LeXtudio OpenTools SDK Documents

Bilingual Version

PRELIMINARY DRAFT

Lex Mark

March 23, 2006

Disclaimer

Trademarks

Contents

Part I

English

Chapter 1

LeXDK Developer's Guide

1.1 General Notes

This SDK is based on David Hervieux' project Sharp Builder Tools 3.1.0.0, so it contains the "Hervieux DNA".

1.2 History Background

At the beginning stage, the Delphi language did not support interfaces. When interfaces were added, Borland R&D provides programmers Open Tools Architecture (for Win32) to extending the IDE. This OTA is too powerful to miss. However, it is hard to master.

Mark Miller, author of famous OTA product CodeRush, invented a series of API himself, named RushAPI (for Win32). Because CodeRush is a commercial product, no source code is available and I don't know what languages were used to develop it (C++ or Delphi). Before Delphi 8 was shipped, Mike's company was merged with DevExpress. Since then no breaking version of CodeRush had come except the ones for Visual Studio .NET. You cannot use CodeRush or RushAPI to extending Delphi 8/2005/2006. It is a pity.

Other commercial OTA products such as Castalia has implemented similar architectures. However, they are not open to other developers except the producers' companies.

Open source OTA tools like GExperts and CnPack don't have such plug-in architecture but they are easy to extend, too. There may be some open source OTA architectures/SDK but I don't meet.

First introduced in C#Builder 1.0, Borland Developer Studio IDE offers OTA for .NET. David Hervieux created Sharp Builder Tools for C#Builder since its version 1. The latest version of SBT is 3.1. This version is too old to support Delphi 2005/2006 (in fact after a few modifications it can). Although David won the OTA competition held by Borland, he failed to continue this project for "too many problem with Borland".[1] His codebase is written in C#.

[1]See License.pdf for details.

I build Code Beautifier Collection 2 (and its ancestor JCFExpert) using SBT codebase. At that moment, I found it really easy to extending SBT but I have to mix my code with David's. It showed that David had not completed the architecture. Inspired by his ideas, I bring the codebase further to make an SDK for building OpenTools. Although not as power as RushAPI, it really reduces the complexity of extending the BDS IDE. I build Code Beautifier Collection 2 using this SDK. This SDK is named as LeXtudio OpenTools SDK, LeXDK for short.

Somebody may fear to use this SDK because they think it reduces their freedom. Actually, LeXDK does not prevent you from developing directly with BDS OTA. You can directly use OTA when you develop a feature.

1.3 Terms

The codebase Here it means Sharp Builder Tools 3.1.0.0.

Hervieux DNA It also means Sharp Builder Tools 3.1.0.0.

LeXDK It is contained in the assembly Lextm.LeXDK.Core.dll.

It consists of a few useful classes.

The Code Beautifier Collection 2 (versions above 5) is built on this SDK.

Minus An assembly that extends the SDK library is a Minus.

Plus The assemblies you develop with LeXDK.

They are called "Plus" for CBC 2/LeXDK.

Feature Certain classes contained in your Plus.

They are subclasses of CustomOtaFeature.

They should be called "Features".

LWT LeXtudio Wrapping Technology.

It is applied in some Plus written by Lex Mark, such as C#Builder Goodies Wrapper and AddMany Wrapper.

1.4 Procedures

The CBC itself and its first Plus Pack are the best demos I can give you. so if there are problems you can not find a solution, you can contact me or resort to Google or MSDN.

The classes and functions are detailed in the SDK Document in CHM format.

Think more about your intentions before the start.

The Plus I wrote, are used as the examples here, such as the C#Builder Goodies Wrapper (Lextm.CSBuilderGoodies.Plus.dll).

1.4.1 How to Start

Every Plus should contain at least one feature. C#Builder Goodies Wrapper contains the feature of C#Builder Goodies 1.1.

Your First Plus

Make an empty project in BDS or SharpDevelop. Name it as Lextm.Code-BeautifierCollection.CSBuilderGoodies.

Your First Feature

Add a class named OtaCSBuilderGoodies. Its base class should be Custo-mOtaFeature. This base class implements some useful functions.

1.4.2 How to Make a Feature

The class you create in last step is empty. In order to complete it, follow me.

Add a Name

I suggest you add a name to every feature you write.

It can be like this.

//private const string Name = "CsbGoodies";

A commented one is OK because it may not be used. But in the future, maybe names are required.

1.4.3 How to Configure a Menu Item

Override a function named IDERegisterMenus, and your menus should be created here.

The debug lines are used when debugging. You may not need these lines.

Call base function although it is empty in fact.

RegisterMenu function call will register the menu for you. Send a menu object as its parameter.

A menu object can be created by calling a few functions. CreateActionMenu returns a menu item that can do something (other types are EmytyMenu, SeparatorMenu, and PlaceHolderMenu).

Here, a Doc Preview menu item is created by calling CreateActionMenu. It is a child of the base menu MenuLeXtudio. It s name is MenuViewDoc. Its shortcut is DefaultViewDocShortcut, and the text is MenuTextViewDoc. When you click the menu item finally in BDS, the handler DoViewDoc is called.

```
private const int DefaultViewDocShortcut = 32849; // Alt + Q
private const string MenuViewDoc = "CBCExpertViewDocMenu";
private const string MenuLeXtudio = "CBCLeXtudioMenu";
private const string MenuTextViewDoc = "View Doc";
/// <summary>
/// Registers menus.
/// </summary>
public override void IDERegisterMenus()
{
base.IDERegisterMenus();
Debug.Indent();
Debug.WriteLine("–> Enter
OtaCsbGoodies.IDEReigsterActions");
// View Doc
RegisterMenu(
CreateActionMenu(
OTAMenuItemLocation.otamlChild,
MenuLeXtudio,
MenuViewDoc,
DefaultViewDocShortcut,
MenuTextViewDoc,
new EventHandler(DoViewDoc)
)
);
Debug.WriteLine("<– Leave
OtaCsbGoodies.IDERegisterActions");
Debug.Unindent();
}
private static void DoViewDoc(object sender, EventArgs e) {
CSBuilderGoodies.QuickdocViewer.getInstance().Activate();
}
```

Here in DoViewDoc, I call a function of CSBuilderGoodies.QuickdocViewer by first getting its singleton object. The Doc Preview Form is shown when this function, Activate, is called.

1.4.4 How to Add a Tab Page to Preferences Dialog

With a Tab Page

You can see Code Beautifiers Plus source. This Plus adds two Tabs. If you add a Tab, you should also add a corresponding Page to the PageControl.

Without a Tab Page

If you know CnPack or GExperts, you may like to have a separate dialog to configure your feature without registering in a FormPreferences. It is okay.

Add a dialog in your feature, and create a menu item to display it. So it is very easy.

1.4.5 How to Manage Preferences for a Feature

There are a lot of ways to store preferences, Registry and INI files are old-fashion. I strongly recommend that you should use an XML file instead.

Using XML Files Method I (Serialization and Deserialization)

The Code Beautifiers Plus uses this method to store data. The feature OtaCodeBeautifiers has a subclass named preferences which is serializable. You can read the source for details as well as .NET SDK Documentation on that subject.

Using XML Files Method II (Manipulating Elements)

The ShortcutArrayList class shows this method. Before loading or saving a shortcut, the subclass Shortcuts first finds a correct element in the XML file and then manipulate its inner text. You can read the source for details as well as .NET SDK Documentation on that subject.

1.4.6 More...

Remember that this is an SDK for .NET OTA, any .NET languages can be used and mix-language programming is allowed. For example, AddMany 4.1 is totally in Delphi for .NET and the wrapper is purely in C#.

The SDK itself can be extended by adding more classes assemblies, or called Minus (which reduces OTA complexity).

You can even extends or reimplement OtaFramework and Lextm.CodeBeautifierCollection.Main in the framework using LeXtudio Wrapping Technology.

More functions can be overridden in order to extend your features list.

You can even enhance the SDK to build more powerful features.

If you don't decide to use this SDK because of its GPL blood, you may use SBT source directly or resort to Borland OTA API calls. I choose GPL to cover this piece in order to keep it open source for ever.

This SDK now is still in its early age. Many interesting features or libraries I write are not mature enough to be added in. So keep waiting, or take part in. If you write a Minus or Plus, send it to me (with or without source). I can publish the news on the homepage of CBC to let others know of your work.

Let us make a community of our own.

1.5 References

- Borland Help "Extending the IDE"
- GExperts source files
- Sharp Builder Tools source files
- Code Beautifier Collection 2 source files
- Google or MSDN ...

Chapter 2

LeXDK Class Reference

2.1 How I make it

This chapter is automatically generated by using NDoc.

2.2 Contents

(Not available in this format now. When NDoc is ready I will be ready.)

(NDoc generates a complete document, so cannot be embedded here directly. As a result, you should see the created PDF file.)

Chapter 3

Plus Registration File Format Specifications v.I

3.1 Why Version I is Simple

Actually, I knew little about file manipulations before I designed this architecture. So I have to first do some experiments in order to make myself familiar with this staff. A simple version I of this file format can already provide basic configuration ability for Plus. As a result, I take it as the specifications.

Later version should be more complex, but still easy to configure.

3.2 File Extension

.plus is the file extension.

3.3 How to Generate

The file should be a text file.

The file name should be the same with the corresponding Plus assembly file.

Each lines is for a feature.

One line should contains two sections separated by a semicolon.

The first section is the name of the feature class.

The second is a flag. Set it to true for enabled and false for disabled.

> For example, this is how I write for Code Beautifiers Plus.
>
> The file name should be Lextm.CodeBeautifiers.Plus.plus.
>
> In the file there is only one line:
>
> Lextm.CodeBeautifierCollection.Ota.OtaCodeBeautifiers;true

Chapter 4

Plus Registration File Format Specifications v.II (DRAFT)

4.1 Undo in Version I

Basic functions are done in version I. However, it is not easy to configure a feature freely even with a GUI tool. The file format lacks a few flexibility in fact. So this version II should go further.

It is sometimes comfortable to enable one feature in BDS 1 and disabled it in BDS 2. So Version II takes this need into account.

Also, there are a few calls for personality level enabled/disabled control. Such a further requirement may be included in this version or in a later version.

In order to make this format more powerful, simple text file may not be suited. So it should be in INI format or XML format.

I am open to discussions.

4.2 File Extension

.plus2 is the file extension.

4.3 How to Generate