LeXtudio OpenTools SDK Documents

Bilingual Version

PRELIMINARY DRAFT

Lex Mark

March 30, 2006

Disclaimer

Lex Mark has taken due care in preparing this manual and the programs and data on the electronic disk media (if any) accompanying this book including research, development and testing to ascertain their effectiveness.

Lex Mark makes no warranties as to the contents of this manual and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Lex Mark further reserves the right to make changes to the specifications of the program and contents of the manual without obligation to notify any person or organization of such changes.

Newest version of this manual should be shipped in the distribution package but is not guaranteed.

Trademarks

lextm, LeXtudio, LeXDK, Code Beautifier Collection, LeXtudio Wrapping Technology, LeKeys, LanguageX, and LeXtyles are trademarks of Lex Mark. All other products and brand names used in this manual are registered trademarks or tradenames of their respective holders.

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 you start.

The Minus/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 a Plus

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

This wrapper should enable C#Builder Goodies 1.1's function "Doc Preview" in CBC. This is the initial decision.

So the story goes.

Your First Plus

Make an empty project in SharpDevelop (or BDS). Name it as Lextm.CSBuilderGoodies.Plus.

Plus is used as the last part so as to indicate it is a Plus project. Lextm is used because Lex Mark is the author.

Your First Feature

Add a class named OtaCSBuilderGoodies. Its base class should be CustomOtaFeature.

CustomOtaFeature is the base class of every feature I created. It 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";
```

Even a commented one is OK when it is not used in fact.

```
//private const string Name = "CsbGoodies";
```

But in the future, maybe names are required in the registration process.

1.4.3 Configure a Menu Item

This is how C#Builder Goodies Plus adds a "View Doc" to BDS.

```
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();
}
```

At first, 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 in the front (although the base fucntion is currently empty) in case that I add some lines in future versions of LeXDK.

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

A menu object (there are four types currently) can be created by calling a few functions.

CreateActionMenu here returns a menu item that can do something (this item must have an event handler).

Other types are EmytyMenu, SeparatorMenu, and PlaceHolderMenu.

An EmptyMenu example is the LeXtudio menu item. It is just serving as the first menu item in the menu tree. Click it, and no action executes.

A SeparatorMenu will be translated to a separator in the menu.

A PlaceHolderMenu will not be translated to a real menu item. They are place holder when you configure the menu tree storing the menu items. The ROOT of this tree is a good example. If you use this type carefully, you can add menu items to ToolsMenu and FileMenu directly (however, it is not recommended so details are not listed here).

Here, a Doc Preview menu item is created by calling CreateActionMenu. It is a child of the base menu MenuLeXtudio. Its 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.

According to BDS OTA contraints, the menu name, MenuViewDoc, should not contain dots (else a run time error occurs). Also, the shortcut value should be calculated with TShortcut style value calculation. Later I may include such a feature in C#Builder Goodies Plus in order to ease your pain. Yes, this keycode analyzer is there in C#Builder Goodies 1.1 and I don't have time to port/wrap it in.

Here in DoViewDoc, I do some wrapping of C#Builder Goodies 1.1. 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 Register a Tab Page to Global Preferences Dialog

```
public override void IDERegisterTabs() {
base.IDERegisterTabs();
RegisterTab(CreateTabNode(TabAStyle));
RegisterTab(CreateTabNode(TabJcf));
}


private System.Windows.Forms.GroupBox groupBox5;
```

```
//...
public override void IDERegisterComponents( ) {

base.IDERegisterComponents();

this.txtJcfPath = new System.Windows.Forms.TextBox();
//...

TheOtaFramework.TabControl.Controls.Add(this.tbpAStyle);
TheOtaFramework.TabControl.Controls.Add(this.tbpJCF);

this.tbpAStyle.ResumeLayout(false);
//...
}
```

If you add a Tab, you should also add a corresponding Tab Page to the TabControl (however now, I haven't implement a 100% reliable mechanism right now).

Create a Node in the Left Tree

If you want, you can add a node in the global FormPreferences.

You can see Code Beautifiers Plus source code. This Plus adds two Tabs when it overrides IDERegisterTabs. The steps are almost the same with IDERegisterMenus.

Create a Tab Page for a Tab

IDERegisterComponents should be overridden this time. Some WinForms controls should be created manually and added to a TabPage component (the container). At last this container is added to TheOtaFramework's TabControl.

1.4.5 Without Registering Global Preferences Dialog

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

Make such a configuration dialog in your Plus, and create a configuration menu item in your feature to activate it. So it is very easy.

1.4.6 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 because it is easy and quite readable.

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. The settings are loaded and saved during the execution. You can read the source for details as well as .NET SDK Documentation on that subject.

This method is adopted from Sharp Builder Tools.

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 corresponding XML file and then manipulate its inner text. You can read the source for details as well as .NET SDK Documentation on that subject.

This method is taken from C#Builder Goodies 1.1.

1.4.7 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

Plus Registration File Format

2.1 Specifications Version I

2.1.1 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.

2.1.2 File Extension

.plus is the file extension.

2.1.3 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

2.2 Specifications Version II (DRAFT)

2.2.1 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.

2.2.2 File Extension

.plus2 is the file extension.

2.2.3 How to Generate

Part II

简体中文

Chapter 3

LeXDK 开发手册

3.1 概述

这个 SDK 基于 David Hervieux 的 Sharp Builder Tools 3.1.0.0 项目代码开发，所以说它包含了"Hervieux DNA"。

3.2 历史背景

在 Delphi 语言诞生的时候，接口并不是其中一个概念。当接口概念加入之后，Borland R&D 向开发者提供了 Open Tools Architecture (for Win32) 来扩展 IDE 的功能。你不可能忽略 OTA，因为它实在太强大了，只是全面的学习需要太多的时间精力。

Mark Miller ，OTA 架构作品 CodeRush 的作者，发明了一套很不错的 API 来扩展 IDE，称为 RushAPI (for Win32)。由于 CodeRush 是商业软件，所以我没有机会了解它是用何种方式实现的(C++ 还是 Delphi)。在 Delphi 8 发布前，Mike 的公司与 DevExpress 合并。从那时起，除了支持 Visual Studio .NET 的版本，再也没有看到创新性的 CodeRush 版本。所以你无法利用 CodeRush 或者 RushAPI 来扩展 Delphi 8/2005/2006。这很令人遗憾。

其他的商业软件例如 Castalia 也实现了类似的架构，只是除了开发的公司，没有其他人了解其中的奥妙。

开源的 OTA 项目例如 GExperts 和 CnPack 都没有实现这样的插件架构，但是它们仍然很容易扩展。当然或许还有其他的 OTA 或者 SDK，但是我不知道。

从 C#Builder 1.0 开始，Borland Developer Studio IDE 开始提供 OTA for .NET。从那时起 David Hervieux 制作了 Sharp Builder Tools。SBT 最新的版本号是 3.1。当然这个版本太老了，不支持 Delphi 2005/2006 (实际上只需要一点点修改)。尽管 David 在 Borland 主持的 OTA 比赛中夺冠，他没有能够继续这个开源项目，因为"too many problem with Borland"。他使用的开发语言是 C#。

我使用了 SBT 代码来开发 Code Beautifier Collection 2 (以及它的前身 JCFExpert)。那时我发现很容易扩展 SBT 然而我不得不将自己的代码混杂进去。看得出来 David 还来不及完成一个插件架构。受他的想法启迪，我利

用这些代码设计了一个 SDK 用于 OpenTools 的开发。虽然不如 RushAPI 那样强大，但是，它已经可以减少很多扩展 BDS IDE 的难度。Code Beautifier Collection 2 现在也是利用这个 SDK 开发的。LeXtudio OpenTools SDK 就是这个 SDK 的名字。LeXDK 是其简称。

难免有人会害怕使用一个 SDK 来做开发，因为或许这减少了他们的自由发挥。实际上，LeXDK 没有限制你直接利用 BDS OTA 来开发。你甚至可以直接用 OTA 来开发一个插件 feature。

3.3 用语

代码库 这里指的是 Sharp Builder Tools 3.1.0.0。

Hervieux DNA 也是指的 Sharp Builder Tools 3.1.0.0。

LeXDK 包含在 Lextm.LeXDK.Core.dll中的代码。

由一系列的类组成。

Code Beautifier Collection 2 (5 之后的版本)都是利用这个 SDK 开发的。

Minus 扩展 SDK 库的代码库可以被称作 Minus。

Plus 利用 LeXDK 开发的代码库。

它们被称作CBC 2/LeXDK 的”Plus”.

Feature Plus 中的一些特殊类。

它们都是 CustomOtaFeature 的继承类。

它们被称作”Features”。

LWT 全称是 LeXtudio Wrapping Technology。

在由 Lex Mark 所写的几个 Plus 里面有使用过，例如 C#Builder Goodies Wrapper 和 AddMany Wrapper。

3.4 步骤

CBC 和第一个 Plus Pack 是我可以给与你的最好的范例。如果你不能解决你遇到的问题，你可以与我联系，或者利用 Google 或者 MSDN 查到你需要的信息。

类和函数都在 SDK 文档中以 CHM 格式保存。

我以自己所写的 Minus/Plus 为例子，例如 C#Builder Goodies Wrapper (Lextm.CSBuilderGoodies.Plus.dll)。

3.4.1 如何开始一个 Plus

每一个 Plus 都包含至少一个 feature。在我的计划中，C#Builder Goodies Wrapper 包含有一个 feature，和全部 C#Builder Goodies 1.1 的功能。

这个包装类在 CBC 中激活了 C#Builder Goodies 1.1 的"Doc Preview"功能。这是最初的想法。

那么，故事开始了。

第一个 Plus

在 SharpDevelop (或者 BDS)中建立一个空的工程，名称是 Lextm.CSBuilderGoodies.Plus。

Plus 被作为名称的最后一部分，以表明这是一个 Plus 工程。Lextm 被使用是因为 Lex Mark 是作者。

你的第一个 Feature

添加一个名为 OtaCSBuilderGoodies 的类。它的基类是 CustomOtaFeature。

CustomOtaFeature 是我设计的每一个 feature 的基类。它实现了一些有用的函数。

3.4.2 如何完善一个 Feature

上一步里建立的是一个空的类。跟着我来完成它。

加入一个 Name

我建议你为你创建的每一个 feature 起一个名字。
可以像这样。

```
private const string Name = "CsbGoodies";
```

甚至一个没有使用的注释也不错。

```
//private const string Name = "CsbGoodies";
```

因为将来或许这些名字会在注册 Plus 的时候被用到。

3.4.3 注册一个菜单项

C#Builder Goodies Plus 创建了一个名叫"View Doc"的 BDS 菜单项。

```
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();
}
```

首先，重载名叫 IDERegisterMenus 的函数，你的菜单项将在这里创建。

调试的几行用于调试工程，你不一定需要这样的几行。

开头的时候调用基类的同名函数(尽管当前是空的)以防止我今后在里面添加新的内容。

RegisterMenu 函数调用的目的是为你注册一个菜单。把一个菜单对象作为参数传给它。

一个菜单对象可以通过调用下面的一些函数来创建(共有四类)。

CreateActionMenu 返回的是一个带有回调函数/事件响应函数的菜单。

其他的类型包括 EmytyMenu，SeparatorMenu，和 PlaceHolderMenu。

一个 EmptyMenu 的例子就是 LeXtudio 菜单项。它是菜单树的第一项。单击它不会触发任何事件响应。

一个 SeparatorMenu 则会被转化为菜单中的一个分隔符。

一个 PlaceHolderMenu 不会被转化为一个实际的菜单项。它们仅仅是作为占位符。菜单树的根节点就是一个不错的例子。如果你巧妙的使用这个类型，你可以给 ToolsMenu 或者 FileMenu 直接添加一个子菜单(当然，这不是我所推荐的方式)。

这里，Doc Preview 菜单项通过调用 CreateActionMenu 函数创建的，作为 MenuLeXtudio 的子项。它的名字是 MenuViewDoc。它的快捷键是 Default-ViewDocShortcut，而菜单上的内容是 MenuTextViewDoc。当你在 BDS 中最终单击这个菜单项，DoViewDoc 将被启动。

由于 BDS OTA 的限制，菜单的名字，MenuViewDoc，不能够包含”点”(否则会有运行时错误)。同时快捷键的值需要依照 TShortcut 类型的方式计算。将来我会把 C#Builder Goodies Plus 的这个 keycode analyzer 特性加入以减少你的计算麻烦。

在 DoViewDoc 中，我封装了一些 C#Builder Goodies 1.1 的函数。首先呼叫 CSBuilderGoodies.QuickdocViewer 以得到它的 Singleton 对象。那个 Doc Preview Form 就会在 Activate 函数被呼叫时弹出。

3.4.4 在 Global Preferences Dialog 中注册一页

```
public override void IDERegisterTabs() {
base.IDERegisterTabs();
RegisterTab(CreateTabNode(TabAStyle));
RegisterTab(CreateTabNode(TabJcf));
}

private System.Windows.Forms.GroupBox groupBox5;
//...
public override void IDERegisterComponents( ) {

base.IDERegisterComponents();

this.txtJcfPath = new System.Windows.Forms.TextBox();
//...

TheOtaFramework.TabControl.Controls.Add(this.tbpAStyle);
TheOtaFramework.TabControl.Controls.Add(this.tbpJCF);
```

```
this.tbpAStyle.ResumeLayout(false);
//...
}
```

如果你在树结构里面添加了一个页的节点，那么你应该在 TabControl 里面加入相关的一页(不过现在我还没有实现一个100%可靠的机制)。

在左边的树结构中加入一个节点

如果你需要，你可以在全局的 FormPreferences 中加入一个页。

你可以参考 Code Beautifiers Plus 的源代码。这个 Plus 通过重载 IDERegisterTabs 函数加入了两个节点。步骤和使用 IDERegisterMenus 差不多。

创建一个新页

IDERegisterComponents 函数将被重载。一些必要的 WinForms 控件将被手工创建，并且加入到一个 TabPage 控件(容器)内。最后，这个容器将被加入到 TheOtaFramework 的 TabControl 里面。

3.4.5 不注册 Global Preferences Dialog 的方式

如果你知道 CnPack 和 GExperts，你或许会喜欢用一个独立的对话框来注册你的 feature 选项，而不是注册一个全局的 FormPreferences。这是一个不错的选择。

在你的 Plus 里面设计一个这样的对话框，然后在你的 feature 里面加入一个管理用的菜单项来激活这个对话框。这并不困难。

3.4.6 如何管理一个 Feature 的首选项

有很多方式来存储首选项。注册表和 INI 文件都是很古老的方式。我强烈建议你使用 XML 文件，这样既简单也很容易阅读。

方式一，序列化和反序列化

Code Beautifiers Plus 采用这个方式。这个 feature 叫做 OtaCodeBeautifiers 有一个可序列化的私有成员类 Preferences。在执行过程中，设置被加载和保存。你可以阅读源代码以获得细节，或者从 .NET SDK 文档中取得相关信息。

这个方法是从 Sharp Builder Tools 中学到的。

方式二，操作元素

ShortcutArrayList 类采用了这个方式。在加载和存储快捷键时，成员类 Shortcuts 首先在相关的 XML 文件中查找到一个正确的元素。然后操作它的 InnerText 属性。你可以阅读源代码以获得细节，或者从 .NET SDK 文档中取得相关信息。

这个方法是从 C#Builder Goodies 1.1 中学到的。

3.4.7 更多

记住这是一个 SDK for .NET OTA，任何 .NET 语言都可以用来开发，甚至可以混合多种语言。例如，AddMany 4.1 是由 Delphi for .NET 开发的，而我做的封装是纯粹的 C#.

SDK 本身可以通过加入更多的类来实现扩展。加入的类将会作为 Minus 的一部分(当然这减少了 OTA 开发的复杂度)。

你甚至可以 LeXtudio Wrapping Technology 使用扩展或者重新实现 OtaFramework 和框架中的 Lextm.CodeBeautifierCollection.Main。

等多的函数可以被重载，以扩展你的 feature 列表。

你可以扩展这个 SDK 以获得更加强大的 feature。

如果你由于 GPL 的限制不打算使用这个 SDK，那么直接使用 SBT 的源代码或者依赖 Borland OTA API 函数都是不错的选择。我之所以选择 GPL 就是为了让这个 SDK 永远开源。

这个 SDK 现在仍然在它的初级阶段。我写的更多有意思的 feature 或者库还不是十分的成熟，所以没有加入。所以，继续等待，或者加入进来。如果你创作了一个 Minus 或者 Plus，把它发给我(有无源代码并不重要)。我可以在 CBC 的主页上面发布相关的消息，让别人了解你的工作。

让我们建立一个我们的社群好了。

3.5 参考资料

- Borland 帮助"Extending the IDE"
- GExperts 的源代码
- Sharp Builder Tools 的源代码
- Code Beautifier Collection 2 的源代码
- Google 或者 MSDN ...

Chapter 4

Plus 注册文件格式

4.1 版本 I 说明

4.1.1 问什么版本 I 十分简单

实际上才开始设计这个架构的时候我对于文件格式没有什么了解。所以我通过一些简单的实验来熟悉这些东西。一个简单的版本就可以提供 Plus 所需要的基本的设置能力。作为结果，这个版本的说明如下。

今后的版本应该会复杂很多，不过依然很容易设置。

4.1.2 文件后缀

.plus 就是文件的后缀。

4.1.3 如何生成

这是一个文本文件。

文件名需要和对应的 Plus 文件一致。

文件中的每一行代表一个 feature。

一行需要有两个分号分隔的部分组成。

第一个部分是 feature 类的名字。

第二个部分是一个标志位。设为 true 时启用这个 feature，否则不启用。

> 例如，这是我给 Code Beautifiers Plus 写的注册文件。
>
> 文件名应该是 Lextm.CodeBeautifiers.Plus.plus。
>
> 在文件中仅有一行：
>
> Lextm.CodeBeautifierCollection.Ota.OtaCodeBeautifiers;true

4.2 版本 II 说明，草稿

4.2.1 版本 I 中没有实现的

版本 I 实现了基本的功能。但是，还不能很容易的通过一个 GUI 界面的工具来设置，因为这个版本还缺乏不少的灵活性。所以这一个版本包含更多内容。

有时候在 BDS 1 中启用一个 feature 而在 BDS 2 中禁用它会是十分合适的。所以本版 II 考虑了这一需求。

同时，应该还有一些和当前 personality 相关的启用/禁用设置。这样的需求将会在这一版本或者今后的版本中实现。

为了让这个版本更加强大，简单的文本文件或许不合适。XML 格式或许是不错的选择。

我很乐意与你们讨论。

4.2.2 文件后缀

.plus2 就是这个版本的文件后缀。

4.2.3 如何生成

暂缺