LeXtudio OpenTools SDK Developer's Guide

PRELIMINARY DRAFT

Lex Mark

March 14, 2006

Powered by LaTeX.

This file contains important supplementary information that may not appear in the main product documentation. Lex Mark recommends that you read this file in its entirety.
For information about licensing issues, see other files located, by default, at C:\Program Files\Code Beautifier Collection 2\doc.

Contents

1 General Notes

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

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 one for Visual Studio .NET. You cannot use CodeRush or RushAPI to extending Delphi 8/2005/2006. It is a pity.

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#.

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

[1] See License.pdf for details.

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.

3 Terms

The codebase here means Sharp Builder Tools 3.1.0.0.

The LeXDK is contained in the assembly Lextm.CodeBeautifierCollection.Common.dll. It consists of a few classes.

The Code Beautifier Collection 2 (versions above 2.4.5.2) is built on this SDK.

When you develop an assembly with LeXDK, it can be called an "Plus" for CBC 2.

The classes contained in your Plus (subclasses of CustomOtaFeature) should be called "Features".

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.

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

Think more about your intentions before the start.

C#Builder Goodies Wrapper (Lextm.CodeBeautifierCollection.CSBuilderGoodies.dll), the first Plus I wrote, is used as the example here.

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.

4.1.1 Your First Plus

Make an empty project in BDS or SharpDevelop. Name it as Lextm.CodeBeautifierCollection.CSBuilderGoodies.

4.1.2 Your First Feature

Add a class named OtaCSBuilderGoodies. Its base class should be CustomOtaFeature. This base class implements some useful functions.

4.2 How to Make a Feature

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

4.2.1 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.

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 don't 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 function is implemented in this function call, Activate.

4.4 How to Add a Tab Page to Preferences Dialog

4.4.1 With a Tab Page

You can

4.4.2 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.

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.

4.5.1 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.

4.5.2 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.

4.6 More...

More functions can be overriden in order to extend your features.

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. I choose GPL to cover this piece in order to keep it free for ever.

5 References

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