CnWizards专家编写指南

作者：周劲羽（[zjy@cnpack.org](mailto:zjy@cnpack.org)）

刘啸（[liuxiao@cnpack.org](mailto:liuxiao@cnpack.org)）

发布：[http所://www.cnpack.org](http://www.cnpack.org/)

版本：V0.1.0.1（大纲）

创建：2004-06-18

更新：2007-06-12

目 录

[一、 概 述 3](#_Toc144902871)

[1、 前言 3](#_Toc144902872)

[2、 适用读者 3](#_Toc144902873)

[3、 术语列表 3](#_Toc144902874)

[二、 CnWizards框架简介 5](#_Toc144902875)

[1、 IDE扩展与CnWizards 5](#_Toc144902876)

[2、 CnWizards框架概述 5](#_Toc144902877)

[三、 编写专家插件 5](#_Toc144902878)

[1、 编写专家插件类 5](#_Toc144902879)

[2、 定义Action和菜单项 6](#_Toc144902880)

[3、 使用框架提供的服务 8](#_Toc144902881)

[4、 实现专家插件功能 8](#_Toc144902882)

[5、 定义配置界面 9](#_Toc144902883)

[6、 进行多语言处理 9](#_Toc144902884)

[7、 使用图标与注册专家插件 10](#_Toc144902885)

[8、 调试专家插件 10](#_Toc144902886)

[四、 CnWizards框架详解 11](#_Toc144902887)

[1、 一个最小化的专家框架 11](#_Toc144902888)

[2、 ToolsAPI封装及扩展 11](#_Toc144902889)

[3、 快捷键管理器 11](#_Toc144902890)

[4、 Action和菜单项管理器 11](#_Toc144902891)

[5、 专家插件管理器 11](#_Toc144902892)

[6、 专家设置管理器 11](#_Toc144902893)

[7、 多语言处理机制 11](#_Toc144902894)

[8、 事件通知服务 11](#_Toc144902895)

[9、 IDE挂接服务 12](#_Toc144902896)

[10、 IDE停靠窗体 12](#_Toc144902897)

[11、 属性及组件编辑器系统 12](#_Toc144902898)

[12、 仓库专家系统 12](#_Toc144902899)

[13、 与WEB服务器的交互 12](#_Toc144902900)

[五、 深入研究IDE 12](#_Toc144902901)

[1、 查看IDE内部控件 12](#_Toc144902902)

[2、 分析IDE设计期包 12](#_Toc144902903)

[3、 挂接IDE的控件和方法 12](#_Toc144902904)

[六、 附 录 13](#_Toc144902905)

[1、 专家框架类图 13](#_Toc144902906)

## 概 述

### 前言

CnWizards是由CnPack开发组开发并维护的Delphi/C++Builder/BDS的IDE扩展工具，全称为CnPack IDE 专家包，英文全称为CnPack IDE Wizards，后文简称CnWizards。本文档较全面地介绍了CnWizards的框架结构及编写专家插件的方法，用于指导CnPack开发人员编写专家插件，也可供对ToolsAPI感兴趣的朋友参考。同时，该文档也部分描述了CnWizards的现有框架的设计，小部分内容可能未在现有框架中实现，文中会有说明。

文档按由浅入深的顺序分成以下六部分：

第一部分 介绍文档的基本内容及术语列表等。

第二部分 简述CnWizards框架的基本结构。

第三部分 以一个专家插件实例说明编写专家插件的方法。

第四部分 详细介绍CnWizards框架的方方面面。

第五部分 介绍更深入的IDE扩展方法。

第六部分 附录，包含一些图表等。

### 适用读者

编写IDE专家扩展相对是复杂的，建议阅读本文的读者应具备以下能力：

* 理解Delphi面向对象开发的思想，有较丰富的开发经验。
* 编写过组件和DLL。
* 熟悉接口及相关知识。
* 对ToolsAPI有一定的了解，并编写过简单的专家。

如果你还不熟悉ToolsAPI，请参考CnPack开发组翻译的官方帮助《扩展Delphi的IDE》。

### 术语列表

以下是本文档使用的术语列表：

* 插件（Add-in），以设计期包或DLL形式被设计期的IDE调用的扩展工具。
* 专家（Wizard），实现了IOTAWizard接口的IDE插件工具。
* 专家插件，本文指CnWizards中由专家框架统一管理的专家插件。
* 仓库专家（Repository Wizard），用来创建新的单元、窗体或工程的专家。
* 包（Package），Delphi中使用的特殊的动态链接库。
* 设计期包（Design-time Package），被编译为允许IDE在设计期装载的包。
* 运行期包（Runtime Package），被编译为允许DLL在运行期调用的包。
* 接口（Interface），Delphi中使用的COM风格的接口。
* 通知器（Notifier），由用户实现特定接口并由IDE在特定事件中调用的用户对象。
* 创建器（Creator），由用户实现特定接口的用于创建新的单元、窗体或工程的用户对象。
* 工程（Project），Delphi中的Project。
* 单元（Unit），Delphi中的Unit。
* 模块（Module），对应着一组在IDE中打开的逻辑上关联的文件集，可以是一个单元、包含窗体的单元、工程文件等对象。
* 编辑器（Editor），IDE中用来设计和编辑模块的对象。

## CnWizards框架简介

### IDE扩展与CnWizards

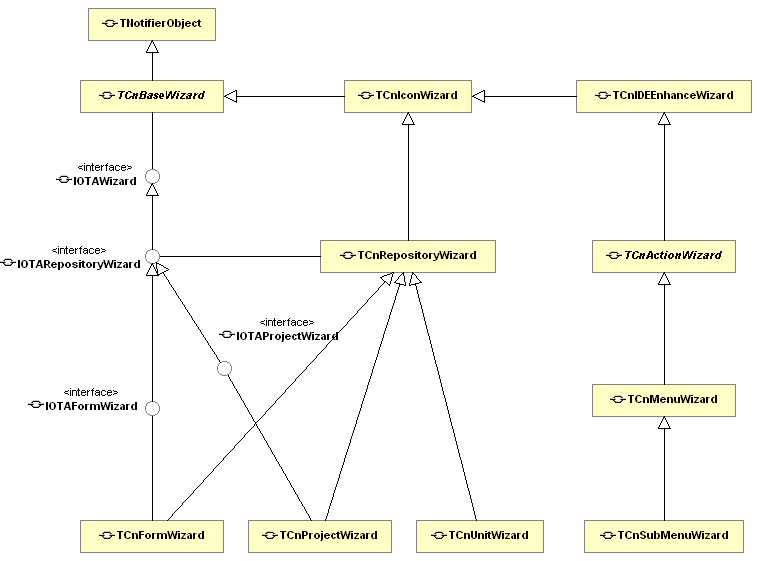
这里介绍一下IDE扩展的基本概念及CnWizards的工作原理。通过使用Open Tools API（通常缩写为Tools API或OTA），开发者可以在IDE中使用自定义的菜单项、工具栏按钮、动态的窗体等东西来扩展和定制IDE，这种IDE扩展模块通常称为专家。CnWizards是一堆专家的集合。

### CnWizards框架概述

CnWizards框架指不涉及具体专家功能，但为所有专家服务，实现公共功能的部分。

框架要完成在IDE主菜单中插入CnPack菜单、配置和管理快捷键、专家子菜单、提供专家设置界面、实现关于窗口，以及定义专家基础类等功能。同时，框架还要实现专家管理器，负责专家列表的维护、注册、释放等操作。

在CnWizards中，专家分为两大体系：基于菜单、快捷键或在后台运行的专家，以及用来创建新单元、窗体、工程的资源模板向导。其类继承关系如图：



具体内容可参考《CnWizards概要设计说明书》和《CnWizards框架详细设计说明书》。

## 编写专家插件

### 编写专家插件类

#### 专家框架提供的插件基类介绍

CnWizards提供了以下几种专家基类供专家开发者继承使用：

* TCnMenuWizard：带菜单项的 CnWizard 专家抽象基类，继承自此类的子类专家对应着在CnPack菜单下的一菜单项，可带图标，点击它可执行一功能。
* TCnSubMenuWizard：带子菜单项的 CnWizard 专家抽象基类，继承自此类的子类专家对应着在CnPack菜单下的一带子菜单的菜单项，该菜单项本身不带图标，各个子菜单可带图标，点击子菜单执行不同的功能。
* TCnIDEEnhanceWizard：普通的IDE功能增强专家的抽象基类，不包含现成的菜单项，常用于从其他地方调用的场合，如编辑器扩展专家等。
* TCnProjectWizard：Repository工程向导基类，用来在Repository中生成一个工程。
* TCnUnitWizard：Unit向导基类，用来在Repository中生成一个单元。
* TCnFormWizard：Form向导基类，用来在Repository中生成带窗体的单元。

#### 怎样开始编写一个新的专家插件

要编写一个新的专家插件，首先要从功能需求等方面确定新专家应该继承自哪个基类。如果是需要在菜单中调用，则可以考虑继承自TCnMenuWizard或TCnSubMenuWizard。关于TCnMenuWizard或TCnSubMenuWizard的子类的例子可参考源码中cnwizards\Source\Examples中的CnSampleMenuWizard.pas和CnSampleSubMenuWizard.pas的内容。

对于其他类型的功能，可以考虑继承自TCnIDEEnhanceWizard，普通IDE功能增强专家。这种专家没有菜单项Action等处理，只有一图标属性供专家设置窗口使用。

如果要实现一个什么都不做的专家，可从TCnIDEEnhanceWizard继承一子类并实现其GetWizardInfo方法来提供专家的说明和版权信息即可；如果此专家需要显示在CnPack的菜单项中，则需要改为继承自TCnMenuWizard并实现GetCaption等方法以返回菜单项名称和快捷键等；如果菜单项需要图标，则需要创建一和专家类名同名的ICO文件放置于Icons目录下，专家框架会自动读取它并用来显示在菜单项和专家设置窗口中。

#### 实现专家插件类的基本规则并注册

确定好基类后，就可以着手实现一专家插件类，此专家类名应该取为TCnXXXXWizard等类似的直观名字。实现此类的源文件也应该按照CnPack的命名规则命名为CnXXXXWizard.pas并放置于cnwizards\Source的相应目录中。文件头部应该包含CnPack的协议说明，单元名后也应该包含单元头，interface后应该通过{$I CnWizards.inc}来引入基础的条件定义，然后initialization部分再通过调用RegisterCnWizard(TCnXXXXWizard)来注册本专家。注册这一步是很重要的，否则即使包含此专家类的单元编译进去了该专家类也得不到创建的机会，更别说发挥其功能了。

单元头格式内容等，可参考 cnwizards\Source\Examples中的CnSampleMenuWizard.pas

### 定义Action和菜单项

专家框架提供了统一的快捷键、Action和菜单项管理，继承自TCnMenuWizard的子类只需要重载实现以下几个重要方法即可：

  public   
    function GetState: TWizardState; override;   
 {\* 返回专家状态，IOTAWizard 方法，子类重载该方法返回专家状态 }  
    procedure Config; override;   
 {\* 专家配置方法，由专家管理器在专家配置界面中调用，当 HasConfig 为真时有效 }  
    procedure LoadSettings(Ini: TCustomIniFile); override;   
 {\* 装载专家设置方法，子类重载此方法从 INI 对象中读取专家参数 }  
    procedure SaveSettings(Ini: TCustomIniFile); override;   
 {\* 保存专家设置方法，子类重载此方法将专家参数保存到 INI 对象中 }  
    class procedure GetWizardInfo(var Name, Author, Email, Comment: string);  
  override;   
 {\* 取专家信息，用于提供专家的说明和版权信息。抽象方法，子类必须实现。

var AName: string - 专家名称，可以是支持本地化的字符串

var Author: string - 专家作者，如果有多个作者，用分号分隔

var Email: string - 专家作者邮箱，如果有多个作者，用分号分隔

var Comment: string - 专家说明，可以是支持本地化带换行符的字符串 }  
    function GetCaption: string; override;   
 {\* 返回专家的标题 }  
    function GetHint: string; override;   
 {\* 返回专家的Hint提示 }  
    function GetDefShortCut: TShortCut; override;   
 {\* 返回专家的默认快捷键，实际使用时专家的快捷键会可能由管理器来设定，这里

只需要返回默认的就行了。 }  
    procedure Execute; override;   
 {\* 专家执行体方法，IOTAWizard 抽象方法，子类必须实现。   
       当用户执行一个专家时，将调用该方法。 }  
  end;

一个TCnMenuWizard的子类内部拥有一个Action关联到此菜单项上。该Action的Caption、Hint、ShortCut以及执行方法等都从子类实现的上述方法中取得。

如果是子菜单专家，则需要继承自TCnSubMenuWizard，重载其AcquireSubActions方法，在其实现中通过调用RegisterASubAction来注册子菜单项的各个Actions的Caption、Hint、Shortcut和图标等，每注册一个Action子菜单项，便会返回一个此菜单项的索引号供专家内部保存，当子菜单项被点击或者需要更新（Action的Update）时，会调用专家的SubActionExecute或SubActionUpdate方法，将被点击（或需要更新）的子菜单Action的索引号传入。专家需要自行重载SubActionExecute和SubActionUpdate，根据传入的索引来处理对应的Action或执行对应的功能。

procedure AcquireSubActions; virtual;   
    {\* 子类重载此过程，内部调用 RegisterASubAction 创建子菜单项。   
        此过程在多语切换时会被重复调用。 }

procedure SubActionExecute(Index: Integer); virtual;   
    {\* 子菜单项执行方法，参数为子菜单的索引号，专家类重载该方法处理 Action 执行事件 }

procedure SubActionUpdate(Index: Integer); virtual;   
    {\* 子菜单项更新方法，参数为子菜单的索引号，专家类重载该方法更新 Action 状态 }

function RegisterASubAction(const ACommand, ACaption: string;   
      AShortCut: TShortCut = 0; const AHint: string = '';   
      const AIconName: string = ''): Integer;   
    {\* 注册一个子 Action，返回索引号。   
       ACommand: string         - Action 命令字，必须为一个唯一的字符串值   
       ACaption: string         - Action 的标题   
       AShortCut: TShortCut     - Action 的默认快捷键，实际使用的键值会从注册表中读取   
       AHint: string            - Action 的提示信息   
       Result: Integer          - 返回列表中的索引号 }

### 使用框架提供的服务

CnWizards框架提供的服务主要包括三方面：一是框架的专家基类提供的一些专家基本功能，二是封装的OTA的相关函数库，三是专家自行封装的各种通知事件服务。

* 专家框架中，所有专家由一专家管理器实例进行控制，此专家管理器TCnWizardMgr在专家DLL加载时创建，创建的同时负责创建专家、注册属性编辑器、处理IDE主菜单、初始化多语等诸多总体性的工作，同时专家管理器提供按类名、索引等访问专家实例列表中某个专家实例的服务（如果某窗体中需要访问某专家实例，则可通过此专家管理器获得），详细的属性方法等，可参考cnwizards\Source\Framework\CnWizManager.pas中的TCnWizardMgr声明部分。另外，专家基类等提供的LoadSettings/SaveSettings/LanguageChanged等方法在某需要的特定时刻被调用，子专家可重载它们实现自己需要的功能，从某种意义上来说这也算专家框架提供的服务之一。
* CnWizards对OTA的封装函数库大部分都在cnwizards\Source\Utils\CnWizUtils.pas，其中一半多来自GExperts的对应源码，提供了许多将复杂的OTA函数调用简化成一次调用的函数或过程，比如直接获得当前工程、工程组、单元、窗体编辑器、模块编辑器、打开文件名等，详情也请参考该文件内部的各个函数的声明部分，都有各自的注释。另外，一些和OTA无关的函数库的封装则处于cnpack\Source\Common\CnCommon.pas中。
* CnWizards也提供了对IDE的各种通知事件的封装服务，该服务对外是一ICnWizNotifierServices接口实例，通过全局函数CnWizNotifierServices获得，获得该接口实例后，便可通过调用其Add或Remove系列方法添加或删除所需要的通知方法。至于支持哪些事件通知，可参考cnwizards\Source\Utils\CnWizNotifier.pas的ICnWizNotifierServices的声明部分。

### 实现专家插件功能

实现专家的功能，首先要确定各个功能点需要在什么地方执行，如果是通过菜单、工具栏等被调用，则可写在专家的Execute或SubActionExecute方法中；如果是在某事件通知的时候被调用，如打开、关闭文件后得到通知，则可在专家的创建过程中注册被通知的方法，在自己的被通知方法实现部分写需要的代码。如果需要和IDE打交道，则需要熟悉OTA中的各个概念，再去查询ToolsApi源码或帮助，查询CnWizUtils等单元来看有无现成的实现函数库，或者阅读类似的CnWizards子专家的源码来获得帮助。

### 定义配置界面

专家框架提供了统一的专家配置管理，如果某专家重载的GetHasConfig函数返回True，则表示此专家存在配置窗口，可从专家的配置窗口中统一调用，调用时会调用专家的Config方法，子专家重载此方法来弹出一专家配置窗口并实现保存。

专家的配置信息一般存在专家实例的私有变量中，在Config方法被调用时，创建专家配置窗口，并将变量赋值给配置窗口界面，然后ShowModal它。当用户配置好点确定后，再从配置界面窗口中把值取出赋值给私有变量并进行配置更新的动作。另外在专家的LoadSettings方法中，可根据传入的Ini对象读入专家的配置信息赋值给私有变量，SaveSettings方法中则保存私有变量到Ini对象中。该Ini对象由CnWizards框架统一处理，目前此类配置都保存于注册表中。开发者也可在LoadSettings和SaveSettings方法中自定义其他不适合于在注册表中载入保存的内容。

### 进行多语言处理

如果CnWizardsd的子专家涉及了界面方面的东西，则必须支持多语言（目前是简体中文、繁体中文和英文）。多语的支持是由CnPack组件包中的多语组件实现的。它提供了一些多语服务给专家使用，同时专家中的字符串等也必须符合多语的处理要求。

专家里新增窗体时，窗体必须继承自CnWizTranslate单元中的TCnTranslateForm以达到能在运行期动态翻译的目的。然后再生成整个窗体的翻译字符串，添加入Lang目录下的多语文件中（专家包是CnWizards.txt，其他独立工具可有独立的语言文件名）。

生成翻译字符串的步骤如下：

* 在控件都放置好后，保存。注意以下步骤完成后，不要保存该窗体。
* 拖一个CnHashLangFileStorage到窗体上，设置其LanguagePath为一临时目录，用来存放临时生成的多语文件。
* 拖一CnLangManager到窗体上，把LangStorage属性设置成CnHashLangFileStorage1。
* 编辑CnHashLangFileStorage1的Languages属性，增加一种语言。因为需要的只是目标字符串，所以一种语言足够，而且LangID随便设置。
* 拖一CnTranslator到窗体上，双击。在弹出的窗体中左边选中该语言，点击生成按钮，右面会生成字符串，点击复制按钮以生成完整的带翻译值的字符串，点保存按钮。
* 在LanguagePath的临时目录中找到该语言文件，打开，复制，剩下的工作就是翻译后加入Lang目录下的多语文件中。

如果是新增多语字符串变量，则按下列三步走：

* 在CnWizConst单元里头加入这个字符串变量。
* 在WizTranslate翻译单元里头加上这个字符串变量的翻译赋值语句。
* 在各个语言文件里头加上新字符串的翻译条目。

GB（GBK）到BIG5码的转换目前采用李志成的ConvertZ 8.02的中文内码转换器，至于英文，则需要自己翻译了，千万别使用机器翻译，会惨不忍睹的。

### 使用图标与注册专家插件

专家插件的图标目前使用两种机制获得，一是根据专家插件的类名去CnWizards的资源DLL中查找图标，无则根据专家插件的类名查找专家安装目录中Icons目录下的同名ICO文件；开发期如需要替新增专家创建图标，可以创建一32×32的256色（或16位色，色彩超过256可能会有显示兼容性的问题）图标，以专家类名命名放置于Icons目录下即可，发布时此图标会自动被编译入资源DLL中。当专家中需要此专家的16×16的图标时，会按比例缩小32×32，不会取其图标文件中的16×16部分（即使它存在），这点请注意。

如果是其他地方需要使用工具栏图标等，可以将其放入CnWizShareImages单元中的Images这个ImageList中，注意请勿改变已经存在的图标。

注册专家插件上文已经提到过，需要在专家类的实现单元的initialization部分通过调用RegisterCnWizard(TCnXXXXWizard)来注册本专家。这一步注册实际上是把此专家类加入CnWizards的专家管理器框架中的专家类列表中，框架初始化时会根据此类列表逐个创建专家实例来加以管理，这也是上面提到的CnWizards框架提供的服务之一。

### 调试专家插件

专家的调试并不太容易，因为它最终是DLL的形式，所以需要启动一IDE来加载此DLL，也就是说专家的调试属于用IDE来调试IDE，方法稍微复杂一点（这里拿Delphi 5进行举例说明）：

* 首先，需要在Run菜单的Parameters子菜单中设置DLL的宿主程序，也就是Host Application设置为“C:\Program Files\Borland\Delphi5\Bin\delphi32.exe”
* 其次，为了让被运行的IDE能加载编译后的专家DLL进行源码级调试，也必须在注册表的HKEY\_CURRENT\_USER\Software\Borland\Delphi\5.0\Experts中添加一项加载此DLL的字串值CnWizards\_D5，值的内容是编译后输出的DLL全路径名。
* 再次，DELPHI调试器有个奇怪的规矩，那就是只有项目文件和编译后输出的DLL必须在同一个目录下才能进行源码级的调试，否则即使能通过Reload Symbol等方法下断点也可能出莫明其妙的错误（BDS 2006不存在此问题）。因此需要临时修改CnWizards\_D5的项目设置，将Directories/Conditional里的Output Directory清空，这样编译后的输出DLL就放到了dpr文件的所在目录（原来是在..\..\Bin目录下）。

如果在运行期担心其他专家的功能对待调试的功能有干扰，则可以禁用其它专家，禁用有两种，一种是在专家设置对话框中选中不需要的专家，不勾其使能选项；这样此专家的Active属性为False，许多功能便不起作用；更彻底的是在高级设置中让不需要的专家不创建，这样专家连创建的机会都没有，除了占用内存之外，几乎不会产生任何运行期影响。这是最彻底的禁用办法，需要IDE重新启动。

如果实在没法启动IDE进行正常IDE单步调试（比如IDE存在Bug），或IDE单步调试会影响某些正常功能的消息走向，则可以考虑使用日志型调试工具来进行日志记录，这也是CnWizards提供的服务之一，只要uses CnDebug单元，便可在代码中使用全局对象CnDebugger的各种方法来输出调试字符串，输出的调试字符串信息可以通过单独启动的CnDebugViewer来查看。

更多的详情，请参考《CnWizards调试说明书》与《CnDebugger接口帮助文档》。

## CnWizards框架详解

### 一个最小化的专家框架

一个最小化专家框架必须包括Framework目录下的单元：

CnWizBplEntry.pas

使用BPL方式时的专家入口单元。

CnWizClasses.pas

专家基础类实现单元。

CnWizCompilerConst.pas

编译器常量定义单元。

CnWizConsts.pas

专家字符串定义单元。

CnWizDllEntry.pas

使用DLL方式时的专家入口单元。

CnWizManager.pas

专家管理器单元。

CnWizMenuAction.pas

专家菜单以及Action管理单元。

CnWizOptions.pas

专家常用参数管理单元。

CnWizShortCut.pas

专家快捷键管理单元。

另外，多语部分虽然目前未从专家框架中独立出来，但实际上专家的最小框架是可不需要多语机制的。多语可作为一个外挂模块，挂接在专家框架上以提供多语翻译和切换功能。

### ToolsAPI封装及扩展

专家框架对ToolsAPI进行的封装扩展集中在CnWizUtils.pas和CnWizIdeUtils.pas中，提供ToolsApi和IDE部分对象的封装，其中部分代码移植自GExperts。开发者可通过阅读其内容和注释来了解其使用方法。

### 快捷键管理器

专家框架通过CnWizShortCut.pas封装了快捷键注册及管理功能。它主要被菜单专家和子菜单专家使用。一般菜单专家和子菜单专家中使用的快捷键都对应到了快捷键管理器中的每一个条目，它们由该快捷键管理器的实例负责向IDE注册、载入和保存快捷键等。

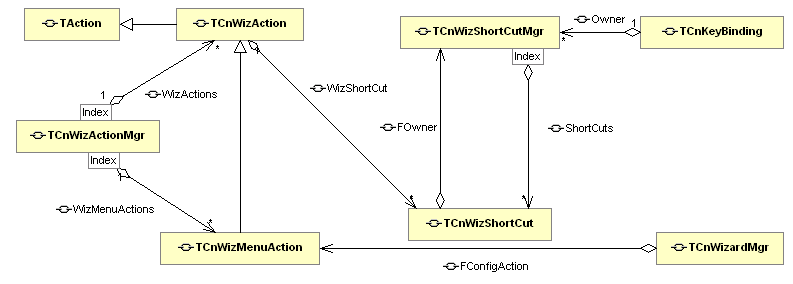
### Action和菜单项管理器

专家框架通过CnWizMenuAction.pas封装了Action及关联菜单项管理功能。它也是主要被菜单专家和子菜单专家使用。一般菜单专家和子菜单专家中所生成的菜单项都对应到了Action和菜单项管理器中的每一个条目。

### 专家插件管理器

专家的插件管理器是指CnWizManager.pas中的CnWizardMgr实例，它是真正在IDE中注册的专家，管理专家插件列表及提供专家公共服务。

以上三项管理器的关系如图：



### 专家设置管理器

框架目前未提供的专家统一设置管理功能，只在所有专家的设置界面中统一调用各个专家的Config虚拟方法，子专家可重载它的实现来弹出一设置窗口供用户设置内容。另外框架提供了统一的INI注册表接口来读写设置信息，开发者可不关心读写的具体位置。

在设置内容不适合于注册表中储存而必须保存在外部文件中的场合，开发者需要注意区分院原始的文件内容以及用户更改设置后的保存的文件内容。关于这部分，专家包提供了一部分简略的封装机制：初始化的数据文件都放在安装目录的Data目录下，而用户修改后保存的数据都放在安装目录的User目录下。专家在从文件读入设置时，应当先在User目录下搜索此文件，如存在则使用它；如不存在，则搜索Data目录下的同名文件，存在则也使用它。

这样设计的好处是有利于升级过程中用户自定义数据的保留。安装新版专家包后Data目录的内容会被专家包中的新内容所覆盖，但User目录不受影响。因此如果用户在User目录里保存了自定义的数据文件，则新版的专家包会优先使用它。

该部分的相关处理代码在cnwizards\Source\Framework\CnWizOptions.pas下，其中WizOptions对象的GetUserFileName方法可以供专家获得读写时需要使用的数据文件的完整路径：

function GetUserFileName(const FileName: string; IsRead: Boolean;

 FileNameDef: string = ''): string;   
    {\* 返回用户数据文件名，如果 UserPath 下的文件不存在，返回 DataPath 中的文件名}   
    function CheckUserFile(const FileName: string; FileNameDef: string = ''):   
      Boolean;   
    {\* 检查用户数据文件，如果 UserPath 下的文件与 DataPath 下的一致，删除   
       UserPath 下的文件，以保证 DataPath 下的文件升级后，使用默认设置的   
       用户可以获得更新。如果两文件一致，返回 True}

调用GetUserFileName时可根据当前的读写需要，传入不包含路径的单独文件名和标识读还是写的布尔变量即可。

WizOptions对象中还包含了一些全局的变量和设置内容，具体可参考代码中的注释。

### 多语言处理机制

可挂接的多语言支持的实现及处理。

### 事件通知服务

框架提供的IDE通知封装及更底层的通知器服务。CnWizNotifier.pas便是专家提供的各种通知服务的接口。它通过比较简略的方式封装了各种通知器。用户只需提供一通知事件处理函数添加到服务，不需要的时候删除即可。

### IDE挂接服务

框架对IDE常规功能的挂接服务。

### IDE停靠窗体

IDE中的可停靠窗体，比如Object Inspector，呈SizeToolWin的风格，可以方便地拖放到代码编辑器边缘，也可和其他可停靠窗体组合。要实现IDE中的可停靠窗体，窗体本身必须继承于TDockableForm，此类所在的单元为DockForm，但在VCL中无源码存在，仅仅存在于DsnIdeXX/DesignIde 包中，所以专家包工程的链接选项中有它们。

另外，由于多语的要求，专家中的窗体必须继承自TCnTraslateForm，这样就有了一点冲突。为了解决此问题。专家新增了一CnWizIdeDock单元，从TDockableForm继承了一TCnIdeDockForm，同样实现了TCnTraslateForm的多语支持功能，因此专家中需要实现停靠功能的窗体，只需从TCnIdeDockForm继承就够了。

### 属性及组件编辑器系统

属性及组件编辑器的统一管理机制。

### 仓库专家系统

为编写仓库专家提供的公共服务及仓库专家管理器。

### 与WEB服务器的交互

由框架提供的专家插件与WEB服务器交互的统一接口。

## 深入研究IDE

### 查看IDE内部控件

我们可以使用DelSight工具来查看IDE的内部控件。DelSight是Bill Friedrich写的一专家，支持D7以及以上版本的IDE。安装后通过Alt+1的热键能调出IDE中当前控件的Children，Parent等树状信息，同时还可显示各个对象的属性。

### 分析IDE设计期包

通过分析IDE内核设计期包的Dump数据来调用隐藏的功能。

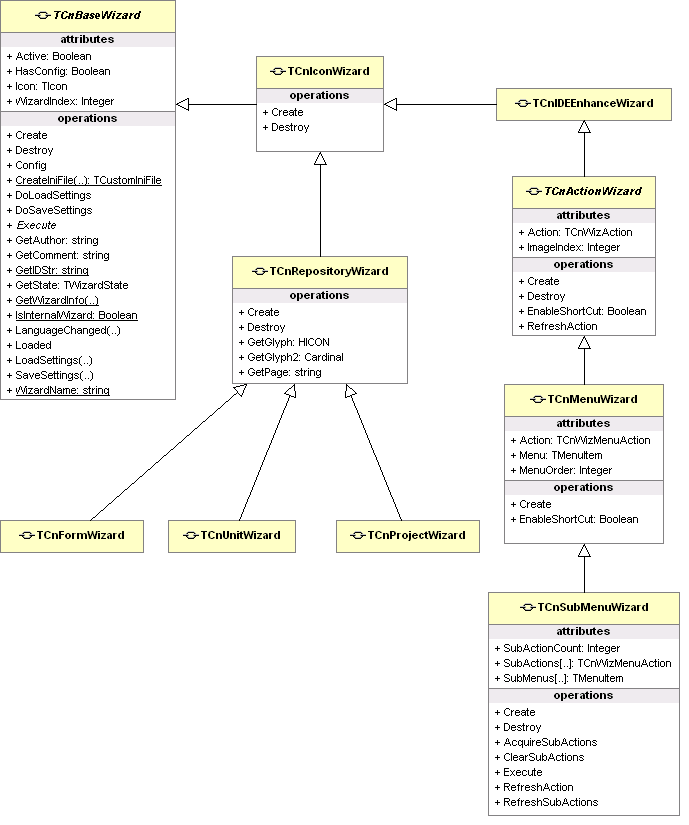
### 挂接IDE的控件和方法

通过对IDE内部控件的HOOK及对方法和函数的HOOK来实现特殊的功能。

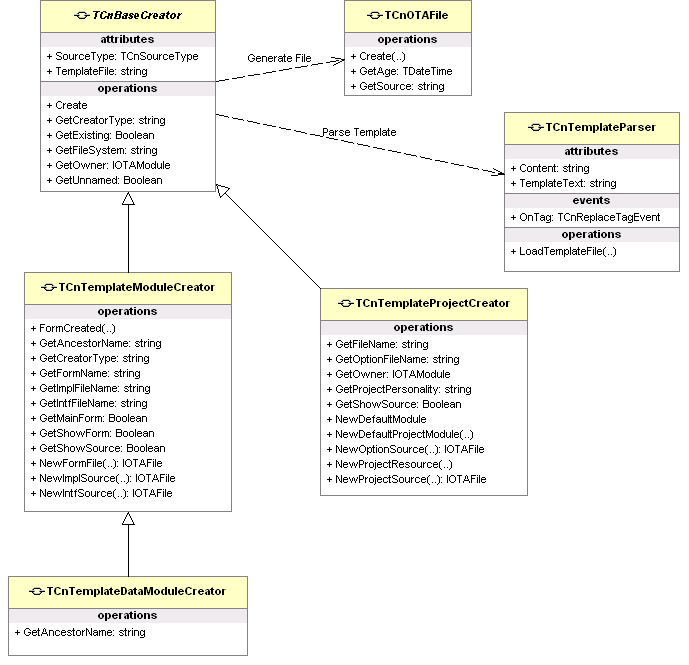
## 附 录

### 专家框架类图

专家框架中定义的主要类及接口的UML图。



专家基础类图



CnOTACreators类关系图