifierIndex);
#endif
```

Implementation

Below is the implementation of the class:

```
#pragma hdrstop
#include <CPPOTATemplateIDENotifier.h>
#include "CPPOTATemplateMacros.h"
#include <SysUtils.hpp>
#pragma package(smart_init)

__fastcall TCPPOTATemplateIDENotifier::TCPPOTATemplateIDENotifier(String strObjectName) :
  TDGHNotifierObject(strObjectName) {
  // Do nothing constructor
}

__fastcall TCPPOTATemplateIDENotifier:: TCPPOTATemplateIDENotifier() {
  // Do nothing deconstructor
}

void __fastcall TCPPOTATemplateIDENotifier::FileNotification(
  TOTAFileNotification NotifyCode, const System::UnicodeString FileName, bool &Cancel) {
  const String strNotifyCode[11] = {
    "ofnFileOpening",
    "ofnFileOpened",
    "ofnFileClosing",
    "ofnDefaultDesktopLoad",
    "ofnDefaultDesktopSave",
    "ofnProjectDesktopLoad",
    "ofnProjectDesktopSave",
    "ofnPackageInstalled",
    "ofnPackageUninstalled",
    "ofnActiveProjectChanged",
```

```
    "ofnProjectOpenedFromTemplate"
  };
  DoNotification(
    Format("FileNotification: NotifyCode(%d): %s, FileName: %s",
      ARRAYOFCONST((
        NotifyCode,
        strNotifyCode[NotifyCode],
        ExtractFileName(FileName)
      ))
    )
  );
}

void __fastcall TCPPOTATemplateIDENotifier::BeforeCompile(const _di_IOTAProject Project,
  bool &Cancel) {
  DoNotification(
    Format("::BeforeCompile: Project: %s",
      ARRAYOFCONST((ExtractFileName(Project->FileName)))
    )
  );
}

void __fastcall TCPPOTATemplateIDENotifier::AfterCompile(bool Succeeded) {
  DoNotification(
    Format("::AfterCompile: %s",
      ARRAYOFCONST((
        Succeeded   "True" : "False"
      ))
    )
  );
}

void __fastcall TCPPOTATemplateIDENotifier::BeforeCompile(const _di_IOTAProject Project,
  bool IsCodeInsight, bool &Cancel) {
  DoNotification(
    Format("50::BeforeCompile: Project: %s, IsCodeInsight: %s",
      ARRAYOFCONST((
        ExtractFileName(Project->FileName),
        IsCodeInsight   "True" : "False"
      ))
    )
  );
}

void __fastcall TCPPOTATemplateIDENotifier::AfterCompile(bool Succeeded,
  bool IsCodeInsight) {
  DoNotification(
    Format("50::AfterCompile: Succeeded: %s, IsCodeInsight: %s",
      ARRAYOFCONST((
        Succeeded   "True" : "False",
        IsCodeInsight   "True" : "False"
      ))
    )
  );
}

void __fastcall TCPPOTATemplateIDENotifier::AfterCompile(const _di_IOTAProject Project,
  bool Succeeded, bool IsCodeInsight) {
  DoNotification(
    Format("80::AfterCompile: Project: %s, Succeeded: %s, IsCodeInsigh: %s",
      ARRAYOFCONST((
        ExtractFileName(Project->FileName),
        Succeeded   "True" : "False",
        IsCodeInsight   "True" : "False"
      ))
    )
  );
}

void __fastcall TCPPOTATemplateIDENotifier::BeforeSave() {
  inherited::BeforeSave();
```

```
  DoNotification("::BeforeSave");
}

void __fastcall TCPPOTATemplateIDENotifier::AfterSave() {
  inherited::AfterSave();
  DoNotification("::AfterSave");
}

void __fastcall TCPPOTATemplateIDENotifier::Destroyed() {
  inherited::Destroyed();
  DoNotification("::Destroyed");
}

void __fastcall TCPPOTATemplateIDENotifier::Modified() {
  inherited::Modified();
  DoNotification("::Modified");
}

HRESULT __stdcall TCPPOTATemplateIDENotifier::QueryInterface(const GUID& iid, void** obj) {
  QUERY_INTERFACE(IOTAIDENotifier50, iid, obj);
  QUERY_INTERFACE(IOTAIDENotifier80, iid, obj);
  QUERY_INTERFACE(IOTAIDENotifier, iid, obj);
  return inherited::QueryInterface(iid, obj);
}

ULONG __stdcall TCPPOTATemplateIDENotifier::AddRef() {
  return inherited::AddRef();
}

ULONG __stdcall TCPPOTATemplateIDENotifier::Release() {
  return inherited::Release();
}
```

In the above most methods either call their inherited method or output a notification to show the method has been called. As before with the wizard the `QueryInterface` method uses the macro `QUERY_INTERFACE` to dispatch incoming calls for each of the 3 notifier interfaces else allow the inherited version to be called. It should also be noted that unlike Delphi all versions of the overloaded methods are called not just the latest implementation. This is different behaviour to Delphi.

The below method is used to install the notifier into the IDE. This should be installed in the wizard's constructor.

```
int __fastcall AddIDENotifier() {
  _di_IOTAServices IDEServices;
  if (BorlandIDEServices->Supports(IDEServices)) {
    return IDEServices->AddNotifier(new TCPPOTATemplateIDENotifier("TCPPOTATemplateIDENotifier"));
  }
  return -1;
}
```

The below method is used to uninstall the notifier from the IDE. This should be uninstalled in the wizard's destructor.

```
void __fastcall RemoveIDENotifier(int iIDENotifierIndex) {
  _di_IOTAServices IDEServices;
  if (BorlandIDEServices->Supports(IDEServices)) {
    if (iIDENotifierIndex > -1) {
      IDEServices->RemoveNotifier(iIDENotifierIndex);
    }
  }
}
```

For more information on this notifier please refer to Chapter 7: IDE Compilation Events.

Adding AutoSave Functionality

The auto save functionality is quite simple. Add a `TTimer` to your wizard's constructor setting the method below as its event handler and set the timer interval to 1 second (1000 milliseconds) and set it's enabled property to `true`. You will also need a counter to count the number of seconds between saves.

The below method then saves modified files, if enabled, by calling the `SaveModifiedFiles()` method.

```
void __fastcall TCPPOTATemplateWizard::AutoSaveTimerEvent(TObject* Sender) {
  FTimerCounter++;
```

```
  if (FTimerCounter >= FAppOptions->AutoSaveInterval) {
    FAutoSaveTimer->Enabled = false;
    try {
      FTimerCounter = 0;
      if (FAppOptions->EnableAutoSave) {
        SaveModifiedFiles();
      }
    } __finally {
      FAutoSaveTimer->Enabled = true;
    }
  }
}
```

The below method does the work of saving the files by getting a buffer iterator from the Editor Services, checking each file for being modified and if so saving the file.

```
void __fastcall TCPPOTATemplateWizard::SaveModifiedFiles() {
  _di_IOTAEditBufferIterator Iterator;
  _di_IOTAEditorServices EditorServices;
  if (BorlandIDEServices->Supports(EditorServices)) {
    if (EditorServices->GetEditBufferIterator(Iterator)) {
      for (int i = 0; i < Iterator->Count; i++) {
        if (Iterator->EditBuffers[i]->IsModified) {
          if (Iterator->EditBuffers[i]->Module->Save(false,  FAppOptions->PromptOnAutoSave)) {
            DoNotification(
              Format("... Auto Saved: %s",
                ARRAYOFCONST((ExtractFileName(Iterator->EditBuffers[i]->FileName)))
              )
            );
          }
        }
      }
    }
  }
}
```

For more information on this please refer to Chapter 2: A simple custom menu (AutoSave) and Chapter 3: A simple custom menu (AutoSave) Fixed.

Creating an IDE Options frame for the AutoSave Options

Finally I've added an AutoSave options frame to the IDE's options dialogue as below (I've skipped over the frame creation and code as its so simple and been covered in other chapters – see below).

Declaration

Below is the declaration of an Add-in Options class to allow the IDE to create and interact with our options frame for the AutoSave functionality. Also declared below are 2 methods to install and uninstall the dialogue from the IDE.

```
#ifndef CPPOTATemplateAddinOptionsH
#define CPPOTATemplateAddinOptionsH

#include <ToolsAPI.hpp>
#include "CPPOTATemplateAppOptionsFrame.h"#
#include "CPPOTATemplateAppOptions.h"#
#include "CPPOTATemplateNotifierObject.h"

class PACKAGE TCPPOTATemplateAddInOptions : public TDGHNotifierObject, public INTAAddInOptions {
  typedef TDGHNotifierObject inherited;
  private:
    TCPPOTATemplateOptions* FOptions;
    TframeAppOptions* FAppOptionsFrame;
  protected:
    // IOTANotifier
    void __fastcall BeforeSave();
    void __fastcall AfterSave();
    void __fastcall Destroyed();
    void __fastcall Modified();
    // IInterface
    virtual HRESULT __stdcall QueryInterface(const GUID& iid, void** obj);
    virtual ULONG __stdcall AddRef();
```

```
    virtual ULONG __stdcall Release();
    // INTAAddinOptions
    UnicodeString __fastcall GetArea();
    UnicodeString __fastcall GetCaption();
    TCustomFrameClass __fastcall GetFrameClass();
    void __fastcall FrameCreated(TCustomFrame* AFrame);
    void __fastcall DialogClosed(bool Accepted);
    bool __fastcall ValidateContents();
    int __fastcall GetHelpContext();
    bool __fastcall IncludeInIDEInsight();
  public:
    TCPPOTATemplateAddInOptions(String strObjectName, TCPPOTATemplateOptions* Options);
};

TCPPOTATemplateAddInOptions* __fastcall AddOptionsFrameToIDE(TCPPOTATemplateOptions* Options);
void __fastcall RemoveOptionsFrameFromIDE(TCPPOTATemplateAddInOptions* IDEOptions);

 #endif
```

Implementation

Below is the implementation of the above class:

```
pragma hdrstop
include "CPPOTATemplateAddinOptions.h"#
include "CPPOTATemplateMacros.h"#
pragma package(smart_init)

TCPPOTATemplateAddInOptions::TCPPOTATemplateAddInOptions(String strObjectName,
  TCPPOTATemplateOptions* Options) : TDGHNotifierObject(strObjectName) {
  FOptions = Options;
}

UnicodeString __fastcall TCPPOTATemplateAddInOptions::GetArea() {
  return "";
}

UnicodeString __fastcall TCPPOTATemplateAddInOptions::GetCaption() {
  return "C++ OTA Template.AutoSave Options";
}

TCustomFrameClass __fastcall TCPPOTATemplateAddInOptions::GetFrameClass() {
  return __classid(TframeAppOptions);
}

void __fastcall TCPPOTATemplateAddInOptions::FrameCreated(TCustomFrame* AFrame) {
  FAppOptionsFrame = dynamic_cast<TframeAppOptions*>(AFrame);
  if (FAppOptionsFrame  = NULL) {
    FAppOptionsFrame->LoadFrame(FOptions);
  }
}

void __fastcall TCPPOTATemplateAddInOptions::DialogClosed(bool Accepted) {
  Accepted = true;
}

bool __fastcall TCPPOTATemplateAddInOptions::ValidateContents() {
  if (FAppOptionsFrame  = NULL) {
    FAppOptionsFrame->SaveFrame(FOptions);
  }
  return true;
}

int __fastcall TCPPOTATemplateAddInOptions::GetHelpContext() {
  return 0;
}

bool __fastcall TCPPOTATemplateAddInOptions::IncludeInIDEInsight() {
  return true;
}
```

```
void __fastcall TCPPOTATemplateAddInOptions::BeforeSave() {
  inherited::BeforeSave();
}

void __fastcall TCPPOTATemplateAddInOptions::AfterSave() {
  inherited::AfterSave();
}

void __fastcall TCPPOTATemplateAddInOptions::Destroyed() {
  inherited::Destroyed();
}

void __fastcall TCPPOTATemplateAddInOptions::Modified() {
  inherited::Modified();
}

HRESULT __stdcall TCPPOTATemplateAddInOptions::QueryInterface(const GUID& iid, void** obj) {
  QUERY_INTERFACE(INTAAddInOptions, iid, obj);
  return inherited::QueryInterface(iid, obj);
}

ULONG __stdcall TCPPOTATemplateAddInOptions::AddRef() {
  return inherited::AddRef();
}

ULONG __stdcall TCPPOTATemplateAddInOptions::Release() {
  return inherited::Release();
}
```

Most of the above is the same as you would for Delphi. The `QueryInterface` method has an additional statement to handle this classes interface (`IOTAAddInOptions`). The only other note worthy method is the one that returns the frame class reference, this is achieved using the `__classid()` method.

The below method installs the dialogue into the IDE and should be called from the wizard's constructor.

```
TCPPOTATemplateAddInOptions* __fastcall AddOptionsFrameToIDE(TCPPOTATemplateOptions* Options) {
  _di_INTAEnvironmentOptionsServices EnvOpServices;
  if (BorlandIDEServices->Supports(EnvOpServices)) {
    TCPPOTATemplateAddInOptions* IDEOptions =
      new TCPPOTATemplateAddInOptions("TCPPOTATemplateAddInOptions", Options);
    EnvOpServices->RegisterAddInOptions(IDEOptions);
    return IDEOptions;
  }
  return NULL;
}
```

The below method uninstalls the dialogue from the IDE and should be called from the wizard's destructor.

```
void __fastcall RemoveOptionsFrameFromIDE(TCPPOTATemplateAddInOptions* IDEOptions) {
  _di_INTAEnvironmentOptionsServices EnvOpServices;
  if (BorlandIDEServices->Supports(EnvOpServices) && (IDEOptions  = NULL))  {
    EnvOpServices->UnregisterAddInOptions(IDEOptions);
  }
}
```

For more information on creating options dialogues for the IDE please refer to Chapter 17: Options Page(s) inside the IDE's Options Dlg.

After Thoughts

The above has been a little rushed as I only have today to write this up so if you find errors I've not corrected or you think there is information missing let me know and I'll update the post.

For me it would seem that implementing the Open Tools API in C++ Builder is a little more difficult than the same implementation with Delphi however you should be able to do all the same things with C++ Builder that you can do with Delphi.

It would be nice if Embarcadero could investigate why DLLs cannot link to the DesignIDE package as that would allow splash screens in DLL. I'll raise a QC report for this.

One other thing that is mentioned in the C++ documentation is that if you use a DLL, don't use run-time packages and stick to `IOTAXxxx` interfaces you can build a DLL that can work with ANY version of the IDE. This is something that I'm going to try next as I want to added Code Completion and the <Ctrl>+<shift>+Up/Down navigation keys to C++ Builder as I miss these. This doesn't seem limited to C++ Builder, it can be

done with Delphi. I'm sure someone will ask why you can't use the `INTAXxxx` interfaces? These are specific to the RAD Studio you are working with and there is no guarentee they will work with other versions. Think `VTable` changes between different versions of RAD Studio. Also dockable forms might be out although I seem to remember there is a method where you can ask the IDE for a dockable form (Nope – scratch that – just checked and its a method of a native interface 😕) and may be you can embed a frame within it.

Enjoy!

Downloads

All of the code for this example can be downloaded from the page C++ OTA Template.

Related posts:

1. Notify me of everything… – Part 1 (19.6)
2. C++ OTA Template Code (19)
3. The Delphi Open Tools API Book (18.2)
4. Chapter 7: IDE Compilation Events (15.3)
5. Chapter 5: Useful Open Tools Utility Functions (14.8)

Category: C++ OTA Template Open Tools API RAD Studio Tags: Borland, BorlandIDEServices, C++ Builder, CodeGear, Delphi, Embarcadero, Experts, IOTAEditBufferIterator, IOTAEditorServices, IOTAIDENotifier, IOTAIDENotifier50, IOTAIDENotifier80, IOTAMessageServices, IOTANotifier, IOTAWizard, IOTAWizardMenu, OTA, RAD Studio, _di_INTAEnvironmentOptionsServices, _di_IOTAAboutBoxServices, _di_IOTABorlandIDEServices, _di_IOTAEditBufferIterator, _di_IOTAEditorServices, _di_IOTAMessageGroup, _di_IOTAMessageServices, _di_IOTAServices, _di_IOTASplashScreenServices