# iCode 开发者指南

liuzikai@163.com

## iCode 开发概述

iCode是我个人开发的一个VB6外接程序（AddIn），自己断断续续也做了三年多了吧，从一开始的无从下手，到极不稳定的半成品，再到现在基本完善（当然可能还是存在一些Bug）。现在将其稍作整理进行开源。

VB6 Addin开发相关文档并不多，大部分只能依靠自己摸索+搜索。比较常用的是VB6帮助文件、百度、MSDN等。

以下会简单介绍iCode的一些开发思路、注意事项，以及一些Addin相关的信息，希望能有参考作用，如有不当之处，请见谅。一些用户说明文档中的内容（尤其是※号内容），在此就不再赘述。欢迎与我交流。

## 外接程序（AddIn）

VB6外接程序是借用VB提供的接口在IDE上实现特定功能的插件，可能会加上其他一些技术（如Hook、内存修改、汇编等）。

工程主体是ActiveX DLL，调用了特定的部件（Microsoft Add-In Designer等），这些部件将提供有限的接口用于控制IDE行为。工程中还会有一个名为Connect的设计器，用于连接（connect）。

外接程序与IDE连接，触发OnConnection事件，获得一个实例（Instance），每次启动IDE都会有一个实例。外接程序反加载时（VB关闭或外接程序被卸载），会触发OnDisconnection事件。Connect部件还有其他事件，再此就不一一介绍了。

## AddIn 接口

在VBIDE库中，包含了主要的外接程序接口，包含了窗口、代码、文件等一系列的控制接口。在OnConnect时，会获得一个VBE类的实例，这个类中包含了大部分的属性、方法、事件。另外还会获得一个AddIn实例，也会有用。

另外，在Office库中，包含了大部分CommandBar接口，用于控制标签栏、菜单等。

接口就不在此一一赘述了，可以通过对象浏览器查看，以及参考[美]Gene Swartzfager所著《Visual Basic开发人员指南》，里面有极其详细的接口说明（这本书我也是后来才发现，相见恨晚啊，不过是九几年出版的了，现在只能找到扫描版的，而且也找不到原书配套光碟中的实例程序了），以下是我开发过程中的一些经验分享。

* 这个库中有不少隐藏成员，如Window类的hWnd属性
* Event2类（隐藏）比Event类多一个VBBuildEvents，但通过VBE只能得到Event类。其实，只需要再声明一个Event2类，用Set直接赋值，虽然类不同，却是有效的。其他同理
* ActiveCodePane、ActiveVBProject、ActiveWindow、SelectedVBComponent这几个属性是很常用的，但这几个属性可能为Nothing，使用其中的成员时可能需要预判。
* VB弹出菜单也是有名字的，可以通过CommandBars类中的方法遍历找到，iCode\Document\CommandBars.txt是某次遍历扫描的结果，供参考
* CommandButton事件需要声明CommandBarEvents类，并通过VBE.Events获取，直接赋值WithEvents的CommandButton会出错

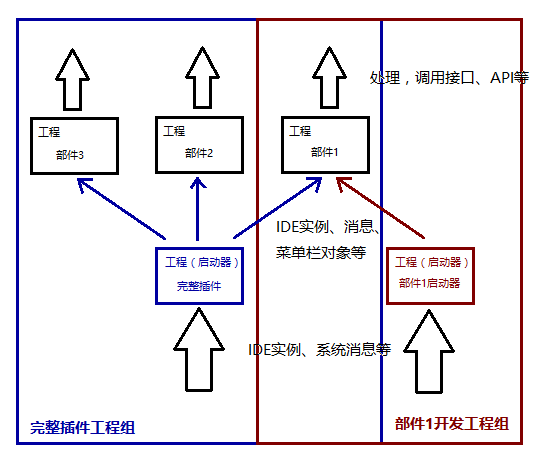
## iCode 架构

iCode的架构修改过多次，现在采用的是各个部件工程独立，启动器（含Connect）独立另一个工程，启动器工程（称为A）引用部件工程（称为B），通过调用类模块的接口实现功能。主要代码都在B中编写，A主要做提供IDE实例、提供Hook信息等作用。

这样做的好处是方便移植。iCode由许多组件组成，如果全都在一个工程中编写，会比较混乱，同时不方便调试。要知道，由于用上了Hook等，iCode时常要编译以后再测试，编译其他不必要的组件会浪费许多时间。采用这样的架构，先编写一个部件工程+一个启动器工程，测试完成后，iCode真正的启动器再引用部件工程，只需添加少量的调用代码即可，如果部件出现问题，也可以通过原启动器调试。

另外，一些公共的方法（主要在iFunction中）通过使用同一个代码文件保持同步。

下面是随手画的一张图：



## 调试与编译

iCode的调试并不容易。主要有IDE内调试和编译后调试。

不得不赞叹VB强大的IDE调试功能（尤其是用过Dev C++的调试后）。IDE内运行外接程序工程，类似于编译了DLL并注册，再启动一个VB，外接程序就随之启动了（VB正在跨线程调试！），这种情况还是相对比较方便的。但注意，使用iCode这种双工程结构后，ActiveWindow等有更大的可能会出现“连接丢失”，就是说获取不到，但编译后是没有问题的，这种问题也是比较头疼的。同时，IDE调试不能进行Hook（跨线程Hook需要调用者自身为DLL，Hook时会被注入目标线程，这也是许多木马病毒的入侵方法）。

PS：ActiveX DLL的工程属性-调试，选择“启动程序”，设置为VB6.exe，可使运行后自动启动VB，省去点按。

另外一种就是编译调试，这是最接近真实状况的调试，出现各种“迷之错误”的概率也会比较小，但缺点就是没有错误提示，难以跟踪，同时操作比较繁琐。对此，我采用了以下几种措施：

* 设计了个简单的AddIn，在菜单栏上添加“编译”和“启动VB”两个按钮，减少点按次数，源码在iCode\Develop\Develop Tool中，对应VB6.exe的目录可能需要你自行修改。

※这个AddIn直接将按钮添加到了菜单栏末端，虽然已经设置为Temproary（意味着按钮由VB自动回收），但仍可能出现其他按钮迷之消失的问题，可通过“重新设置”找回，测试自己的AddIn时建议不要启动。

* 设计了DebugForm，就是一个总置顶的带有大文本框的窗口，用于显示调试信息，双击清空。另外，在iFunction模块中，声明了一个Public的DBPrint方法，方便调用。DebugForm上还可以放置一些按钮，用于测试，由于每个工程都不一样，所以每个工程都配备了一个DebugForm（虽然大同小异）

※一般在Connect的OnConnect中有段条件编译语句，这是用于控制是否显示DebugForm的。

* 设计了远程Hook。通过C++编写了标准DLL，VB调用（iRemoteHook类），DLL会将自身注入到目标线程，捕获到目标消息（Windows Message）时，通过SendMessage发送WM\_COPYDATA到一个VB创建的不可见的窗口（iMiniWindow类），再从本地拦截这条消息。具体使用方法请参考iCode工程，DLL源码亦有附带。

※在我这里测试时，每次打开iCode工程后，只能运行一次，第二次iCode工程这边的IDE就会崩溃，可能跟汇编有关，原因尚不明确。

PS：VB编译DLL后会自动注册，但在高版本Windows下需要以管理员身份启动VB。注册DLL后，AddIn才会出现在“外接程序管理器”中。

接下来，是各个部件的一点介绍与备忘，供参考。

## iCode 主启动器

iCode\iCode\iCode.vbg

主启动器有调用部件、传递消息、提供设置，以及创建公共菜单、按钮等功能。

* 设置文件存放于AppData\iCode\Settings.ini中，内容以参数形式传递给各部件
* 公用的菜单、工具栏由主启动器创建，以参数形式传递给各部件
* Hook相关代码在启动器工程中编写，通过调用接口传递给各部件。由于VB调用函数时间花销很大，因此在CallBack过程中会进行部分的信息筛选。
* 由于工程放置位置与其他部件不同，而iRemoteHook引用DLL中用的是相对地址，且iRemoteHook是代码共享的，所以在开头设置了条件编译，以便切换。

## AutoComplete – API工具

iCode\Components\AutoComplete.vbg

AutoComplete起初是一个模仿VB自动完成的组件，但由于嵌入式设计极其复杂，又感觉实用性不高，因此改成了工具窗口。如果需要之前版本的文件可以与我联系。

* 连接到IDE中的工具窗口不是普通的Form，而是UserDocument，通过CreateToolWindow方法创建，需要一个GUID，申请一个GUID可以一直用。具体的创建方法请参考代码。
* clsAutoComplete是此工程对外的接口，主要代码则编写在udAPI中。
* 采用将API声明全部读取的方式，由于数目比较巨大，设置了启动动画，并且在加入TreeView时隐藏了Treeview，以加快速度。
* 由于反加载TreeView也会卡顿，原本设置了关闭动画，但是如果随VB一起关闭则没有这个问题，故将关闭动画变为设置中的一个选项，后来又去除了，但保留了相关代码。
* 为了避免重复加载，设置了一个TreeView的控件数组，如果选择了已加载的库，直接调出对应的TreeView（我称为Static TreeView,STV）

## CodeIndent – 代码缩进整理

iCode\Components\CodeIndent.vbg

代码缩进整理工具，主要代码集中在clsCodeIndent，有较为详细的介绍，就不做过多的解释了。

CodeOperator，是我编写的用于控制代码窗口的类，稍作说明：

* SL、SC、EL、EC分别是StartLine（光标或选择的起始行序数）、StartColumn（起始字符）、EndLine（结束行序数）、EndColumn（结束字符数）的缩写。
* CodePane.GetSelection获得的SC、EC，下标是从1开始的！也就是说，若光标在第一个字符前，则SC和EC为1，这简直反人类啊~对于字符串处理也不方便，因此CodeOperator封装的SC、EC均是-1过的，如果要按照习惯序号设置光标，也要通过封装的SetCaret而不是原有的CodePane.SetSelection。
* 对于是否选中最后的换行符，CodeOperator也有处理，详见注释。

## ColorCode – 代码色彩修改器

iCode\Components\ColorCode.vbg

代码色彩修改器，通过读写内存实现，内存地址是通过CheatEngine和OllyDBG找到的，就是有些缺陷就是了。

## IDEEnhancer – IDE工具

iCode\Components\IDEEnhancer.vbg

IDE工具。

## CodeStatistic – 代码统计

iCode\Components\CodeStatistic.vbg

代码统计工具。

* 工程名称是“iCode\_Statistic”，少了个“Code”，这主要是因为当初有个类模块名字也很长，然后超出了限制，所以就改了一下→\_→，现在那个模块也不用了。

## TipsBar – 工作区标签栏

iCode\Components\TipsBar.vbg

工作区标签栏。一开始我做外接程序主要就是奔这个来的，曾用过CodeSmart、AmicForVB，很好奇它们是怎么实现的，一开始也无从下手。摸索了很久，最后终于找到了解决方法（虽然好像不太一样？为什么同时用时我的标签栏总是在它们的上边→\_→）。

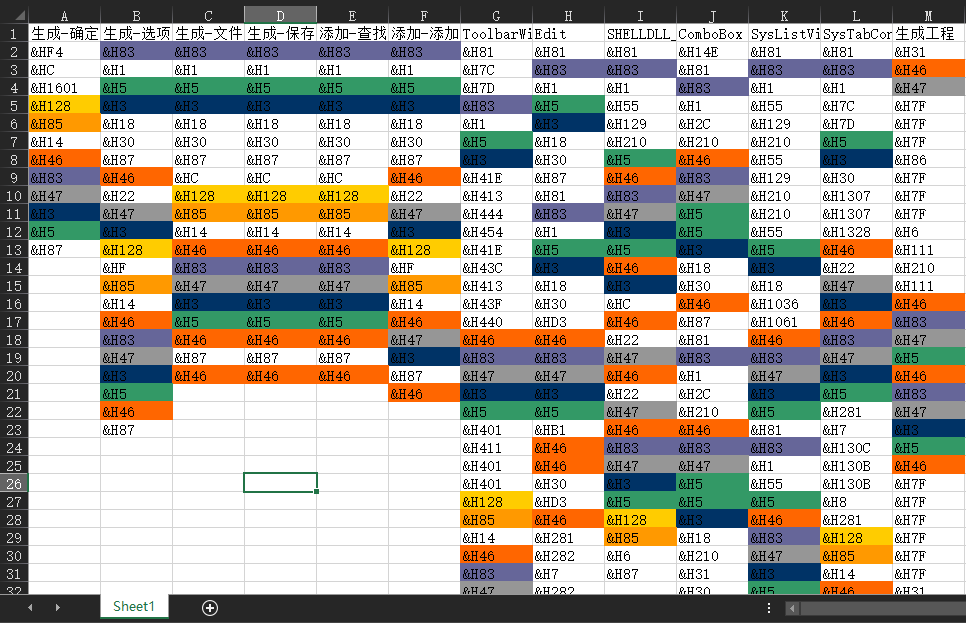
* 嵌入原理：工作区是一个类名为MDIChild的窗口（名称好像不大对？），通过Hook拦截WM\_SIZE信息，在它改变大小时，稍作调整，在其上方移出一小块空间，将TipsBar控件通过SetParent移动到VBIDE中。为什么要用Hook呢？每次最大化、最小化、移动VB窗口，改变工具栏布局时，都会引起MDIChild调整大小，如果只在启动时嵌入一次，不一会标签栏就会和工作区上端重叠了。用Timer则会有较大延迟，故选择了Hook。
* 拦截窗口信息：工作区本质上是一个MDI父窗口，代码窗口等是其子窗口，子窗口动作会向父窗口发送一系列MDI信息，拦截这些信息，分析窗口类名、标题即可。尝试过WM\_SETFOCUS之类的信息，最后选用了MDI系列的信息，精准，不容易错漏。
* 为什么要将TipsBar设计成控件而不直接设计为窗体？尝试过，但是对窗体使用SetParent会出现乱码，以及代码内的连接（方法调用之类的）失效，而使用控件则没有这个问题。以下Linker的按钮也是同理。

## Windows+ Linker – 文件窗口重布局+编译强化

iCode\Components\Windows+Linker.vbg

这两个部件是放在同一个工程的，主体是Windows，原因是Linker设置按钮需要借助Windows的界面控制。Windows是一个技术含量还蛮高的部件，对控件产生的消息进行拦截，并匹配类名、标题，对控件大小、位置进行调整，以实现文件窗口重布局。设计这个部件最初的触发是因为一开始我的所有工程文件堆在一个文件夹，而文件窗口又太小，很麻烦。

* 挑选合适和消息来匹配控件是这个部件的核心所在，消息太早，控件没有加载完，MoveWindow无效；消息太晚，有延迟；消息太多，重复操作，造成卡顿；消息补全，有些控件不能移位，更糟糕。我前后尝试了不下几十种消息，最终配合Excel处理找到了&H128这个极佳的消息，实现了这个功能。放张图片：



这些数据是用DeBugForm打表而成的。颜色是我用VBA编写的宏自动匹配的各列的共有部分。&H128各列均有，且只出现一次，经测试此时使用MoveWindow可行。

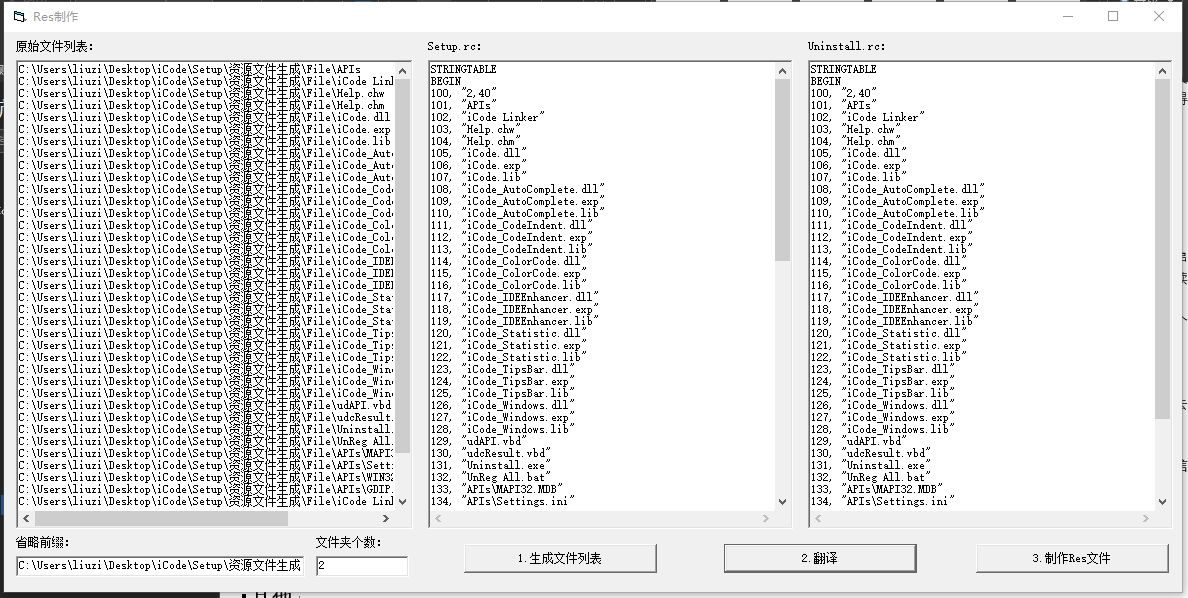
* 现有布局是我用Spy++逐个逐个控件调整再记录下来的，已固化在代码里。
* TaskCollection是用于匹配控件的，同时减少代码编写量。
* Linker的部分代码在clsWindowsHandler中
* Linker的原理是VB在编译结束后触发FileControlEvents\_AfterWriteFile，通过判断后运行iCodeLinker.exe，再通过UpdateResource等API替换EXE文件资源，以达到替换图标等功能。iCodeLinker.exe的工程文件在iCode\Components\Windows+Linker\iCodeLinker，其中提取图标组的代码比较值得参考，网上提取图标的代码大部分是不全的，只能提取到部分图标。

## 安装与卸载程序

iCode的安装程序是自制的单文件安装程序，主要通过资源文件实现。Setup.res文件中包含一个字符串表，一系列用户自定义资源（即文件）。字符串表中，101项为“文件夹个数,文件个数”，接下来分别是以安装目录为根目录的文件夹、文件名称，各文件对应相同序号的用户自定义资源。安装程序先读取101项，然后依据字符串表建立文件夹，然后根据字符串表释放文件，并注册DLL文件，最后读写注册表。

iCode的卸载程序也是自制的程序，其资源文件中含有一个字串表，与安装程序是一样的，运行时根据字符串表删除文件，并在temp目录下建立一个bat文件，用于删除安装程序以及其自身，最后读写注册表。

资源文件生成由Res Maker.exe辅助生成。



1. 首先将所有要安装到安装目录下的文件放置于iCode\Setup\资源文件生成\File，运行Res Maker.exe，
2. 点击“1.生成文件列表”，程序将文件列表自动生成于第一个文本框中（通过dir命令实现，先写入了File List.txt，再读入文件）。在其下方，需要填写去除的前缀（…iCode\Setup\资源文件生成\File\），以及标明文件夹个数（需要手动，文件夹会放置在前几项）。
3. 点击“2.翻译”，程序自动生成rc文件内容（用于生成Res文件）。
4. 点击“3.制作Res文件”，程序将rc文件写入到File文件夹（似乎rc文件必须与目标文件在同一个目录下才能制作成功），运行RC.exe，根据rc文件信息翻译Res文件到“资源文件生成”目录下，为Setup.res、Uninstall.res，弹出的两个控制台窗口需要手动关闭。如果出错，将不能生成res文件。
5. 弹出确定对话框，点击确定后，rc文件将被清除，以免影响下一次生成。
6. Uninstall.exe最初需要编译一次，保证文件列表中出现。第一次制作完Res文件后，Uninstall.res被加载，重新编译Uninstall.exe（由于卸载程序是按照文件名卸载的，如果文件名不变，可跳过这一步骤）。

之后，需要再次生成Res文件，此时的Setup.res才是真正可用的（包含编译过的Uninstall.exe），再装载它编译生成Setup.exe。

1. 1.2版本发布后，部分小伙伴反馈出现“未预期错误”，可能是使用了真彩色图标导致的，故将安装、卸载程序主窗口的图标更改为SINEWAVE.ICO，编译后使用ResHacker替换图标，同时去除了窗口阴影以提高稳定性。

## 目录结构说明

* Components目录下是各个组件的工程文件。
* Develop目录下是各种开发所用文档、工程。
* Dlls目录下是编译后的文件。
* Documents目录下是一些相关文档和资源。
* iCode目录下是iCode主启动器的工程文件
* Public目录下是各个组件公用的工程文件。
* Setup目录下是安装、卸载程序、帮助文件的工程、制作工具、成品等。
* CleanUp.bat用于清理各种tmp、log文件，以及VB崩溃可能在Components文件夹下留下的“工程1.vbp”等文件
* iCode.vbg是iCode主体工程文件。
* iCode\Documents\Reference中有一些参考资料，包括最初一版的iCode（最初架构），如果需要历史版本也可以联系我

## 其他

以上就是iCode以及外接程序的一些信息，希望对你有所帮助。本文没有很系统地叙述外接程序，更多的内容隐藏在iCode源码之中。由于iCode开发时间跨度大，我自己的代码风格也有一些变化，包括一些命名方式、排版之类的，部分组件也没有注释。如果你也要开发VB6 AddIn，如前文所说，Gene Swartzfager所著《Visual Basic开发人员指南》是一本很好的参考资料，同时，你可能会用到以下的工具：

1. Microsoft Spy++。这一工具能够获取、调整窗口、控件等，以及监测系统消息，对于系统层次的开发是很有帮助的。
2. Resource Hacker。可以用它提取图标，对资源文件作出修改等。
3. OllyDbg。强大的反汇编工具。
4. ProcessMonitor。用于监控程序读写文件、注册表等行为。
5. Depends和DLL Export Viewer。用于分析可执行文件等。

各大搜索引擎、MSDN都是很好的资料来源，有时搜集资料也需要研读其他语言，不仅仅是VB。

那么，祝你好运！