**期货交易管理系统V6前台程序开发指南**

# 修改历史

## V 0.9

将所有的开发技巧按照统一的格式进行规划处理。

## V 0.8

将文档规划为几个大部分：“开发规范部分”、“程序核心设计部分”、“开发入门部分”、“具体编码部分”、“程序清单”。

合并了“典型的界面布局.doc”文档。

合并了“集中交易柜台程序技术核心.doc”和“集中交易柜台程序UML模型.doc”文档。

合并了各个开发技巧文档。

合并了完全实例讲解文档。

删除了“开发组织、每日构造”等内容。

## V 0.7

将菜单文件的说明合并进来。

将几个模版的使用说明合并进来。

将其他的一些编码说明合并进来。

将一般开发流程合并进来。

## V 0.6

版本管理部分从TeamSource切换到SourceSafe

编码规范部分增加了对事件处理方法命名的的形式。

增加了对程序运行方式的说明。

## V 0.5

合并了程序框架文档。

## V 0.4

增加对配置文件的说明。

增加按键热键说明。

增加了界面操作指南。

## V 0.3

增加了使用TeamSource管理项目代码的注意事项。

增加了编写程序包的注意事项。

## V 0.2

增加和修改了：

运行环境

源程序的组织

目标程序的目录结构

## V 0.1

综合现有的文档，完成初稿。

# 一、前言

本文档的目的是给国通集中交易前台程序开发人员提供一个程序开发的规范说明。

# 第一部分 开发规范部分

# 二、环境

## 1、开发环境

### 硬件要求

CPU： PII 300以上。

RAM： 128M。

显示器： 14’以上，建议15’ 以上。

显卡： 2M显存以上，建议4M以上。

硬盘： 8G以上，建议10G以上。

### 软件要求

操作系统：中文Win98以上或者中文Win2000以上，注意打补丁。

主要开发工具：Delphi5和补丁1。

版本管理工具：TeamSource和补丁1。

其它工具软件：

中文MS office 97以上。

中文IE5以上，包含outlook express。

WinZip（或者其他支持压缩文件处理的工具）、Ultra Edit（或者其他文本编辑工具）。

Norton Anti Virus（或者其他支持实时病毒监测的工具）。必须打开实时病毒监测。

### 环境的建立

1）安装需要的软件，重点是Delphi5和Delphi5的补丁1。

2）在本地计算机建立一个文件夹“KSClient”。

3）通过版本管理工具（SourceSafe）获得源代码和支持文件，在获得的时候注意设置工作路径WorkFolder为前面创建的“KSClient”目录，保证获得的文件都安装版本管理工具上面的目录结构保存到本地目录。

4）设置环境变量Path，包含“KSClient\Bin”。

5）在“KSClient”下面创建子目录“KSLib”

6）获得开发需要的组件包，复制到KSLib下面的“BPL”（保存\*.BPL文件）和“DCU”（保存\*.DCU文件）子目录。

7）设置环境变量Path，包含“KSClient\KSLib\BPL”。

8）在Delphi里面使用“Component|Install Packages”菜单，安装设计时候需要的程序包。需要安装的程序包在“KSClient\KSLib\BPL”下面，安装顺序如下：

DKSBaseX50.BPL、

DKSUser50.BPL、

DKSStyle50.BPL、

DKSWorkView50.BPL、

DKSHDB50.BPL、

DKSCOM50.BPL、

DKSReport50.BPL。

9）在Delphi里面设置Library Path，包含“KSClient\KSLib\BPL”和“KSClient\KSLib\DCU”。

10）在Delphi里面打开工程组“KSClient\国通50.bpg”，选中“DKSFramework50.BPL”（工程组包含的第二个工程），然后在右键菜单里面选择“Install”。成功安装以后，进行全编译（“Project|Compile All Projects”菜单）。如果全部编译成功，说明开发环境已经配置完毕。

**小结和注意事项**：

1）需要在系统的环境变量Path里面增加“KSClient\Bin”和“KSClient\KSLib\BPL”。

2）需要在Delphi里面安装程序包，除了“KSClient\KSLib\BPL”目录下面“D”开头的设计组件包以外还有“DKSFramework50.BPL”。

3）需要设置Library Path。

## 2、运行环境

### 硬件要求

CPU：Pentium以上

显示卡和显示器：800\*600，256色（强烈建议65535色以上）

内存：64M以上（建议128M）。

硬盘：剩余空间400M以上（提供交换内存）

### 软件要求

操作系统： Win98+IE5。(IE5用于浏览HTML格式的帮助)

# 三、编码规范

## 1、Delphi程序编码规范

参见文档：《Delphi编码规范草案.doc》

## 2、针对国通集中交易前台程序的补充

变量、函数、对象、单元命名的时候使用英文或者缩写命名，具体的中英文证券词汇对照表参考文档《集中交易系统—命名规范new.doc》。遵照标准的Object Pascal命名规范，前台Delphi程序命名和《集中交易系统—命名规范new.doc》文档中的规定有一些的小区别：

1）不使用下划线“\_”分隔各个单词；

2）每个单词的首字母大写。

例如：

|  |  |  |
| --- | --- | --- |
| 词汇 | 《集中交易系统—命名规范new.doc》规定的命名 | 前台程序中的命名 |
| B股 | b\_stock | BStock |
| 比例 | scale | Scale |
| 补购 | patch\_buy | PatchBuy |

## 3、对工作视图、工作字段、监控器的事件命名

缺省情况下，Delphi IDE对工作字段和监控器的事件处理方法自动命名，常见形式如下：

1）“WorkViewWorkFields[I]ValueChanged”（I表示工作字段在字段列表里面的下标索引），例如WorkViewWorkFields0ValueChanged、WorkViewWorkFields2ValueChanged

2）“WorkViewFieldsMonitors[I]ValidChanged”（I表示监控器在监控器列表里面的下标索引），例如WorkViewFieldsMonitors2ValidChanged。

3）其他还有检查字段有效性等等。

如此缺省命名有如下缺点：没有表达出该事件处理完成什么功能、在什么条件下面该事件触发（只有字段下标索引，难以阅读）。另外当字段或者监控器的顺序发生改变的时候，维护起来非常困难，容易出错。

正确的事件编写方式和事件处理的命名方式如下：

|  |  |  |
| --- | --- | --- |
| **完成的功能** | **编写方式** | **事件处理的命名** |
| 根据字段的有效性控制按键的Enabled | 在监控器里面的OnValidChanged编写事件 | 命名方式**DoUpdate+按键名称+Btn**，例如：DoUpdateOkBtn、DoUpdateQueryBtn |
| 控制一个字段的有效性 | 应该在该字段的OnCheckValid里面编写事件，如果该字段受其他字段影响，注意设置MonitorValueChangedFields属性 | 命名方式**DoCheck+字段名+Valid**，例如：DoCheckStockValid、DoCheckCustomerValid |
| 根据一个或者一些字段获取其他数据 | 如果是通过一个字段获取数据，可以在该字段的OnValueChanged事件里面处理，如果是通过多个字段获取数据，可以通过监控器处理。  建议：都通过监控器处理，这样以后可以增加影响的字段。 | 命名方式**DoGet+数据名称+From+主要的字段名**，例如GetMaxBuyAmountFromCustAndPrice（主要根据客户号和价格（还有其他参数）获得最大可买数量） |
| 一些数据改变，关闭以前的结果集 | 如果是受一个字段影响，可以在该字段的OnValueChanged事件里面处理，如果是受多个字段影响，可以通过监控器处理。  建议：都通过监控器处理，这样以后可以增加影响的字段。 | 命名方式**DoCloseDatasets**或者**DoCloseDatasetsOn+主要的字段名+Changed** |
| 其他处理 | 在字段的事件或者监控器的事件里面编写程序。 | 命名方式**Do+主要的动作+触发条件**  **主要的动作**使用动词，**触发条件**可以使用**From/On+字段名称**的形式。例如在营业部内部参数设置里面只有市场和币种其中的一个输入有效，那么在输入市场的时候清空币种，在输入币种的时候清空市场，命名如下：DoClearCurrencyOnMarketChanged、DoClearMarketOnCurrencyChanged。 |

命名方式小结：

1）使用“**Do**”前缀区分事件处理方法和对象的其他私有/公共方法。

2）将关键的处理动作“**Update Btn**”、“**Check Valid**”、“**Get**”和处理的对象名称“**按键名称**”、“**字段名称**”、“**数据名称**”表现到事件处理方法命名里面。

3）表明了触发事件的主要条件，在“**DoCheck+字段名+Valid**”里面，触发处理的条件是检查字段的有效性。在“**Get+数据名称+From+主要的字段名**”里面，触发处理的条件是“**主要的字段名**”对应的字段发生改变的时候触发。

4）这样命名的长度一般比缺省的命名短小，同时增强了可读性，在字段和监控器改变顺序的时候，名称不用发生改变。

# 四、源程序的组织

## 1、源代码的版本控制

### 1.1、选用SourceSafe作为版本控制工具。

### 1.2、术语说明

Check-In：将本地修改以后的文件加入到版本控制中，版本号加一。

Check-Out：将版本控制中的当前版本（一般是最新版本）复制到本地。

### 1.3、版本控制的目录和内容

下面的这个目录结构描述了源文件的基本组织形式。

**注意**：详细的目录结构和工程文件列表见“第四部分 程序清单”。

|  |  |  |  |
| --- | --- | --- | --- |
| **目录** | **内容** | **使用者和使用方式** | **备注** |
| \Framework | 程序框架需要用到的公共代码的源文件。（业务无关）。 | 所有人：CheckIn / ChekcOut。 |  |
| \Main | 主程序的源文件。（业务无关）。 | 所有人：CheckIn / ChekcOut。 |  |
| \Templates | 各种程序界面模版 | 所有人：CheckIn / ChekcOut。 |  |
| \客户管理 | 客户管理部分的源代码 | 所有人：CheckIn / ChekcOut。 | 下面包含子目录，每个子目录对应一个菜单界面的实现。 |
| \客户管理\客户开户 | 综合委托的源代码 | 所有人：CheckIn / ChekcOut。 |  |
| 其它大模块 | 同上 | 同上 | 同上  按照大模块的形式划分 |
| \Report | 报表工具和报表文件 | 同上 |  |
| \Help | 帮助文件 | 同上 |  |
| \Debug | 支持调试的程序包 | 同上 |  |
| \Robot | 支持自动化的程序包 | 同上 |  |
| \Setup | 安装程序 | 同上 |  |
| \Support | 程序运行需要的支持DLL | 同上 |  |
| \ConfigSetup | 配置安装程序 | 同上 |  |
| \CopyRight | 版权程序包 | 同上 |  |
| \ PwdKeyboard | 密码小键盘 | 同上 |  |
| \rwcard | 读写国通卡 | 同上 |  |

### 1.4、需要注意的问题

1）注意工程中应该只包含源代码，不要包含\*.dcu、\*.exe、\*.~\* 等文件。

## 2、源程序的结构

令“$SOURCE”表示源程序的根目录。

**注意**：详细的目录结构和工程文件列表见“第四部分 程序清单”。

### 2.1、部分工程文件一览

|  |  |  |
| --- | --- | --- |
| **目录** | **工程文件** | **说明** |
| $SOURCE | 国通50.bpg | 包含所以的工程文件的工程组(Project Group)。 |
| $SOURCE\Framework | KSFramework50.dpk | 构造集中交易程序框架的程序包的工程文件。 |
| $SOURCE\Framework | DKSFramework50.dpk | 设计时候需要的程序包 |
| $SOURCE\Main | KSClient50.dpr | 构造主程序的工程文件。 |

### 2.2、部分工程文件说明

#### 1、KSFramework50.dpk

构造集中交易程序框架的程序包的工程文件。

包含文件一览

|  |  |  |
| --- | --- | --- |
| **目录** | **文件** | **说明** |
| $SOURCE\Framework | KSFrameWorks.pas | 定义程序框架 |
| KSClientConsts.pas | 前台框架程序中使用到的常量 |
| KSFrameWorkUtils.pas | 辅助的函数 |
| KSFrames.pas | 操作界面的基类 |
| $SOURCE\Templates | UInputTemp.pas  UInputTemp.dfm | 输入界面模版 |
| UQueryTemp.pas  UQueryTemp.dfm | 查询界面模版 |

#### 2、KSClient.dpr

构造主程序的工程文件。

|  |  |  |
| --- | --- | --- |
| **目录** | **文件** | **说明** |
| $SOURCE\Main | UMain.pas  UMain.dfm | 主程序界面 |
| UDatabase.pas  UDatabase.dfm | 数据库连接 |
| UContext.pas  UContext.dfm | 1、实现了上下文对象，提供运行环境  2、负责启动和关闭系统 |
| KSMenus.pas | 菜单处理 |
| UFieldDomains.pas  UFieldDomains.dfm | 统一的字段域 |
| UStyles.pas  UStyles.dfm | 统一的界面风格定义：字体、背景颜色、按键。 |
| UBrowser.pas  UBrowser.dfm | 实现向导 |
| USearch.pas  USearch.dfm | 实现界面搜索 |
| UWait.pas  UWait.dfm | 实现等待界面 |

### 3.3、编写程序包的注意事项

1）功能程序包的引用（“requires”）的其他程序包出了标准Delphi5包含的程序包(VCL\*50)以外，只能包含：“KSFramework50”和“KSGTPlatform50”两个程序包。特别注意不要包含“VCLBDE50”这个程序包。如果在编译的时候要求包含“VCLBDE50”这个程序包，说明在代码里面引用了“DBTables”这个unit，应该把这个引用去掉。

2）功能程序包的的“Project Options”里面应该设置“search path”包含“\bin”这个目录。一般设置为“..\bin”（假定程序包所在的目录是“$SOURCE\XX\”）。

3）功能程序包的的“Project Options”里面应该设置“output directory”为“\bin”这个目录。一般设置为“..\bin”（假定程序包所在的目录是“$SOURCE\XX\”）。这样编译产生的BPL和DCP文件保存到“\bin”目录下，程序运行的时候才可以找到这些程序包。

4）功能界面应该从标准的模版继承下来。

5）开发完一个新的功能界面程序包以后，要检查：

a）功能界面是否正确注册（通过界面工厂类）。

b）要保证菜单文件里面的UICode和程序中设置的相同。

c）要检查配置文件里面是否增加了该程序包的条目。

# 五、目标程序的组织

## 1、目录结构

安装目录是在文件服务器上面安装柜台所指定的目录。建议的安装目录是 C:\Program Files\Kingstar\KSClient50。令$HOME表示产品的安装目录，安装目录下面的目录结构如下：

|  |  |  |
| --- | --- | --- |
| **目录** | **包含的文件** | **说明** |
| $HOME\ | readme.txt  readme.htm  licence.txt | 包含说明文件和版权说明文件。 |
| $HOME\bin | KSClient.exe | 主程序。所有的执行程序(exe)和需要的动态库(dll)、程序包(bpl)。因为这些部件位于相同的目录下，所以不需要设定专门的搜索路径（path）。本产品应该不在 windows\system 目录下安装文件，这样便于产品的安装和卸载。 |
| vcl50.bpl  vcldb50.bpl  vcldbx50.bpl  vclie50.bpl  vcljpg50.bpl  vclsmp50.bpl  vclx50.bpl | Delphi5的标准程序包 |
| KSGTPlatform.bpl | 金仕达的基础代码程序包（包括报表、流程控制、命令模型、程序外观控制、通讯平台接口等等） |
| KSFramework50.bpl | 集中交易程序框架的程序包 |
| drtpapi.dll  kk\_crypt32.dll | 通讯平台需要的支持DLL |
| BasicDataAccess.tlb  tregsvr.exe | 支持数据访问的文件 |
| $HOME\help | HelpIDs.txt | 帮助ID与帮助文件名的映射关系。 |
|  | \*.htm | HTML形式的帮助文件。 |
| $HOME\configs | ksclient.ini | 包含程序的各种配置参数的配置文件。 |
| menus.mns | 菜单文件，包含菜单基本结构和相关业务的菜单. |
| favorites.mns | 常用功能菜单。 |
| $HOME\report | \*.rpt | 报表格式文件。 |
| $HOME\User | （程序在独立运行模式下）保存用户使用信息 | 用户帐号.ini文件，例如“012888.ini”。  表格显示格式配置文件(\*.grid)，例如“营业部内部参数.grid” |
| $HOME\Output | （程序在独立运行模式下）保存输出文件。输出文件包括查询结果打印到文件。 |  |

**注意**：1、程序中要注意路径。2、不要使用注册表，方便产品的安装和卸载。

## 2、配置文件的内容

### 2.1 ksclient.in和ksclient.ini

在configs目录下面包含“ksclient.in”和“ksclient.ini”两个配置文件。其中“ksclient.ini”是标准的windows配置文件（文本形式），“ksclient.in”是相应的加密以后的二进制文件。

注意：维护的时候应该先修改“ksclient.ini”文件的内容，然后使用“SysFileMan50.exe”工具程序将它转换为“ksclient.in”。

ksclient.ini包含针对整个系统的配置信息。具体内容如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **组名** | **键值** | **值** | **举例** |
| server | IP | 通讯平台服务器的ID地址 | IP=172.16.80.3 |
| Port | 通讯平台服务器的端口号 | Port=2000 |
| IP2 | 备用通讯平台服务器的ID地址。当第一台通讯平台无效的时候，被使用。 | IP=172.16.80.3 |
| Port2 | 备用通讯平台服务器的端口号 | Port=2000 |
| DestNo | 目的编号 | DestNo=0 |
| FuncNo | 功能号 | FuncNo=888 |
| Encode | 加密方式 | Encode=0 |
| TimeOut | 超时（毫秒为单位） | TimeOut=30000 |
| Prior | 优先级缺省为2 |  |
| DelayShowMessage | 是否延时显示等待对话框 | DelayShowMessage=1 |
| DelaySeconds | 延时显示等待对话框的秒数 | DelaySeconds=3 |
| Department | ID | 营业部代码 | ID=012 |
| Name | 营业部名称 | Name=南头营业部 |
| Packages | Count | 最大的程序包数目 | Count=32 |
| <程序包序号> | 程序包文件名。  注意：  1、程序包序号在1~最大的程序包数目之间。  2、序号小的程序包先装载。  3、程序包文件应该在bin目录下。  4、使用的程序包序号可以不连续  注意：要使用密码小键盘，一定要在Packages里面装载KSPwdKbd50.bpl程序包。 | 11=SysAdmin50.BPL  12= |
| Main | MaxUICount | 可以同时打开的最大界面个数 | MaxUICount=30 |
| NestedMenu | 菜单是否缩进 | NestedMenu=1 |
| ConfirmClose | 改变操作界面的时候是否提示 | ConfirmClose=0 |
| UseThread | 启动的时候是否使用初始化线程 | UseThread=0 |
| ShowMenuIcon | 显示的菜单是否包含图标，缺省True | ShowMenuIcon=0 |
| ShowGroupChar | 是否为菜单显示“+”，缺省False | ShowGroupChar=1 |
| ShowTutorial | 是否在启动的时候显示教程，缺省True | ShowTutorial=0 |
| AutoLockSeconds | 用户没有操作一定秒数以后，自动锁定程序。缺省600秒。如果<=0，不自动锁定程序。绝对值小于300，按照300计。绝对值表示秒数。 | AutoLockSeconds＝300 |
| AutoFitSize | 是否根据界面大小自动调整 | AutoFitSize＝0 |
| MultiQuery | 是否可以同时打开多个相同的查询界面，缺省为true | MultiQuery=1 |
| MultiQueryInterface | <操作界面> | 0：不可以同时打开多个相同界面。  1：可以同时打开多个相同界面。  缺省为0 | 客户开户=0 |
| Log | Command | 是否对命令写日志，缺省False。 | Command=1 |
| DataPack | 是否对数据包写日志，缺省False。 | DataPack=0 |
| ObjectLife | 是否对对象的创建和释放写日志，缺省False。 | ObjectLife=0 |
| KCDataset | 是否对数据集写日志，缺省False。 | KCDataset=1 |
| Application | 是否对应用程序的信息（启动、结束等等）写日志。缺省True。 | Application=1 |
| Debug | 是否对调试信息写日志，缺省False。 | Debug=0 |
| Input | CustomerNo | 客户号补齐方式 | CustomerNo=12000000 |
| Sequence | 账号输入顺序() | Sequence=02143 |
| OperatorNo | 职工代码补齐方式 | OperatorNo=012000 |
| DefaultMethod | 初始的输入方式，缺省“0”(客户号) | DefaultMethod=1 |
| BrokerNo | 经纪人号补齐方式 | BrokerNo=012000 |
| MaxEntrustVol | 最大可委托数 | MaxEntrustVol=999999 |
| MaxCashSum | 最大现金存取金额 | MaxCashSum=5000000 |
| MaxBankBookCheckSum | 最大银行存折(支票)存取金额 | MaxBankBookCheckSum=1000000000 |
| SecurityLength.市场代码 | 证券代码长度，缺省为6。标准的四个市场有缺省值 | SecurityLength.1=6 |
| SecurityPattern.市场代码 | 证券代码补齐方式。标准的四个市场有缺省值。 | SecurityPattern.1=’600000’ |
| AccountLength.市场代码 | 股东代码长度。标准的四个市场有缺省值。 | AccountLength.1=10 |
| AccountPattern.市场代码 | 股东代码补齐方式。标准的四个市场有缺省值。 | AccountPattern.1=’A000000000’ |
|  |  |  |
| Confirm | <操作界面> | 是否出现确认对话框。  0表示不出现确认对话框  1表示出现特殊的确认对话框  2表示出现普通的确认对话框  支持的操作界面 | 客户开户=1 |
| PrintReceipt | <操作界面> | 是否打印回单。  0表示不打印  1表示弹出询问打印的对话框  2表示直接打印  支持的操作界面 | 客户开户=1 |
| Preview | 打印以前是否预览，缺省True。 | Preview=0 |
| DoubleCheck | <操作界面> | 是否单机双柜员复核 | 客户开户=0 |
| Monitor | 资金存取监控阀值 | 通用(不分币种) | 资金存取监控阀值=500000 |
| 资金存取监控阀值.01 | 人民币，单位圆 | 资金存取监控阀值.01=500000 |
| 资金存取监控阀值.02 | 美元，单位圆 | 资金存取监控阀值.02=50000 |
| 资金存取监控阀值.03 | 港币，单位港元 | 资金存取监控阀值.03=500000 |
| 委托买卖监控阀值 | 通用(不分币种) | 委托买卖监控阀值=300000 |
| 委托买卖监控阀值.01 | 人民币，单位圆 | 委托买卖监控阀值.01=300000 |
| 委托买卖监控阀值.02 | 美元，单位美元 | 委托买卖监控阀值.02=30000 |
| 委托买卖监控阀值.03 | 港币，单位港元 | 委托买卖监控阀值.03=300000 |
| 成交回报监控阀值 | 通用(不分币种) | 成交回报监控阀值=300000 |
| 成交回报监控阀值.01 | 人民币，单位圆 | 成交回报监控阀值.01= 300000 |
| 成交回报监控阀值.02 | 美元，单位美元 | 成交回报监控阀值.02= 30000 |
| 成交回报监控阀值.03 | 港币，单位港元 | 成交回报监控阀值.03= 300000 |
| 刷新时间间隔 | 单位：秒 | 刷新时间间隔=10 |

### 2.2 allusers.ini

user\allusers.ini 也是标准的windows配置文件，用于保存针对所有用户的使用信息。

|  |  |  |  |
| --- | --- | --- | --- |
| **组名** | **键值** | **值** | **举例** |
| AllUsers | User.营业部代码 | 表示最近一次登录到某营业部的职工代码。 | User.012=012888  User.000=000888 |
| Device | PwdComNo | 1：密码小键盘接在Com1  2：密码小键盘接在Com2  注意：要使用密码小键盘，一定要在Packages里面装载KSPwdKbd50.bpl程序包。 | PwdComNo=1 |
| PwdTimeOut | 密码小键盘的超时 |  |

### 2.3 用户.ini

user目录下面还包含“<用户帐号>.ini”形式（例如“012888.ini”）的配置文件。这些文件保存对应用户的使用信息。

|  |  |  |  |
| --- | --- | --- | --- |
| **组名** | **键值** | **值** | **举例** |
| User | RootMenu | 最近一次使用的根菜单的ID | RootMenu=营业部菜单 |
| UserMenus | Count | 用户个人菜单的菜单项目的个数 | Count=6 |
| <序号> | 用户个人菜单的一个菜单项目的ID。序号在1～Count之间。 | 1=客户开户  2=现金存取  3=存折存取 |

## 3、程序运行方式

### 3.1、独立运行方式

在该方式下面，程序安装目录下面包含所有的在第一小节里面说明的目录和文件。

**优点**：每个用户可以有自己独立的配置。

### 3.2、简单运行方式

在该方式下面，首先要在一台文件服务器上面安装前台程序。安装目录下面包含除“User子目录”和“Output子目录”以外的目录和文件。然后将该安装目录共享（共享权限为“只读”）出来，假定共享名为“\\fileserver\ksclient”。

在“\\fileserver\ksclient\bin”下面包含一个“start.ini”文件。start.ini文件的内容如下：

|  |
| --- |
| [start]  **mode=1** |

然后在需要运行前台程序的柜员电脑创建一个程序运行快捷方式，指向“\\fileserver\ksclient\bin\ksclient50.exe”。通过该快捷方式可以运行程序。假定柜员电脑的Windows操作系统安装在“D:\Windows”，那么柜台程序会自动建立“D:\KSClient\User”、“D:\KSClient\Output”目录，分别保存用户数据和输出数据。

运行的时候，执行程序、配置文件、报表格式文件、帮助文件都从\\fileserver\ksclient获取。用户文件、输出文件保存在本地“D:\KSClient”下面。

注意：如果柜员电脑的Windows操作系统安装在其他盘符上面，用户文件、输出文件的保存目录也安装在对应的盘符上面。

**优点**：1、管理员只需要在文件服务器一台电脑上面修改了配置，所有其他的以“简单运行方式”运行的前台程序都可以使用到这个正确的配置。可以大大简化配置。

2、最小化程序安装

### 3.3、混合运行模式

在该方式下面，首先要在一台文件服务器上面安装前台程序，安装目录下面包含所有的目录和文件。然后将该安装目录共享（共享权限为“只读”）出来，假定共享名为“\\fileserver\ksclient”。

然后在需要运行前台程序的电脑上面安装“混合运行模式”的程序，安装目录下面只包含如下目录：

|  |  |
| --- | --- |
| **目录** |  |
| $HOME\bin |  |
| $HOME\User |  |
| $HOME\Output |  |

在“$HOME\bin”下面包含一个“start.ini”文件。start.ini文件的内容如下：

|  |
| --- |
| [start]  **base=\\fileserver\ksclient**  **mode=2** |

运行的时候，配置文件、报表格式文件、帮助文件都从\\fileserver\ksclient获取。执行程序、用户文件、输出文件保存在本地。

**优点**：1、管理员只需要在文件服务器一台电脑上面修改了配置，所有其他的以“简单运行方式”运行的前台程序都可以使用到这个正确的配置。可以大大简化配置。

2、启动速度比较快，因为执行程序在本地。

### 3.4、小结

建议实际使用的程序以简单运行方式运行。开发时候使用的程序以独立运行方式运行。

### 3.5、说明

缺省的安装程序按照简单运行模式安装柜台应用程序。

## 4、菜单配置文件

### 4.1、菜单文件

在configs目录下面包含“menus.mn”和“menus.mnt”两个和菜单相关的配置文件。其中“menus.mnt”是文本文件，“menus.mn”是相应的加密以后的二进制文件。

注意：维护的时候应该先修改“menus.mnt”文件的内容，然后使用“SysFileMan50.exe”工具程序将它转换为“menus.mn”。

下面的格式说明是针对文本文件“menus.mnt”。

menus.mnt菜单文件是基于行的文本格式，每一行代表一条菜单信息。空白行和“;”开始的注释行被忽略掉。

### 4.2、具体格式

**[menus]**

;第一部分，菜单项目定义。

;开始一个菜单的定义

**ID**=菜单唯一标志，中文，不超过10个汉字。

**.no**=数字序号，用于权限位。0表示无关。

**.type**=类型，“group”表示组，“root”表示根。

**.caption**=标题。

**.keyword**=包含的关键字，用“,”分割。

**.see**=相关的菜单的ID，用“,”分割。

**.func**=对应的功能项的标志（数字），用“,”分割。

**.des**=相应的说明文字，限制在一行以内。

**.hide**=在权限设置的时候是否隐藏，用于特殊菜单的控制。

**.spell**=汉语拼音缩写。如果没有设置，那么自动生成，不过自动生成的在处理多音字的时候不准。

**.virtual**=虚拟的菜单，不在界面上面显示，仅仅显示在权限管理里面，用于组织操作权限。

;结束一个菜单的定义

;下一个菜单的定义

**[tree]**

;第二部分，菜单树状结构定义

%根菜单项目。

#一级项目，包含子项目。

-+二级项目，包含子项目。

--三级项目，叶节点，不包含子项目。

--+三级项目，包含子项目。

---四级项目，叶节点，不包含子项目。

;其他依次类推。

### 4.3、例子

**[menus]**

**;第一部分，菜单项目定义。**

**;一个根节点**

ID=营业部菜单

.no=0

.type=root

.caption=营业部菜单

.keyword=

.see=

.func=

.des=

;\*\*\*\*\*\*\*\*\*\*\*\*\*

**;一个组**

ID=客户管理

.no=2

.type=group

.caption=客户管理

.keyword=

.see=

.func=

.des=

;\*\*\*\*\*\*\*\*\*\*\*\*\*

ID=开户

.no=3

.type=group

.caption=开户

.keyword=

.see=

.func=

.des=

;\*\*\*\*\*\*\*\*\*\*\*\*\*

**;一个叶节点，注意.func**

ID=客户开户

.no=4

.type=

.caption=客户开户

.keyword=

.see=

.func=87

.des=

;\*\*\*\*\*\*\*\*\*\*\*\*\*

ID=预留卡开户

.no=5

.type=

.caption=预留卡开户

.keyword=

.see=

.func=90,721

.des=

;\*\*\*\*\*\*\*\*\*\*\*\*\*

**;一个定义比较完整的叶节点**

ID=现金存取

.no=45

.type=

.caption=现金存取

.keyword=现金,资金,存取

.see=客户综合信息查询,资金明细

.func=135,136

.des=以现金的形式为客户存取资金

;\*\*\*\*\*\*\*\*\*\*\*\*\*

**;一个隐藏菜单的例子，注意.hide=1**

ID=营业部常用菜单

.no=0

.type=root

.caption=常用菜单

.keyword=

.see=

.func=

.des=

.hide=1

;\*\*\*\*\*\*\*\*\*\*\*\*\*

**[tree]**

**;第二部分，菜单树状结构定义**

%营业部菜单

#证券业务

-+普通委托

--简单委托

--境外委托

--补委托单

--等量批量委托

--总量批量委托

--批量撤单

--债券转股

--债券回售

-+特殊委托

--买入融资

--卖出融券

--深圳转托管

#资金业务

-+资金存取

--现金存取

--支票存取

--存折存取

--资金复核

--冲帐处理

--强制资金取出

--内部转帐

--帐面调整

--息转本金

%营业部常用菜单

客户开户

添加股东代码

修改委托方式

现金存取

存折存取

银行转帐请求

银行转帐查询

简单委托

批量撤单

## 5、帮助文件

在Help目录下面是帮助文件。帮助文件包括超文本格式文件（\*.htm）、各种图像文件（\*.gif;\*.jpg等等）、JavaScript文件（\*.js）、超文本样式文件（\*.css）等等。可以使用Web浏览器直接浏览这些文件。

该目录下面还有一个“HelpIDs.txt”文本文件。该文件里面列出上下文相关帮助的ID和实际的帮助文件的对应关系，下面是该文件的一个例子。

|  |
| --- |
| #NotImp=NotImp.htm  #Index=Index.htm  客户开户=客户开户.htm  批量添加股东代码=批量添加股东代码.htm |

注意：如果上下文相关帮助的ID没有在该对应关系里面出现，那么使用缺省的对应关系，即假定对应的帮助文件的文件名和标题相同，例如“客户开户”对应“客户开户.htm”。

# 六、界面规范

## 1、界面规范的基本原则

1、平面型界面设计，基本上不使用弹出窗口，没有窗口之间的相互重叠。只有各种提示窗口（例如提示用户确认操作、显示错误信息）才使用弹出窗口。

2、程序占据全屏幕，充分利用屏幕空间，并且视觉上不受其他程序的干扰。

3、非模式化窗口设计，用户可以同时保留多个操作界面（主要针对各种查询操作界面），并通过点击“分页”控制在各个操作界面之间切换。

4、文字的字体颜色和背景颜色的搭配以清晰为主。

5、主要支持鼠标操作，也支持各种键盘快捷键。对操作速度有要求的操作界面对键盘有特殊的快速操作支持。

6、界面主要针对800\*600像素，256色(推荐65535色)以上设计。

7、各种文字的字体颜色应该统一。除了特殊的提示目的以外，所有的提示文字应该使用相同的字体和颜色。同样的，所有的按键的也应该使用相同的字体、颜色。对功能相同的按键（例如出现在不同操作界面的“打印”按键）应该使用相同的图标。按键的大小应该统一。

8、应该明显的区别输入区域和只读（显示）区域，明显的区别有输入焦点的区域和无输入焦点的区域。例如针对编辑框，只读的编辑框使用灰色背景，有输入焦点的编辑框使用黄色背景，无输入焦点的编辑框使用白色背景。

9、让用户可以方便的使用常用的操作界面。

## 2、界面的基本布局

1、屏幕的最上边一条的左边是菜单区，为用户提供基本的操作。右上角是窗口控制图标，用于最小化程序和退出程序。

2、屏幕的中间区域的左边是供用户选择操作功能的树状图和操作功能的说明。为了便于查找，树状图按照操作功能被分成几个组。

3、屏幕的中间区域的右上面一条是显示当前打开的界面标题的分页控制和操作界面控制图标。分页控制用于切换操作界面。控制图标用于最大化/关闭/保存当前的操作界面。

1）在最大化操作界面的时候，隐藏树状图和操作功能的说明，为操作界面提供最大的显示空间。

2）当操作界面没有被保存的时候，打开新的操作界面的时候，将自动关闭旧的操作界面。

4、屏幕的中间区域的右边是具体的操作区域。



5、屏幕的最下边是状态条，显示系统状态和提示信息。

## 3、操作界面的内部布局

1、使用不同于周围（菜单、树状图）的背景色，以区别具体的操作界面的区域。

2、所有的操作按键集中到最下边的一个条带区域。该区域的背景颜色与操作区域的背景颜色不同，以示区别。为了美观和形象一致，按键应该向操作区域的右边对齐。

## 4、查询界面

操作界面分为两页，第一页是条件输入区域，第二页是查询结果显示区域。可以用最大的屏幕区域显示查询结果。



## 5、参数浏览和修改界面

参数设置界面(例如"委托权限设置"、"证券参数设置")应该包含如下界面元素。

1）表格，列出所有的参数和当前的值；

2）表达参数值的一组控件（编辑框、下拉框等等），兼作录入控件；

3）命令按键（增加、修改、保存、取消）；

4）显示操作状态的状态条。 应该明确区分三种操作状态：浏览、增加、修改。

状态图如下：

浏览

表格：有效。

录入控件：只读，显示表格当前行的数据。

有效的按键：增加、修改

增加

表格：无效（不能滚动）。

录入控件：录入数据。

有效的按键：保存、取消

保存/取消

增加

修改

表格：无效（不能滚动）。

录入控件：代表主键的只读，其余可录入。

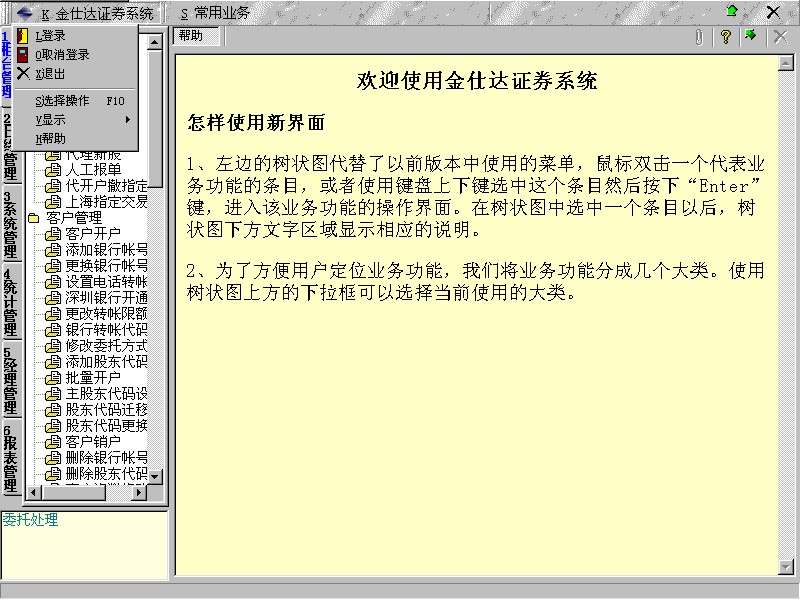
有效的按键：保存、取消

修改



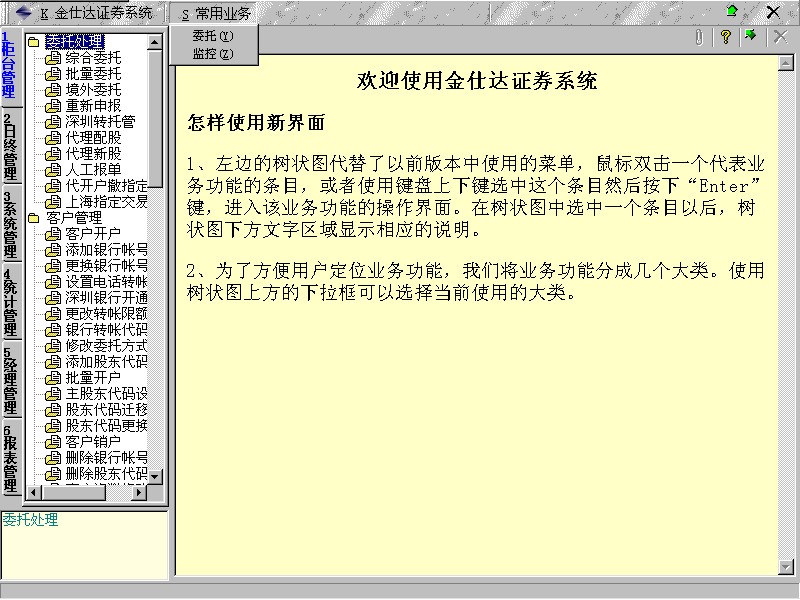
## 6、帮助系统

使用HTML格式保存在线使用帮助。用户从菜单中选择“帮助”或者按下“F1”，显示上下文相关帮助。帮助内容（基于嵌入的WebBrowserActiveX实现）显示在具体操作区域（即屏幕中间的右部）。



## 7、常用功能

常用功能集中在菜单上，用户可以快速选择。具体包含哪些常用功能，应该是用户可以配置的。



## 8、功能查找

由于证券交易和管理系统是一个复杂的系统，包含了许多操作界面。对一些不太常用的功能，用户往往难于找到适当的操作界面，所以需要为用户提供一个查找的功能。

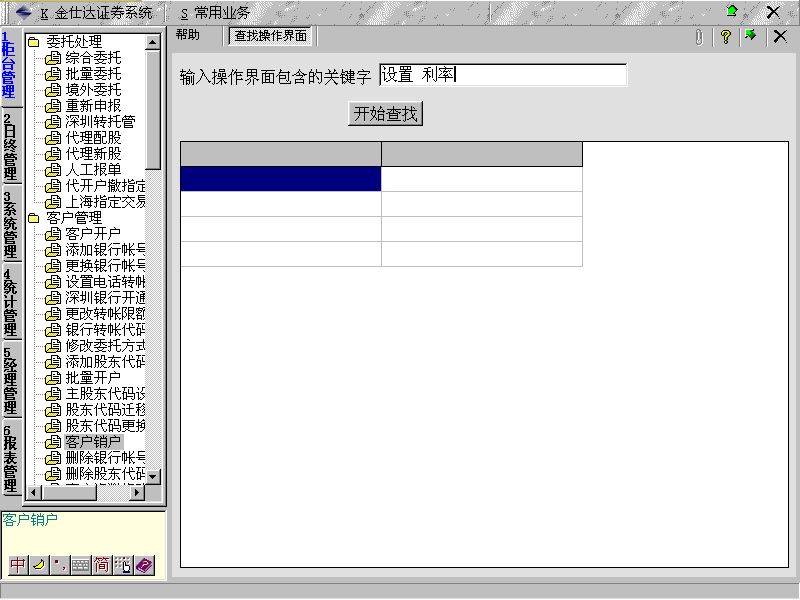
具体的使用方式是：

1、用户在菜单中选择“帮助|查找操作界面”，显示“查找操作界面”

2、在界面中输入一个关键字，例如“委托”、“利率”、“佣金”。

3、选择“查找”，显示出对应该关键字的操作界面（可能有多个），及其说明。

4、用户选择一个操作界面，显示之。



## 9、统一控制界面的外观风格

界面的外观风格包括各种文字的字体和颜色、各种背景色以及各种图标。为了保持风格的一致，统一字体、颜色和各种背景色。将各种用途的文字分成以下几组。

|  |  |  |  |
| --- | --- | --- | --- |
| 组别 | 用途 | 推荐的字体和颜色 | 背景颜色 |
| 提示文字 | 对输入项目的提示或者说明 | 宋体五号黑色 |  |
| 特殊提示文字1 |  | 宋体五号红色 |  |
| 特殊提示文字2 |  | 宋体五号绿色 |  |
| 输入文字 | 编辑框（Edit） | 宋体五号黑色 | 白色 |
| 输入文字带焦点 | 有焦点的编辑框 | 宋体五号黑色 | 黄色 |
| 只读文字 | 只读编辑框 | 宋体五号黑色 | 灰色 |
| 表格文字 | 表格（Grid） | 宋体小五号黑色 | 白色 |
| 表格列标题 | 表格的列标题 | 黑体小五号黑色 | 灰色 |
| 按键文字 | 按键（Button） | 黑体五号黑色 | 灰色 |
| 树状图 |  | 宋体小五号黑色 | 黄色 |

程序可以方便的修改特定组别的外观风格，修改以后自动影响相应的显示。

按键按照功能分组，相同的组具有相同的图标。

|  |  |  |
| --- | --- | --- |
| 功能（按键文字） | 图标（插入图片） | 大小（宽度像素\*高度像素） |
| 确定 |  |  |
| 取消 |  |  |
| 保存 |  |  |
| 查询 |  |  |
| 打印 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

程序中提供统一控制文字风格和图标的组件。通过这些组件方便界面的开发，同时也提供用户在运行时修改界面风格的能力。

## 10、系统热键

|  |  |
| --- | --- |
| **热键** | **用途** |
| F1 | 显示帮助 |
| F2 | 切换账号输入方式 |
| F3 | 继续查找 |
| F4 | 切换最大化状态（显示/不显示菜单） |
| F5 | 设置打印机 |
| F6 | 打印 |
| F7 | 调整表格 |
| F8 | 将焦点移到Grid里面。 |
| F9 | 根据菜单标题拼音查找菜单 |
| F10 | 焦点切换到菜单选择区域 |
| F11 | 搜索操作界面 |
| F12 | 显示向导 |
| ESC | 取消、清空数据、关闭窗口等等 |
| ALT+F2 | 出现特殊的客户选择界面（根据姓名、证件号码查询） |
| ALT+F4 | 关闭程序 |
| ALT+F10 | 最小化 |
| ALT+F11 | 注销 |
| ALT+F12 | 锁定屏幕 |
| CTRL+F2 | 在客户号输入控件里面，刷新客户的资金证券信息。 |
| CTRL+F3 | 查找 |
| CTRL+F4 | 关闭当前操作窗口 |
| ALT+CTRL+F1 | 恢复 |
|  |  |
| Alt+A | （Add）增加：增加一个项目 |
| Alt+B | 控制是否自动注销 |
| Alt+C | （Clear）清空：清空输入的数据，开始重新输入数据 |
| Alt+D | （Delete）删除：删除当前选中的条目 |
| Alt+E | 选择文件 |
| Alt+F |  |
| Alt+G | （Go）设置客户对应模版的信息。 |
| Alt+H |  |
| Alt+I | 列表框的选中项目上移（I在J的键盘上面） |
| Alt+J | 列表框的选中项目下移 |
| Alt+K | 复核 |
| Alt+L | 细节：浏览数据详细信息 |
| Alt+M | 撤单 |
| Alt+N | 增加时以选中行为默认值 |
| Alt+O | （OK）确认：确认操作 |
| Alt+P | （Print）打印 |
| Alt+Q | （Query）查询：查询/刷新数据 |
| Alt+R | （Refresh）刷新 |
| Alt+S | （Sum）合计 |
| Alt+T | 高级：高级功能 |
| Alt+U | （Update）修改：修改当前选中的条目 |
| Alt+V | 预览：预览打印效果 |
| Alt+W | 查询总部：查询总部设置的数据 |
| Alt+X | （eXit）退出、关闭 |
| Alt+Y | 自动刷新行情 |
| Alt+Z | 作废 |
| Alt+0 |  |
| Alt+1 | 用于在客户资料查询里面切换各个查询页 |
| Alt+2 | 用于在客户资料查询里面切换各个查询页 |
| Alt+3 | 用于在客户资料查询里面切换各个查询页 |
| Alt+4 | 用于在客户资料查询里面切换各个查询页 |
| Alt+5 | 用于在客户资料查询里面切换各个查询页 |
| Alt+6 | 用于在客户资料查询里面切换各个查询页 |
| Alt+7 | 用于在客户资料查询里面切换各个查询页 |
| Alt+8 | 用于在客户资料查询里面切换各个查询页 |
| Alt+9 | 用于在客户资料查询里面切换各个查询页 |
| CTRL+HOME | 到第一条数据 |
| CTRL+END | 到最后条数据 |
| CTRL+PgUp | 到上一条数据 |
| CTRL+PgDn | 到上一条数据 |
| CTRL+TAB | 在各个操作界面之间切换（最外面的PageControl） |
| CTRL+1 | 切换到上一个分页（针对操作界面里面的PageControl） |
| CTRL+2 | 切换到下一个分页（针对操作界面里面的PageControl） |

## 11、Tab顺序

基本的tab顺序是，先从上到下, 再从左到右。

在输入控件上面键入“Enter”自动切换到下一个控件。

# 七、界面标准和报表标准

## 基本界面规格

### 操作界面的大小

要求满足整个界面在“800×600”（比例4 ： 3）的界面上面显示出来，最大的操作区域的大小是：“621×457”，建议的最大大小是“608×456”（比例4 ：3）。其他大小的界面也应该保持4:3的长宽比。

下表列出推荐的界面大小。

|  |  |  |
| --- | --- | --- |
| **界面大小** | **宽度** | **高度** |
| 极限大小 | 621 | 457 |
| 最大大小 | 600 | 450 |
| 小型界面 | 400 | 300 |
| 中型 | 576 | 432 |
| 中型 | 480 | 360 |

### 颜色

|  |  |  |  |
| --- | --- | --- | --- |
| **颜色** | **取值** | **用途** | **备注** |
| 黑色 | clWindowText | 文字颜色 |  |
| 白色 | clWhite | 输入项目底色 |  |
| 浅黄色 | clInfoBk | 输入项目底色，表示输入项目不正确 |  |
| 浅蓝色1 | $00EBDED0 | 操作界面外围的背景色，按键背景色 |  |
| 浅蓝色2 | $00F6F3EF | 操作界面背景色 |  |
| 红色 | clRed | 特殊文字的颜色 | 尽量少用 |
| 灰色 | clBtnFace | 背景色 | 尽量少用 |
| 蓝色 | clBlue | 特殊文字的颜色 | 尽量少用 |

### 字体

|  |  |  |
| --- | --- | --- |
| **字体** | **用途** | **备注** |
| 宋体小五号 | 提示文字、输入文字文字 |  |
| 黑体小四 | 操作界面的标题文字 |  |
| 宋体粗体五号 | 按键文字 |  |

### 按键

|  |  |  |
| --- | --- | --- |
| **按键大小** | **说明** | **字体** |
| 75×25 | 标准按键 | 宋体五号粗体 |
| 60×20 | 标准小按键 | 宋体小五号 |
| 50×18 | 标准小按键 | 宋体小五号 |

### 标签（模版提示）

字体：宋体小五黑色。

高度：12。

宽度：如果左对齐，根据文字自动增加；如果右对齐，固定大小。

垂直间距：8（2/3×12＝8），那么Top的差正好是20！

水平间距：10

对齐：提示文字右对齐，内容文字左对齐。

### 标签（输入提示）

字体：宋体小五黑色。

高度：12。

宽度：固定大小。

垂直间距：18（3/2×12＝18），那么Top的差正好是30！

水平间距：10

对齐：右对齐。

(备注:如果标签旁边有编辑框或者下拉框则垂直边距为28,这样Top差为40

在各种界面中编辑框的Top根据实际情况定为:25/35/45,Left根据旁边的编辑框的Left来定(EditBox.Left-10-Label.Width),在实际操作中可以先确定编辑框的位置,然后用Center方式来对齐,必要时可以进行一些微调

### 编辑框

字体：宋体小五黑色。

高度：20。

宽度：100/140/400

垂直间距：20（1×20＝10），那么Top的差正好是40！

水平间距：

在各种界面中编辑框的Top根据实际情况定为:20/30/40,Left定为100/120/150

### 客户基本信息模版

宽度：230

top:20

Left:30

### 包含按键的Panel

高度：30

按键的Top：3。

**里面的按钮位置,大小**

按钮间距:10 最右面的按钮的右边距是:10

## 常用界面布局

### 1、参数设置界面

分为两页，第一页显示查询出来的参数。第二页的作用是：在增加和修改的时候录入数据；在删除和浏览数据的时候显示详细数据。

典型界面：营业部设置、数据字典维护、错误信息维护。

第一个编辑框的Top为20或者10或者5,每列的EditBox的间距为10,label和编辑框的间距为10或为5

第一个Lable的Top为25或者15或者10,Left为20或者10,每列Label的间距为10

### 2、查询界面

#### A、普通查询

分为两页，第一页输入查询条件，第二页显示查询结果。

典型界面：不同委托方式查询

如果是一列(不同委托方式查询)

第一个编辑框的Top为20,每列的EditBox的间距为20,label和编辑框的间距为10

Left为150

第一个Lable的Top为25,Left为(150-10-Lable.Width),每列Label的间距为20

注意同一列中即有编辑框和Label的情况(如当中要输入客户姓名的情况,间距要调整,总之要保证同一列的间距为20

如果是两列或多列(如委托明细查询)

第一列第一个编辑框的Top为20,每列的EditBox的间距为20,label和编辑框的间距为10

Left为120

第一列第一个Lable的Top为25,Left为(120-10-Lable.Width),每列Label的间距为20

第二列的编辑框Left为340,Label的Left为(340-10-Label.Width)

#### B、简单查询

数据条件少或者没有，如果有输入条件，输入条件在上面。查询结果表格占据整个操作界面的其余部分。如(交易汇总查询)

第一个编辑框的Top为20或者10或者5,每列的EditBox的间距为10,label和编辑框的间距为10或为5

第一个Lable的Top为25或者15或者10,,Left为20或者10,每列Label的间距为10

#### C、客户资料查询

在最上面输入客户号。然后在具体的分页里面输入其他查询条件，查询结果表格占据整个操作界面的其余部分。

典型界面：客户综合信息查询、客户档案查询|股东代码查询。

第一个编辑框的Top为20或者10或者5,每列的EditBox的间距为10,label和编辑框的间距为10或为5

第一个Lable的Top为25或者15或者10,,Left为20或者10,每列Label的间距为10

### 3、数据输入界面

#### A、最简单的形式

界面上面全部都是输入项目，没有专门的信息反馈区域。

典型界面：客户开户、职工修改操作密码。

如果分为两列显示的如(客户开户)

第一个编辑框的Top为10/20/40,Left为120每列的EditBox的间距为20,label和编辑框的间距为10

第一个Lable的Top为15/25/45,Left为(120-10-Label.Width),每列Label的间距为20

如果分为两列显示的如(职工修改操作密码)

如果有标题则标题的Top为20

第一个Lable的Top(如果有标题为60,没有为25),Left为(150-10-lable.width),每列Label的间距为20

第一个编辑框的Top(如果有标题为55,没有为20),Left为150,每列Label的间距为20

#### B、包含客户信息显示的简单输入界面

在界面的左边，有一个客户信息显示框。界面的右边是数据输入区域。

典型界面：主股东代码设置、指定股东代码席位。

右边的数据输入界面Label的Top:30,EditBox和ComboBox的Top:25

label和编辑框的间距为10(Label的Left:380-10-Label.Width)

编辑框和客户信息显示框的间距为120(EditBox的Left为380,客户信息显示框的Left为30,这样380-30-230=120)

#### C、资金类界面

界面上边显示，最近的资金存取信息；左边显示客户资料，右边是数据输入区域。

典型界面：现金存取。

界面上面的GBGrid不要随界面的放到而放大,宽度设为和界面一样大,高度为65(现金存取)/120(冲帐处理),Left和Top为0

下面的信息显示框的Left,Top以及数据录入框等参照简单输入界面

#### D、委托类界面

界面上边显示，最近的委托信息；左边显示客户资料，右上边是行情，右下边是数据输入区域。

典型界面：简单委托。

界面上面的GBGrid不要随界面的放到而放大,宽度设为和界面一样大,高度为200(如代理配股)/ 65(简单委托), Left和Top为0

下面的信息显示框的Left为0,宽度为220, Top为70

行情框的Left为225,宽度为400,Top为70

数据录入如果一列排的下的话,则编辑框的Left为340(340-220=120),这样编辑框和信息显示框的间距为120,Label和编辑框的间距为10(Label的Left=340-10-Label.Width)

如果为两列的话第一列的Left为300,第二列为500,每列的第一个编辑框和上面行情的间距为20或者为10,Lable和编辑框的间距为10

## 报表和回单规格

|  |  |  |
| --- | --- | --- |
| **文字部分** | **字体** | **行高度** |
| 大标题 | 宋体四号粗体 | 8 mm |
| 标题 | 宋体小四粗体 | 8 mm |
| 回单正文 | 宋体五号 | 7 mm |
| 列标题 | 宋体小五粗体 | 6 mm |
| 报表正文 | 宋体小五 | 4 mm |

注意行高在报表设计器里面band控件的“大小（0.1mm）”里面设置。因为单位是0.1mm，所以设置的行高值实际为80（回单）和60（报表）。

# 八、界面操作指南－基本的录入和显示

## 1、输入项目与显示项目

用户在界面上输入的数据（“输入项目”）是通过各种“输入控件”来录入数据的。

程序的显示的数据（“显示项目”）是通过各种“输出控件”来显示数据的。

输入项目一般在界面上面表现为白底黑字的编辑框。在编辑框里面可以输入各种文字。另外常见的输入控件还有下拉框、选择框等等。输入控件在输入正确的时候，底色是白色的；输入不正确的时候，底色是浅黄色的。文字的颜色一般都是黑色。

显示项目一般在界面上面表现为表格框或者标签，底色和背景色相同，文字的颜色一般是黑色，特殊提示的使用红色或者蓝色。

## 2、各种输入控件的基本使用方法

常见的输入控件包括：

1）编辑框

2）下拉框

3）选择框

4）列表框

5）列表选择框

## 3、各种输出控件的基本使用方法

常见的输出控件包括：

1）标签

2）表格框

## 4、输入顺序和反馈信息

程序里面应该根据习惯正确设置输入顺序，输入顺序应该和输入控件在屏幕的位置对应。一般输入顺序是先从上到下, 再从左到右。

在大部分输入控件上面支持“Enter”键和“\*”键切换到下一个或者上一个输入项目。具体的说，当对应的数据录入正确的时候，在输入控件上面按下“Enter”键，程序自动切换到下一个输入项目。不管是否输入正确，在输入控件上面按下“\*”都可以切换到上一个输入项目。另外有少数输入控件，不支持使用“Enter”键和“\*”切换输入项目，这时候应该使用操作系统标准的“tab”和“shift+tab”改变输入项目。

输入控件在输入正确的时候，底色是白色的；输入不正确的时候，底色是浅黄色的。没有正确输入的项目的名称会显示在状态条上面。

程序不限制用户实际的输入的顺序。也就是说，可以按照任意的输入顺序录入数据。仅在输入项目正确输入以后，对应的反馈信息才会显示出来。例如正确输入客户号以后，显示客户基本信息。市场和证券代码都正确输入以后，显示行情。修改市场或者证券代码其中任意一项，会更新行情显示。如果对应的证券不存在（指定的市场和证券代码），清空行情显示。

只有全部项目正确输入以后，“确认”按键才有效。

注意：当一个项目没有正确输入的时候，在对应输入控件上面按下“Enter”键，发出报警声，不自动切换到下一个输入项目。就是说，输入焦点保持在这个输入不正确的控件上面。但是不限制使用Tab/Shift Tab以及使用鼠标改变焦点。

## 5、对输入不正确的数据项目的处理

当一个项目没有正确输入的时候：

1）输入控件的底色是黄色；

2）在状态条上面会显示该项目的名称；

3）按下“Enter”键，发出报警声，输入焦点保持在这个输入不正确的项目上面，不自动切换到下一个输入项目。

4）对于一些特定项目的输入，例如客户号，按下“Enter”键，如果客户不存在，将会弹出错误框，提示错误信息，输入焦点保持在客户号输入控件上面。

只有当操作界面上面所有项目正确输入以后，“确认”按键才有效。

## 6、状态提示

在主界面的状态条上显示与当前操作界面相关的操作提示。

第一个是针对当前正在输入的项目的提示。该提示的内容一般是描述在该输入项目里面正确输入的数据以及格式应该是什么样的。

第二个是红色文字，列出在当前界面上面还没有正确输入的项目的名称。如果有多个项目，项目名称之间使用逗号分割。

## 7、数据显示

除了针对输入项目的及时反馈信息以外，其他的显示数据，都是通过用户按下界面上面的对应的功能按键，然后程序从数据库获取数据，最后显示在输出控件上面。常见的功能按键包括：

“确定”：确定当前输入的数据，完成该界面对应的操作。

“查询”：根据输入的查询条件，查询数据。

“合计”：计算合计数据。

注意：必须通过用户明确的按键操作，才触发操作。

# 九、界面操作指南－浏览数据

# 第二部分 程序核心设计部分

# 十、核心设计概述

## 1、目标和原则

### I、界面处理和业务处理的分离

用户界面是与用户直接交互的各种窗口、表格、报表等等。为了美观和易用，在开发过程中和产品的生存周期中都会发生不断的变化要求。

业务处理是指与证券业务相关的处理，例如委托、客户开户等等。业务处理的大部分内容本质上是很少改变的，不同版本的后台程序只是在接口上有所区别（例如存储过程的名称不同、参数的名称和数据类型不同等等）。

将界面处理和业务处理分隔开来，定义一个清晰的接口，有助于两部分各自独立的在实现上面的变化。

### II、数据访问接口和实现的分离

数据访问的形式可能包括：使用数据库提供者的专用接口库（例如DBLIB、OCI）、使用通用接口（ODBC、BDE、ADO）、使用专用的网关/应用服务器接口。具体使用什么访问形式由具体应用决定。

为了适应这种改变，需要一个统一的简单接口，将数据访问的具体实现封装起来。

### III、应用程序按照业务功能分解为若干模块

应用程序分解成模块以后，便于组织独立的开发。

模块应该可以方便的加载和卸载。

模块之间应该是松耦合的。

### IV、录入界面中的控件的事件响应和处理分离

## 2、基本结构

通讯程序

应用5

应用4

应用3

应用2

应用1

存储过程

RS6000

DB2

数据访问层

应用逻辑层

界面层

前端

数据访问层

应用逻辑层

界面层

前端

数据访问层

应用逻辑层

界面层

前端

前台程序分为三层：

1)界面层包括和用户交互的界面和相应的基本输入检查。界面层可能是多变的，以适应不同用户的需求。

2) 应用逻辑层包含证券业务的实现，为界面提供业务处理的接口。也包括一些复杂的。因为目前主要的业务处理由存储过程和应用服务器的应用完成，所以前端程序应用逻辑层主要作用是封装应用接口、提供输入数据检查、处理错误等等。

3) 数据访问层提供访问数据的基础对象，是访问应用服务器的接口。该层还提供数据的本地缓存，提供双向（只读）的游标（cursor）。

体现了“界面处理和业务处理的分离原则”和“数据访问接口和实现的分离原则”。

## 3、以数据为中心的结构

数据

数据存取和处理

数据的录入

数据的表示

表格

统计图

报表

录入控制

流程控制

### I、数据存取和处理

分为三个层次。从下到上依次是：

1. 抽象的数据访问接口，以及这个接口的一个实例(instance：根据具体的访问方法决定)。这个接口简单并且通用的。简单要求便于使用和实现。通用要求能够照顾到大多数的数据访问形式。这个接口的功能提供以下基本功能：数据源连接、基于SQL文本的命令访问、类似存储过程的RPC命令访问、读取返回的结果集（支持多结果集、单向游标）。
2. 在抽象的数据访问接口基础上实现的Dataset的子类。实现本地数据缓存（双向游标）、简化特定数据源的访问方式。
3. 数据对象层，利用面向对象的方法组织数据和业务功能。对象主要分为两类，实体对象(entity objects)代表实体业务对象，例如客户、账号、证券等等，用于获取这些实体的基本信息，例如客户姓名、资金状况等等。业务对象(business objects)，完成具体的业务操作，例如委托操作等等。

第1、2层属于物理层，第3层属于逻辑层。1、2层分离，便于实现对各种数据源访问的支持。第3层封装了后台的处理，真正分离了前后台。对象的划分和组织是整个前台程序的关键。

### II、数据的录入

数据录入要求使用方便，尽量减少用户的操作错误。一方面及时向用户提供反馈信息（例如输入客户号以后显示姓名和资金），另一方面对重要的操作要求用户确认。在处理流程上，一定要把挂在控件上面的事件处理和实际的流程控制分离开来。把流程控制、数据有效性检验等等综合起来，集中编码。改变过去错综复杂的界面程序编写。

具体的讲，就是把要录入的数据和录入数据的控件分离开，每个界面程序只编写针对录入数据的处理流程，把对控件的事件处理交给下层的组件完成。那么即使以后更换了录入数据的具体控件，也可以不改动程序。同时程序编写的时候，也可以不关心具体的控件名称。

另外数据的录入还包括各种特殊的录入控件，如下：

|  |  |
| --- | --- |
| **录入内容** |  |
| 金额 |  |
| 数量 |  |
| 客户号 |  |
| 委托账号（证券账号） |  |

### III、数据的表示

数据以各种形式展现给用户，例如表格形式、统计图形式、打印的报表等等。

表格形式要求支持表格尾部的汇总、左边的固定列、特定数据行的图书显示（红色表示XXX等等）。

统计图能够在一张图上同时显示多种格式的统计图，例如二维坐标图、饼图、柱装图等等。可以方便的设置各种标签文字。对于图中各种成分可以方便的修改颜色字体等等。详细内容见统计图组件分析。

打印的报表要求能够很好的支持大多数的打印机，可以方便的设置打印格式。详细的要求见报表组件分析。

## 4、程序模块的组织

### I、使用程序包(package)的形式组织程序模块

程序包(package)是一种高级的动态连接库(DLL)，可以输出对象。除了Delphi携带的标准VCL包以外，还包括以下

A、基础程序包：

1）数据访问包：实现数据访问层的组件包。

2）界面组件包：实现各种界面功能的组件包。

3）报表组件包

4）统计图组件包

5）其他与业务无关的通用程序包

B、一个包含整个程序框架的总控程序包，用于主界面框架的实现、菜单管理、程序包管理、全局在线帮助、全局错误处理等等。

C、与业务相关的程序包：

1. 包含基本的实体对象的程序包
2. 客户管理：开销户、存取钱。
3. 委托业务：委托、撤单、批量委托、申购等等。
4. 清算业务
5. 基本的系统管理业务：权限管理、系统参数设定。
6. 实时监控
7. 统计查询
8. 经纪人

**汇总表**

|  |  |  |
| --- | --- | --- |
| **程序包类型** | **装载形式** | **备注** |
| 标准VCL包 | 启动的时候装载 |  |
| 基础程序包 | 启动的时候装载 |  |
| 包含整个程序框架的总控程序包 | 启动的时候装载 |  |
| 包含基本实体对象和实现基本功能的程序包 | 启动的时候装载 |  |
| 其他业务程序包 | 使用的时候装载 |  |
|  |  |  |

业务程序包分为2种，分别包含操作界面和业务实现。操作界面程序包包含用户使用的操作界面的实现。业务实现程序包完成实际业务的处理。2种包之间通过命令模式的进行松耦合（参见“七、使用命令模型分隔界面处理和业务实现”）。操作界面创建命令对象，将用户在界面输入的数据填充到命令对象中，然后传输命令对象。命令对象被传输到业务实现程序包对应的处理程序上面，完成处理。处理以后的对象被传输回操作界面，操作界面的处理程序将处理结果显示给用户。权限控制、审计功能由命令对象传输程序集中处理。

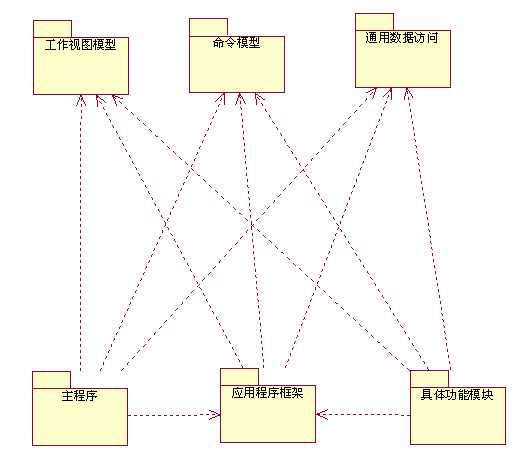
多个相关的业务程序包组成一个程序组。程序组中的程序包被同时装载/卸载。一般在一个程序组中包含一个操作界面程序包和一个对应的业务实现程序包。

### II、菜单的管理

每一条菜单有一个显示文字和ID，对应一个操作界面。缺省情况下ID和显示文字相同。主程序启动的时候，根据菜单文件，获取整个菜单的结构，并且生成相应的菜单树状结构。然后主程序根据配置文件加载程序包。包含操作界面（对应于菜单的实现）的程序包加载的时候，自动注册支持的特定菜单ID的接口实例(instance)。用户登录以后，根据权限隐藏不可操作的菜单。用户点击菜单以后，主程序根据注册的菜单信息，调用相应的操作界面程序。操作界面与主程序之间是松耦合的。

# 十一、逻辑视图

该视图描述了柜台程序的逻辑结构，里面包含了各个对象之间的静态和动态关系。



整个结构分为三个层次，六个部分。最下层是“工作视图模型”、“命令模型”、“通用数据访问”三个部分。中层是“应用程序框架”。上层是“主程序”和“具体功能模块”两个部分。

下面依次介绍各个部分。

## 1工作视图模型

### 引入工作视图模型的原因

在常见的管理信息系统（MIS）中，大部分的界面都是将数据录入到数据库或者将数据库的数据查询出来。在图形化用户界面（GUI）里面，程序是根据消息驱动（即用户的各种操作，例如在编辑框里面输入数据、按下操作按键等等）的，而这些消息的顺序在实践中是不规则的，那么按照传统的控制流的顺序（例如先输入XXX，再输入YYY的顺序）进行编码（Programming）的方法难以适用。

因为在这些界面里面处于核心地位的是数据，所以应该采取以数据为中心，根据数据流驱动的方法进行编码。

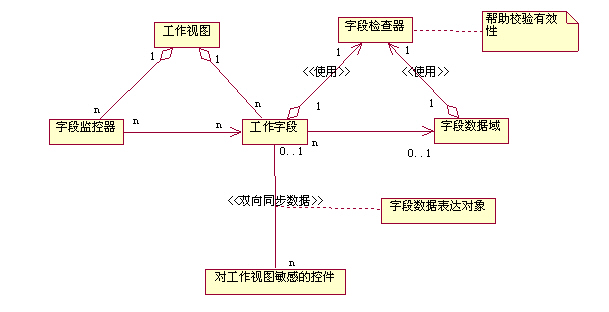
数据流和控制流的在处理方式上面有根本的区别。

控制流的顺序是：先做AAA，如果AAA正确完成了，接着做BBB，否则重新做AAA；等到BBB完成以后，最后做CCC。

数据流的顺序是：如果数据D1准备好了，那么做AAA，产生D3；如果数据D2准备好了，那么做BBB，产生D4；如果D3和D4都准备好了，那么做CCC。

数据流的核心是各种数据项目（例如D1、D2等等）和这些数据项目的状态（例如D1准备好了）。**工作视图**就是一种描述数据流的计算机编码模型。

### 静态结构



各个对象介绍如下。

### 工作字段

工作字段是描述单个数据项目及其状态的计算机编码模型。一个工作字段代表一个数据项目的对象。工作字段包含的语意有：

1、名称(Name具有唯一性，用来标志一个字段)和标题(Caption显示的时候使用)。

2、数据的数据类型(DataType)：整数、浮点数、对象(结果集以及其他)等等。

3、数据的当前值(在Data属性里面，是TKSDataObject对象)和缺省值(DefaultValue)。

4、数据的约束(有效性控制)(Constrained、Valid、Checker、DomainName等属性)

5、文字性的补充描述(Description、Notes)

6、处理基本的键盘输入，允许Enter(GoNextWhenPressEnter)切换到下一个输入控件、“\*”或者其他特殊按键(GetPrevChar)切换到上一个输入控件

7、将几个相关的工作字段编组(GroupIndex)。

8、和数据源的字段(TField)绑定关系(DataField)。

9、提供输入提示(Hint和ErrorMessage)。

10、Script和Features提供了额外的扩充功能。

11、在值或者有效性发生的时候触发事件。

其中1～4是核心的语意，代表了一个数据项目的名称（用于区别其他数据项目）、数据类型、数据的当前值、如何判断数据项目是否有效（即数据准备好了）等关键属性。

当工作字段的当前值发生改变或者有效性发生改变的时候，都会触发事件（Event），程序员可以在这些事件里面编写代码。这种编码方式就是数据流驱动的方式，操作处理是根据数据的改变或者数据就绪状态的改变而触发的。

**实现**：工作字段的声明和实现是“**WorkView.pas**”里面的“**TWVField**”对象。

**实现**：工作字段对象可以通过工作视图对象的“**WorkFields**”或者“**Fields**”属性获得，或者根据工作视图对象的“**FieldByName**”或者“**FindField**”方法获得。

**实现**：工作字段对象的值保存在**TKSDataObject**对象里面。**TKSDataObject**对象可以代表各自类型的数据。**TKSDataObject**对象的定义和实现在“**DataTypes.pas**”里面。

**程序员视点**：

1）工作字段的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | Name | string | 名称，唯一标识工作字段 |
| 属性 | Caption | string | 标题 |
| 属性 | FieldType | TWorkFieldType | 字段类型 |
| 属性 | DomainName | string | 字段数据域的名称 |
| 属性 | Features | TStrings | 字段的特性，留作扩充 |
| 属性 | Description | string | 字段的描述，留作扩充 |
| 属性 | Hint | string | 提示，用于各种和该字段相关的控件。 |
| 属性 | Notes | string | 注释，留作扩充 |
| 属性 | Script | string | 留作扩充 |
| 属性 | DataType | TWFDataType | 数据类型 |
| 属性 | OwnObject | Boolean | 如果数据类型是对象类型，该属性确定是否持有对象（负责对象的释放）。 |
| 属性 | MonitorValueChangedFields | string | 监控其他字段的值的改变。属性是需要监控的字段名称，名称之间用“|”分割。 |
| 属性 | MonitorValidChangedFields | string | 监控其他字段的有效性的改变。属性是需要监控的字段名称，名称之间用“|”分割。 |
| 属性 | Visible | boolean | 是否控件，留作扩充 |
| 属性 | Constrained | Boolean | 对字段的值的有效性是否进行约束 |
| 属性 | Checker | TWVFieldChecker | 对字段有效性的具体规定 |
| 属性 | DefaultValue | Variant | 字段的缺省值 |
| 属性 | GoNextWhenPressEnter | Boolean | 在对应的输入控件里面，按下“Enter”是否自动将输入焦点下移。 |
| 属性 | GetPrevChar | Char | 在对应的输入控件里面，按下该字符，自动将输入焦点上移。#0表示无此特性。 |
| 属性 | GroupIndex | Integer | 字段分组值。0表示不分组 |
| 属性 | DataField | string | 对应的数据集字段的名字。 |
| 事件 | OnUpdateValue |  |  |
| 事件 | OnCheckValid |  | 当需要校验有效性的时候触发 |
| 事件 | OnValidChanged |  | 当有效性改变的时候触发 |
| 事件 | OnValueChanged |  | 当值改变的时候触发 |
| 事件 | OnKeyDown |  | 当对应的控件里面按键时触发 |
| 事件 | OnKeyPress |  | 当对应的控件里面按键时触发 |
| 事件 | OnKeyUp |  | 当对应的控件里面按键时触发 |
| 事件 | OnInvalidInput |  | 当用户输入无效数据的时候触发 |

2）工作字段的运行时候的属性和方法

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 方法 | CheckValid |  | 校验字段的有效性 |
| 方法 | Reset |  | 复位字段的值（恢复到缺省值） |
| 方法 | GetMaxLength | Integer | 可以输入的字符的最大长度 |
| 方法 | GetHint | string | 返回提示 |
| 方法 | SetInvalidMessage |  | 当数据无效的时候，设置错误信息 |
| 属性 | Valid | Boolean | 字段的有效性（是否有效）。 |
| 属性 | Domain | TWVFieldDomain | 对应的字段数据域对象 |
| 属性 | WorkView | TWorkView | 对应的工作视图对象 |
| 属性 | ErrorMessage | string | 错误信息 |
| 属性 | Data | TKSDataObject | 数据对象，里面保存实际的数据 |
| 属性 | AutoGoNext | Boolean | 确定在这次输入中，按下Enter是否自动到下一个控件 |

3）**TKSDataObject**对象的属性和方法

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ReadOnly | Boolean | 数据是否是只读的 |
| 属性 | DataType | TKSDataType | 数据类型 |
| 属性 | IsEmpty | Boolean | 数据是否为空（没有赋值） |
| 方法 | Clear |  | 清空当前数据 |
| 属性 | OwnObject | Boolean | 是否持有对象 |
| 方法 | SetInteger  SetInt64  SetString  SetWideString  SetChar  SetWideChar  SetDatetime  SetBoolean  SetObject  SetClass  SetPointer  SetFloat  SetCurrency  SetInterface |  | 设置某种类型的数据。数据类型必须和DataType里面的数据匹配 |
| 属性 | AsString  AsInteger  AsFloat  AsCurrency  AsBoolean  AsObject  AsInterface |  | 返回特定类型的数据。完成必要的数据转换。  其中AsString是可以修改的属性 |
| 属性 | Value | Variant | Variant类型值的表现 |
| 事件 | OnChanged | 值发生改变的时候调用 |  |
| 事件 | OnException | 产生意外的时候被调用 |  |

### 工作视图

一个工作视图里面包含了一个操作界面需要用到的所有工作字段。“视图”的意义在这里是“场景”的意思，工作视图代表了操作界面需要涉及到的所有数据项目。

**程序员视点**：从程序员编码的角度说，工作视图是工作字段的容器，负责管理工作字段。工作视图还是字段监控器的容器，有关字段监控器见下文。

工作视图还提供其他一些统一的控制，例如对应有效输入数据的编辑框的底色、对应无效输入数据的编辑框的底色等等。另外工作视图还统一处理在工作字段的处理当中发生的意外。

**实现**：工作字段的声明和实现是“**WorkView.pas**”里面的“**TWorkView**”对象。

**程序员视点**：

1）工作视图的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | WorkName | string | 名字，留作扩充 |
| 属性 | Description | string | 描述，留作扩充 |
| 属性 | Notes | string | 注释，留作扩充 |
| 属性 | WorkFields | TCollection | 包含的工作字段 |
| 属性 | FieldsMonitors | TCollection | 包含的字段监控器 |
| 属性 | ValidColor | TColor | 对应有效输入数据的编辑框的底色 |
| 属性 | InvalidColor | TColor | 对应无效输入数据的编辑框的底色 |
| 属性 | ReadOnlyColor | TColor | 只读数据的底色 |
| 属性 | ReserveNGroups | Boolean | 是否在Reset的时候保护GroupIndex<0的字段。 |
| 事件 | OnException |  | 当发生意外的时候被触发 |
| 事件 | OnInvalidInput |  | 当输入数据无效的时候被触发 |

2）工作视图的运行时候的属性和方法

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 方法 | FieldByName |  | 根据名字查找字段，如果找不到，抛出意外。 |
| 方法 | FindField |  | 根据名字查找字段，如果找不到，返回nil。 |
| 方法 | Reset |  | 将一组或者所有工作字段复位 |
| 方法 | SynchronizeCtrlsToFields |  | 将控件的值同步到字段 |
| 方法 | CheckValid |  | 检查一组或者所有工作字段的有效性 |
| 方法 | LoadFromDataset |  | 从标准数据集读取数据 |
| 方法 | SaveToDataset |  | 向标准数据集保存数据 |
| 方法 | SelectPrev |  | 输入焦点移到到上一个控件 |
| 方法 | SelectNext |  | 输入焦点移到到下一个控件 |
| 属性 | Fields | TWVField | 根据索引获得工作字段 |
| 属性 | FieldCount | Integer | 工作字段数目 |

### 字段监控器

字段监控器的目的是同时监控多个工作字段的变化（当前值的改变或者有效性的改变）。当字段监控器监控的任何一个工作字段发生改变，字段监控器的事件都被触发。程序员可以对这些事件进行编码。

**程序员视点**：从程序员编码的角度说，字段监控器就是为了方便数据流驱动方式的编码。

**实现**：字段监控器的声明和实现是“**WorkView.pas**”里面的“**TWVFieldMonitor**”对象。

**实现**：字段监控器可以通过工作视图对象的“**FieldsMonitors**”属性获得。

**程序员视点**：工作视图的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | Description | string | 描述 |
| 属性 | MonitorValueChangedFields | string | 监控其他字段的值的改变。属性是需要监控的字段名称，名称之间用"|"分割。 |
| 属性 | MonitorValidChangedFields | string | 监控其他字段的有效性的改变。属性是需要监控的字段名称，名称之间用"|"分割。 |
| 属性 | GroupIndex | Integer | 字段的分组 |
| 事件 | OnValueChanged |  | 当监控的任何一个字段的有效性改变的时候触发 |
| 事件 | OnValidChanged |  | 当监控的任何一个字段的值改变的时候触发 |

### 对工作视图敏感的控件

对工作视图敏感的控件是指一类可以自动和工作视图对象协作的输入/输出控件。一般一个控件对应一个工作字段，一个工作字段可以对应多个控件。在控件里面输入的数据可以自动更新到工作字段里面。当工作字段的数据发生改变的时候，也可以自动显示在控件里面。

**程序员视点**：工作字段和控件之间的关系是典型的“模型－视图－控制（MVC）”设计模式（Design Pattern）。工作字段是模型代表了数据。控件是视图，负责表现模型里面的数据。控制是内建在工作字段和控件之间，自动完成工作字段和控件之间双向同步。

**实现**：对工作视图敏感的控件都在“**WVCtrls.pas**”里面。

|  |  |
| --- | --- |
| **控件** | **用途** |
| TWVEdit | 编辑框 |
| **TWVComboBox** | 下拉框 |
| **TWVLabel** | 标签 |
| **TWVCheckBox** | 选择框 |
| **TWVCheckListBox** | 多选列表框 |
| **TWVDigitalEdit** | 数字输入框 |
|  |  |

### 字段检查器

字段检查器是一个辅助对象，用于工作字段数据有效性检查。另外还可以控制允许输入的字符的类型。

**程序员视点**：属性和事件列表

|  |  |  |
| --- | --- | --- |
| **名称** | **类别** | **用途** |
| MinLength | 属性 | 如果工作字段的数据类型是字符串，那么该属性规定了字符串的最小长度。 |
| **MaxLength** | 属性 | 如果工作字段的数据类型是字符串，那么该属性规定了字符串的最大长度。 |
| **Min** | 属性 | 如果工作字段的数据类型是数字类型，那么该属性规定了数值的最小值。 |
| **Max** | 属性 | 如果工作字段的数据类型是数字类型，那么该属性规定了数值的最大值。 |
| **DefaultValid** | 属性 | 缺省是否有效 |
| **AcceptDigital** | 属性 | 是否允许输入数字 |
| **AcceptAlphabet** | 属性 | 是否允许输入字母 |
| **AcceptOther** | 属性 | 是否允许输入各种符号 |
| **AcceptHigh** | 属性 | 是否允许输入汉字 |
| **LowerCase** | 属性 | 是否字母都用小写 |
| **UpperCase** | 属性 | 是否字母都用大写 |
| **Required** | 属性 | 工作字段是否必须输入 |
| **OnCheckValid** | 事件 | 当检查字段有效性的时候被触发。在这里编码可以通过代码的形式处理复杂的有效性判断。 |
| **OnKeyDown** | 事件 | 当对应的输入控件有按键按下的时候被触发 |
| **OnKeyPress** | 事件 | 当对应的输入控件有按键按下的时候被触发 |
| **OnKeyUp** | 事件 | 当对应的输入控件有按键按下的时候被触发 |
|  |  |  |

**实现**：字段检查器的声明和实现是“**WorkView.pas**”里面的“**TWVFieldChecker**”对象。

**实现**：字段检查器可以通过工作字段对象的“**Checker**”属性或者字段数据域对象的“**Checker**”属性获得。

**实现**：工作字段和字段数据域的几个事件（例如**OnCheckValid**）都是定向到字段检查器同名事件上面的。

### 字段数据域

在不同界面出现的一些数据项目，往往代表相似的含义，具有相似的性质。例如在客户开户界面出现的证件号码和股东开户界面出现的证件号码，是相似的数据类型，它们允许输入的字符和号码的长度是相同的。为了表达这种相似性，简化工作字段有效性的判断，提出字段数据域对象的概念。

一个字段数据域对象代表一类相似的数据项目，字段数据域对象负责对这些数据项目的有效性进行统一的校验。一个字段数据域对象具备一个唯一的域名，用于和其他字段数据域对象区别。工作字段对象通过自己的属性“**DomainName**”（字符串类型），查找到合适的字段数据域对象，然后将有效性检查委托给字段数据域对象处理。

**实现**：字段数据域的声明和实现是“**WorkView.pas**”里面的“**TWVFieldDomain**”对象。

**实现**：一个字段数据域对象包含一个字段检查器对象，借助该字段检查器对象提供的功能，完成具体的字段有效性校验。

**程序员视点**：字段数据域的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | DomainName | string | 名称，唯一标识 |
| 属性 | Checker | TWVFieldChecker | 字段检查器 |
| 属性 | Hint | string | 提示 |
| 事件 | OnCheckValid |  | 检查字段有效性的时候被触发 |
| 事件 | OnKeyDown |  | 当对应的输入控件有按键按下的时候被触发 |
| 事件 | OnKeyPress |  | 当对应的输入控件有按键按下的时候被触发 |
| 事件 | OnKeyUp |  | 当对应的输入控件有按键按下的时候被触发 |

### 校验工作字段的有效性

当工作字段的属性“**Constrained**”设置True的时候，表示工作字段的值的有效性有一定的约束，需要对字段的有效性进行校验。校验的顺序如下：

A）如果设置了字段的**DomainName**，而且对应的字段数据域存在：

1）根据字段数据域的字段检查器对象（**Checker**）的属性，进行有效性校验；

2）然后调用字段数据域的的**OnCheckValid**事件，完成补充的校验；

3）最后调用字段本身的**OnCheckValid**事件，完成补充的校验。

B）没有对应的字段数据域：

1）根据字段的字段检查器对象（**Checker）**的属性，进行有效性校验；

2）然后调用字段本身的**OnCheckValid**事件，完成补充的校验。

检查器对象判断有效性的方法是：

1)首先设置**Valid**=**DefaultValid**

2)如果字段的值为空(**Field**.**Data**.**IsEmpty**)，**Valid**=**not** **Required**

3)否则

a)对于字符串，检查长度是否合适(如果**MaxLength**>0)；

b)对于数字，检查数值范围(如果**Max**-**Min**>0)。

### 字段数据表达对象

一些特殊的数据项目，在表现的时候，不能使用直接的方式。例如一个代表性别的数据项目，在内部用“M”和“W”分别代表男女，在外部提供给操作者使用的时候，需要显示出“男女”。就是说数据项目的表达形式和普通的直接映射不相同。这个时候需要用到字段数据表达对象。

字段数据表达对象负责工作字段的数据在控件里面表现形式和控件里面的输入如何同步回到工作字段，即字段数据表达对象控制工作字段的数据和控件里面的数据之间的双向同步。在同步的时候，完成必要的数据转换工作。

**程序员视点**：属性和事件列表

|  |  |  |
| --- | --- | --- |
| **名称** | **类别** | **用途** |
| DataPresentType | 属性 | 数据表达形式的名称，用于区分字段数据表达对象 |
| **OnCtrlToField** | 事件 | 当将数据从控件同步到字段的时候被触发 |
| **OnFieldToCtrl** | 事件 | 当将数据从字段同步到的时候被触发 |
| **OnSetFieldHandler** | 事件 | 为数据集的字段对象（TField）设置特殊的事件处理器 |

**实现**：字段数据表达对象的声明和实现是“**WorkView.pas**”里面的“**TWVCustomFieldPresent**”和“**TWVFieldPresent**”对象。

**实现**：对工作视图敏感的控件一般具有两个属性“**DataPresentType**”和“**DataPresentParam**”。“**DataPresentType**”标志该控件对应的字段数据表达对象的名字，“**DataPresentParam**”提供额外的参数。

### 结果集绑定对象

工作字段对象可以包含的数据类型比较多样，特别的，工作字段的数据类型可以是对象类型（**Object**）。程序员可以在工作字段的里面保存另外一个对象。在集中交易柜台系统里面最常见的方式是在工作字段里面保存标准的数据集对象（**TDataset**）。

结果集绑定对象的功能是将工作字段里面的数据集对象传递给数据源对象（**TDataSource**）,使得和数据源相连的数据敏感控件（例如**TDBGrid**）可以显示出结果集里面的数据。

**程序员视点**：结果集绑定对象的属性和事件列表

|  |  |  |
| --- | --- | --- |
| **名称** | **类别** | **用途** |
| DataSource | 属性 | 关联的数据源对象。 |
| WorkView | 属性 | 关联的工作视图对象。 |
| FieldName | 属性 | 包含数据集的工作字段的名称 |
| Bindings | 属性 | 包含“数据字段绑定项目对象”的集合 |
| HideUnbindingFields | 属性 | 是否只显示在“数据字段绑定项目对象”的集合里面定义的字段 |
| OnBinding | 事件 | 将数据集从工作字段传递给数据源的时候触发 |

结果集绑定对象还可以控制结果集对象（**TDataset**）的数据字段（**TField**）的显示特征。具体实现的方法是，结果集绑定对象包含许多“数据字段绑定项目对象”。每个数据字段绑定项目对象对应一个数据字段，用于设置这个数据字段的显示属性。

**程序员视点**：数据字段绑定项目对象的属性和事件列表

|  |  |  |
| --- | --- | --- |
| **名称** | **类别** | **用途** |
| FieldName | 属性 | 数据集字段名称 |
| WVFieldName | 属性 | 工作字段名称 |
| DataPresentType | 属性 | 字段数据表达对象的名字 |
| DataPresentParam | 属性 | 字段数据表达对象的需要的额外的参数 |
| OnBinding | 事件 | 绑定数据集字段的时候被触发 |

**实现**：结果集绑定对象的声明和实现是“**WVDBBinders.pas**”里面的“**TWVDBBinder**”对象。

**实现**：数据字段绑定项目对象的声明和实现是“**WVDBBinders.pas**”里面的“**TWVDBBinding**”对象。

**实现**：数据字段绑定项目对象利用到了字段数据表达对象的“**OnSetFieldHandler**”事件。

### 工作视图中心对象

工作视图中心对象的目的是管理各个操作界面上面的工作视图对象。一个应用程序程序应该只有一个工作视图中心对象。

工作视图中心对象可以统一控制工作视图的一些属性，例如对应有效输入数据的编辑框的底色、对应无效输入数据的编辑框的底色等等。

**程序员视点**：工作视图中心对象设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ValidColor | TColor | 对应有效输入数据的编辑框的底色 |
| 属性 | InvalidColor | TColor | 对应无效输入数据的编辑框的底色 |
| 属性 | ReadOnlyColor | TColor | 只读数据的底色 |
| 事件 | OnException |  | 当发生意外的时候被触发 |
| 事件 | OnInvalidInput |  | 当输入数据无效的时候被触发 |

### 小结

工作视图模型的核心是以数据为中心，按照数据流的组织程序。

## 2命令模型

### 引入命令模型的原因

一个应用程序的界面设计和功能实现在实践中往往各自发生变化。例如，为了美观和实用，调整界面的布局。更换界面上面的各种控件，改变数据的输入/输出发生。以上这些是界面部分的变化。为了增强程序的安全性，在实现里面增加权限的判断或者增加记录操作日志。以上这些是功能实现部分的变化。

为了将界面设计和功能实现分割开来，允许两者分别变化，尽量互相不影响。所以引入了命令模型。命令模型的目的就是实现界面和实现两者的分离。

### 静态模型



每个对象的具体描述见下。

### 命令对象和命令描述对象

一个命令对象代表了一个操作。命令对象有一个标识符（ID），表明命令的种类。命令对象有一定的命令参数。每个命令参数有自己的参数名称、参数数据类型、参数类别（输入/输出/输入并且输出）、当前值等等属性。即一个命令对象封装了一个操作的所有输入/输出数据，并且通过命令ID标识出需要进行的操作。例如一个代表现金存取的命令对象，在该对象被处理以前，它的属性如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | 现金存取 | | |
| **参数名称** | **参数数据类型** | **参数类别** | **当前值** |
| 客户号 | 字符串 | 输入 | 12001234 |
| 币种 | 字符串 | 输入 | 1 |
| 存取 | 整数 | 输入 | 1 |
| 金额 | 浮点数 | 输入 | 110.12 |
| 操作员 | 字符串 | 输入 | 012888 |
| 流水号 | 字符串 | 输出 | <空> |
| 返回码 | 整数 | 输出 | <空> |
| 返回信息 | 字符串 | 输出 | <空> |

在该命令被成功处理以后，它的属性如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | 现金存取 | | |
| **参数名称** | **参数数据类型** | **参数类别** | **当前值** |
| 客户号 | 字符串 | 输入 | 12001234 |
| 币种 | 字符串 | 输入 | 1 |
| 存取 | 整数 | 输入 | 1 |
| 金额 | 浮点数 | 输入 | 110.12 |
| 操作员 | 字符串 | 输入 | 012888 |
| 流水号 | 字符串 | 输出 | 200112120000101 |
| 返回码 | 整数 | 输出 | 0 |
| 返回信息 | 字符串 | 输出 |  |

一般而言，命令ID相同的命令对象，除了具体的参数值不同以外，它们的参数名称、参数数据类型、参数类别都是相同的。我们可以用命令描述对象来描述这一类相似的命令对象。

命令描述对象的ID和命令对象的ID相同，唯一的标识需要进行的操作处理的内容。命令描述对象描述了一类操作处理所需要的所有参数的性质。

例如现金存取对应的命令描述对象如下：

|  |  |  |
| --- | --- | --- |
| **ID** | 现金存取 | |
| **参数名称** | **参数数据类型** | **参数类别** |
| 客户号 | 字符串 | 输入 |
| 币种 | 字符串 | 输入 |
| 存取 | 整数 | 输入 |
| 金额 | 浮点数 | 输入 |
| 操作员 | 字符串 | 输入 |
| 流水号 | 字符串 | 输出 |
| 返回码 | 整数 | 输出 |
| 返回信息 | 字符串 | 输出 |

**实现**：命令对象的声明和实现是“**WVCommands.pas**”里面的“**TWVCommand**”对象。

**实现**：命令描述对象的声明和实现是“**WVCommands.pas**”里面的“**TWVCommandDescriptor**”对象以及“**WVCmdTypeInfo.pas**”里面的“**TWVStdCommandDescriptor**”和“**TWVCommandTypeInfo**”对象。

**TWVCommandDescriptor**对象是一个抽象的命令描述对象类，**TWVStdCommandDescriptor**是它的一个具体实现类。**TWVCommandTypeInfo**对象用于在Delphi IDE里面设计**TWVStdCommandDescriptor**对象。

**程序员视点**：命令描述对象的责任：1）说明命令对象；2）负责创建命令对象的实例。命令描述对象其实是一个类工厂。

**程序员视点**：

1）**TWVCommandTypeInfo**对象的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ID | TWVCommandID | 命令ID |
| 属性 | Version | TWVCommandVersion | 版本号 |
| 属性 | Params | TCollection | 命令的参数列表 |
| 事件 | OnGetDescription |  | 当需要获得一个命令的具体描述的时候被触发 |

2）**TWVParam**对象（**TWVCommandTypeInfo**对象的**Params**属性包含的对象，用于描述命令的参数）的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ParamName | string | 参数名称 |
| 属性 | ParamType | TWVParamType | 参数类别（输入/输出/输入并且输出） |
| 属性 | ParamDataType | TKSDataType | 参数数据类型 |
| 属性 | DefaultValue | Variant | 缺省值 |

3）命令对象的属性和方法

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | Error | Exception | 保存的意外对象 |
| 方法 | GetDescription |  | 返回简单描述信息 |
| 方法 | GetDetail |  | 返回详细描述信息 |
| 方法 | GetDescriptor |  | 返回对应的描述对象 |
| 方法 | FindParamData |  | 根据索引或者名字返回参数的值（保存在对象里面） |
| 方法 | ParamData |  | 根据索引返回参数的值（保存在对象里面） |
| 方法 | CheckError |  | 检查返回值 |

### 命令工厂对象

命令工厂对象提供一个创建新的命令对象实例的接口。从实现的角度说，命令工厂对象负责管理系统里面的所有已经注册的命令描述对象。当命令请求对象要求创建指定ID的命令对象实例的时候，命令工厂对象根据ID查找到合适的命令描述对象，由该命令描述对象创建出最终的命令对象实例。

**实现**：命令描述对象的声明和实现是“**WVCommands.pas**”里面的“**TWVCommandFactory**”对象。

### 命令请求对象

命令请求对象用于创建一个命令对象，然后根据工作视图里面工作字段的值，填充命令对象的输入参数的值。然后将命令对象发送到命令总线（概念见下文）上面，要求处理。命令处理完毕以后，命令请求对象可以将命令对象输出参数的值设置到工作字段里面。

**程序员视点**：命令请求对象用于衔接工作视图和命令对象。通过命令请求对象将界面和功能实现联系起来。

命令请求对象包含许多绑定项目，用于说明工作字段和命令参数之间的对应关系。

**实现**：命令请求对象的声明和实现是“**WVCmdReq.pas**”里面的“**TWVRequest**”对象。绑定项目是“**TWVFieldParamBinding**”对象，可以通过“**TWVRequest**”对象的“**Bindings**”属性获得。

**程序员视点**：

1）**TWVRequest**的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | WorkView | TWorkView | 对应的工作视图 |
| 属性 | ID | TWVCommandID | 命令ID |
| 属性 | Version | TWVCommandVersion | 命令版本号 |
| 属性 | Bindings | TCollection | 工作字段和命令参数绑定关系 |
| 属性 | ClearOutputsBeforeExec | Boolean | 在执行前，清空返回值 |
| 事件 | BeforeExec |  | 在命令执行前被触发 |
| 事件 | AfterExec |  | 在命令执行后被触发 |

2）**TWVFieldParamBinding**对象（**TWVRequest**对象的**Bindings**属性包含的对象，代表工作字段和命令参数的对应关系）的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ParamName | string | 命令参数的名称 |
| 属性 | FieldName | string | 工作字段的名称 |
| 属性 | Direction | TWVBindingDirection | 值的传递反向 |
| 属性 | DefaultValue | Variant | 缺省值 |
| 事件 | OnBindInput |  | 在工作字段传输到命令参数的时候被调用 |
| 事件 | OnBindOutput |  | 在命令参数传输到工作字段的时候被调用 |

### 命令处理对象

命令处理对象完成一类命令的实际处理过程。命令处理对象的ID和要处理的命令的ID相同。命令处理对象一般从命令对象里面得到输入参数的值，然后根据业务逻辑进行计算，然后将返回结果放在命令对象的输出参数里面。

**实现**：命令处理对象的声明和实现是“**WVCommands.pas**”里面的“**TWVCommandProcessor**”对象以及“**WVCmdProc.pas**”里面的“**TWVStdCommandProcessor**”和“**TWVCustomProcessor**”对象。

**TWVCommandProcessor**对象是一个抽象的命令处理对象类，**TWVStdCommandProcessor**是它的一个具体实现类。**TWVCustomProcessor**对象用于在Delphi IDE里面设计**TWVStdCommandProcessor**对象。

**实现**：命令处理对象的另外一个实现是“**KCWVProcBinds.pas**”里面的“**TKCWVProcessor**”和“**TKCWVQuery**”。这两个命令处理对象类用于访问金仕达通讯平台。

### 命令总线对象

命令总线对象的作用是将命令请求对象初始化的命令对象发送给合适的命令处理对象处理，然后将处理以后的命令对象返回给命令请求对象。在命令对象被处理以前或者处理以后，命令对象还会提交给命令过滤器对象进行额外的处理。有关命令过滤器对象见下文。

总之，命令总线对象就是命令对象传输的通道，和计算机硬件的总线有一定的相似。在计算机硬件的总线上面传输的是代表数据的电子信号，在命令总线对象上面传输的是命令对象。

命令处理对象必须在命令总线对象上面注册以后，才能够接受到合适的命令对象。

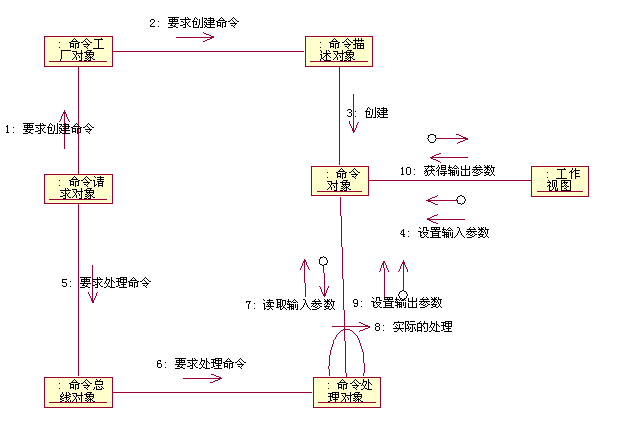
**实现**：命令处理对象的声明和实现是“**WVCommands.pas**”里面的“**TWVCommandBus**”对象。

### 命令模型上下文对象

命令模型上下文对象提供一个运行环境，为命令模型提供运行上面的支持。具体的说，上下文对象负责创建一个命令工厂对象和一个命令总线对象。命令工厂对象管理命令描述对象的注册，命令总线对象管理命令处理对象和命令过滤器对象的注册。

**实现**：命令处理对象的声明和实现是“**WVCommands.pas**”里面的“**TWVContext**”对象。

### 命令对象的生命周期－动态模型



在程序环境中有了上下文对象，和它包含的命令工厂对象和命令总线对象，以及注册在上面的命令描述对象和命令处理对象，命令模型就可以正常工作了。

具体的工作流程如下：

1、准备阶段：

1）命令描述对象在上下文对象包含的命令工厂对象上面注册。

2）命令处理对象在上下文对象包含的命令总线对象上面注册。

2、运行阶段：

1）命令请求对象要求命令工厂对象创建一个已知ID的新的命令对象。

2）命令工厂对象要求一个已经注册的命令描述对象（它的ID和要求的ID相同）创建出新的命令对象。这个时候命令对象的各个参数的值是命令描述对象规定的缺省值。这个时候命令对象被产生。

3）命令请求对象根据绑定关系，将工作字段的值设置到命令对象的输入参数上面。这个时候命令对象已经准备好了。

4）命令请求对象要求命令总线对象传输命令。

5）命令总线对象将命令对象传输给合适的命令处理对象要求处理。

6）命令处理对象一般从命令对象里面得到输入参数的值，然后根据业务逻辑进行计算，然后将返回结果放在命令对象的输出参数里面。这个时候命令对象已经处理完了。

7）命令对象被传回命令请求对象。

8）命令请求对象根据绑定关系，将命令对象的输出参数的值设置到工作字段上面。

9）命令请求对象释放已经使用完毕的命令对象。这个时候命令对象被销毁。

### 命令过滤器对象

命令过滤器对象的目的是在命令对象被命令处理对象处理以前或者处理以后，截获命令对象。命令过滤器对象可以完成权限管理、日志记录等功能。

**实现**：命令处理对象的声明和实现是“**WVCommands.pas**”里面的“**TWVCommandFilter**”对象。

### 小结

命令模型主要的目的是分离界面和实际的业务处理，实现松耦和。

## 3通用数据访问

### 引入通用数据访问的原因

在现在的应用系统当中，有各种数据源，例如各种关系数据库、文件系统等等。要访问这些系统的数据，往往需要各个系统规定的不同的应用程序接口（API）。虽然已经有一些通用的数据访问接口，例如**ODBC**和**BDE**，但是它们都主要针对现有的关系数据库，而且接口都比较复杂，效率也不是特别高。

为了即能够访问到这些现有的数据源，又能够快速的扩充，以支持各种新形式（例如金仕达通讯平台），所以需要建立一种简单实用的通用数据访问。简单的意思是接口简单，便于访问和扩充。实用的意思是可以覆盖大部分的常见应用。

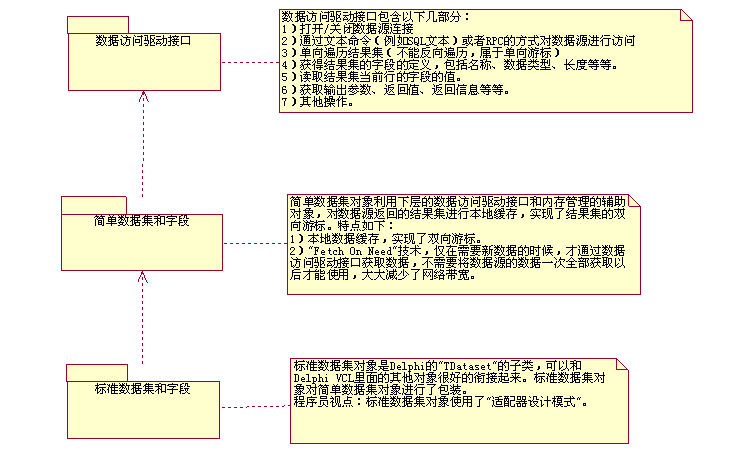
### 通用数据访问的层次

通用数据访问分为三个大的层次：

1）最下层是数据访问驱动接口的实现层。驱动接口是标准的，只需要为各个不同的数据源实现各自的处理。

2）中层是简单数据集和字段。该层利用下层的驱动接口，利用本地数据缓存，实现双向游标。

3）最上层是标准数据集和字段，对中层的简单数据集和字段进行包装，以兼容**Delphi**标准的**TDataset**，方便程序员使用。



### 数据访问驱动接口

数据访问驱动接口包含以下几部分：

1）打开/关闭数据源连接

2）通过文本命令（例如SQL文本）或者RPC的方式对数据源进行访问

3）单向遍历结果集（不能反向遍历，属于单向游标）

4）获得结果集的字段的定义，包括名称、数据类型、长度等等。

5）读取结果集当前行的字段的值。

6）获取输出参数、返回值、返回信息等等。

7）其他操作。

该接口的主要特点是：

1）单向游标，便于实现。

2）支持各种访问形式：文本命令、RPC。

3）支持多种返回数据的形式：结果集、多结果集、返回值、返回信息、输出参数（RPC）。

**定义**：数据访问驱动接口的定义在“**DBAIntf.pas**”里面的“**IDBAccess**”。相关常量、结构的定义都在同一个文件里面。

### 内存管理

为了在本地保存数据需要对内存进行管理。

**实现**：“**TxtDB.pas**”里面的“**TTextDataset**”。

### 简单数据集和字段

简单数据集对象利用下层的数据访问驱动接口和内存管理的辅助对象，对数据源返回的结果集进行本地缓存，实现了结果集的双向游标。特点如下：

1）本地数据缓存，实现了双向游标。

2）“Fetch On Need”技术，仅在需要新数据的时候，才通过数据访问驱动接口获取数据，不需要将数据源的数据一次全部获取以后才能使用，大大减少了网络带宽。

**实现**：“**BDAImp.pas**”里面的“**THDataset**”和“**THField**”。

### 标准数据集和字段

标准数据集对象是Delphi的“**TDataset**”的子类，可以和Delphi VCL里面的其他对象很好的衔接起来。标准数据集对象对简单数据集对象进行了包装。

**程序员视点**：标准数据集对象使用了“适配器设计模式”。

**实现**：“**BDAImpEx.pas**”里面的“**THCustomDataset**”、“**THQuery**”、“**THSimpleDataset**”。

## 4访问通讯平台

### 通讯平台的API接口

通讯平台的主要API函数的功能是打开/关闭到应用服务器的连接，向应用服务器发送数据包，从应用服务器接受数据包。

**实现**：“**DRTPAPI.pas**”。需要动态连接库“**DRTPAPI.DLL**”的支持。

### 通讯包格式

通讯包的格式是一个结构体，包括包头和包体两部分。包头有一个区域，是客户端和服务器之间传递数据的cookie。另外一部分是位图结构，说明包体里面，哪些字段有效。在传输的时候，根据位图结构，包体的数据被紧缩（只传递有效的数据）。

**实现**：“**KCDataPack.pas**”里面的常量和结构的实现。

### 实现访问通讯平台的数据访问驱动接口

根据前面的说明，只需要根据通讯平台的API和通讯包格式，实现数据访问驱动接口。使得其他部分的程序可以方便的访问通讯平台。

**实现**：在“**KCDataAccess.pas**”里面包含了所有的需要的实现。其中“**TKCAccess**”对象根据通讯平台的API和通讯包格式，实现了数据访问驱动接口。“TKCDa**t**aset”对象是“**THCustomDataset**”的子类，也是标准数据集（**TDataset**）的子类，利用“**TKCAccess**”对象实现的数据访问驱动接口，访问通讯平台应用服务器。“**TKCDatabase**”对象负责管理到通讯平台的连接。

**程序员视点**：

1）**TKCDataset**的设计时候的属性和事件（不列出从**TDataset**继承下来的缺省属性和事件）

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | FuncNo | Word | 主功能号 |
| 属性 | RequestType | Integer | 请求的功能号 |
| 属性 | Params | THRPCParams | 输入参数 |
| 属性 | Timeout | Integer | 等待超时的毫秒数 |
| 属性 | Priority | Byte | 优先级 |
| 属性 | SleepBetweenReceive | Integer | 在两次接受之间的等待毫秒数 |
| 属性 | IsCallback | TDatasetCallback | 是否在等待的时候调用回调函数，更新显示 |
| 事件 | OnDoProgress |  | 在等待接受的时候被调用 |
| 事件 | OnWait |  | 在等待接受的时候被调用 |
| 事件 | AfterReceive |  | 在接受数据之后被调用 |
| 事件 | BeforeReceive |  | 在接受数据之前被调用 |

2) **TKCDataset**的运行时候的属性和事件（不列出从**TDataset**继承下来的缺省属性和方法）

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ReturnCode | Integer | 返回码 |
|  |  |  |  |

3）使用TKC**D**ataset的方法

a）现在Pa**r**ams属性里面增加输入参数

b）然后用Op**e**n方法或者设置**Active**:=**True**，打开结果集。

c）利用**Fields**属性获得返回数据。

4）**TKCDatabase**的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | GatewayIP | string | 通讯平台IP地址 |
| 属性 | GatewayPort | Word | 通讯平台端口号 |
| 属性 | GatewayIP2 | string | 备用通讯平台IP地址。当第一台通讯平台无效的时候使用。 |
| 属性 | GatewayPort2 | Word | 备用通讯平台端口号。 |
| 属性 | GatewayEncode | Word | 通讯平台加密方式 |
| 属性 | DestNo | Word | 目的地址编号 |
| 属性 | FuncNo | Word | 主功能号 |
| 属性 | Priority | Byte | 优先级 |
| 属性 | UserID | Integer | 用户ID（内部使用） |
| 属性 | Timeout | Integer | 超时毫秒数 |
| 属性 | SleepBetweenReceive | Integer | 两次等待之间的间隔毫秒数，内部使用 |
| 属性 | IsCallback | Boolean | 是否在等待的时候调用回调函数，更新显示 |
| 属性 | MaxTryReceiveCount | Longword | 最多尝试接受的次数，内部使用 |
| 属性 | TryReceiveTimeOut | Longword | 再次尝试接受数据的超时毫秒数，内部使用 |
| 属性 | CheckDataTimeOut | Longword | 每次检查数据是否到达的等待毫秒数，内部使用 |
| 属性 | Connected | Boolean | 是否建立了连接 |
| 属性 | MaxConnection; | Integer | 保留不用，内部使用 |
| 属性 | MinConnection; | Integer | 最小缓冲的连接数，内部使用 |
| 事件 | OnDoProgress |  | 在等待接受的时候被调用 |
| 事件 | OnWait |  | 在等待接受的时候被调用，内部使用 |
| 事件 | AfterReceive |  | 在接受数据之后被调用 |
| 事件 | BeforeReceive |  | 在接受数据之前被调用 |
| 事件 | OnConnected; |  | 在建立连接的时候被调用 |
| 事件 | OnDisConnected; |  | 在关闭连接的时候被调用 |

### 通讯平台访问和命令模型的衔接

**实现**：在“**KCWVProcBinds.pas**”里面实现了两个命令处理对象的子类“**TKCWVProcessor**”和“**TKCWVQuery**”。这两个对象可以方便的将通讯平台访问和命令模型的衔接起来。

**程序员视点**：

1）**TKCWVCustomProcessor**（“**TKCWVProcessor**”和“**TKCWVQuery**”的公共父类）的设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | InputBindings | TCollection | 输入参数和数据集参数的绑定关系 |
| 属性 | OutputBindings | TCollection | 输出参数和数据集字段的绑定关系 |
| 属性 | FuncNo | Word | 主功能号 |
| 属性 | RequestType | Integer | 功能号 |
| 属性 | Timeout | Integer | 超时毫秒数 |
| 事件 | OnBindInput |  | 输入绑定的时候触发 |
| 事件 | OnBindOutput |  | 输出绑定的时候触发 |
| 属性 | ID | TWVCommandID | 命令ID |
| 属性 | Version | TWVCommandVersion | 命令版本号 |

2）**TKCParamBinding**（绑定关系）设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | ParamName | string | 命令对象的参数每次 |
| 属性 | FieldName | string | 输入绑定的时候：数据集对象的参数名称。  输出绑定的时候：数据集对象的字段名称。 |
| 属性 | DefaultValue | Variant | 缺省值 |
| 事件 | OnBindInput |  | 输入绑定的时候触发 |
| 事件 | OnBindOutput |  | 输出绑定的时候触发 |

3）**TKCWVProcessor**设计时候的属性和事件

|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | KCDataset | TKCDataset | 关联的数据集对象 |

4）**TKCWVQuery**设计时候的属性和事件

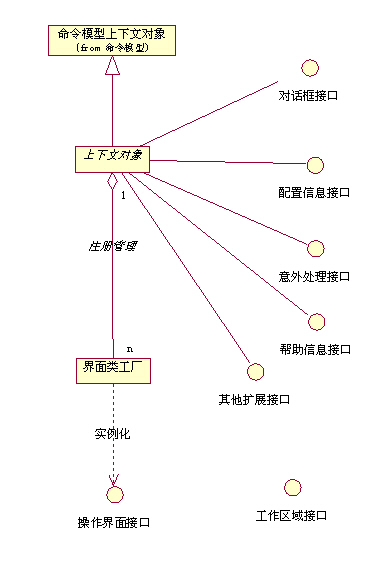
|  |  |  |  |
| --- | --- | --- | --- |
| **类别** | **名称** | **数据类型** | **说明** |
| 属性 | KCDatabase | TKCDatabase | 关联的数据库对象 |

## 5应用程序框架

### 引入应用程序框架的目的

为高效率高质量地开发应用程序，势必需要一套程序框架。程序的各个部分逐渐在框架的基础上面搭建起来。成熟的框架一方面可以简化程序的开发，另一方面可以较好的进行模块分解。

### 静态结构



具体介绍如下。

### 上下文对象和各种服务接口

应用程序上下文对象是命令模型上下文对象的子类。除了为命令模型提供运行环境以外，还为整个应用程序的运行提供环境上面的支持。应用程序上下文对象提供了几个可以被各部分程序访问的功能接口：

1）对话框接口：提供一个在应用程序使用常见对话框的统一接口。

2）配置信息接口：提供一个读取应用程序配置信息的统一接口。配置信息可以来源于多个配置文件，也可以来自数据源。

3）意外处理接口：提供统一的处理错误的接口。

4）帮助信息接口：提供统一的显示在线帮助信息的接口。

5）其他扩展接口。

**实现**：在“**KSFrameWorks.pas**”里面包含了所有程序框架需要的接口定义。其中“**TKSContext**”是上下文对象，它是一个抽象类。“**IHelpSystem**”是帮助信息接口，“**IExceptionHandler**”是意外处理接口，“**IDialogSystem**”是对话框接口，“**IConfigSystem**”是配置信息接口。

**实现**：在“**UContext.pas**”里面完成了大部分的对这些接口的实现。其中“**TMainContext**”是“**TKSContext**”的具体实现子类，“**TdmContext**”实现了“**IHelpSystem**”、“**IDialogSystem**”、“**IConfigSystem**”、“**IExceptionHandler**”这些接口。

**程序员视点**：

1）**TMainContext**抽象方法

|  |  |  |
| --- | --- | --- |
| **类别** | **名称** | **说明** |
| 抽象 | GetHelpSystem | 返回帮助接口的实现 |
| 抽象 | GetExceptionHandler | 返回意外处理接口的实现 |
| 抽象 | GetDialogSystem | 返回对话框接口的实现 |
| 抽象 | GetConfigSystem | 返回配置信息接口的实现 |
| 抽象 | GetSystemObject | 返回系统级别的共享对象 |
| 抽象 | GetDebugSystem | 返回调试接口的实现 |
| 抽象 | SetDebugSystem | 设置调试接口的实现 |
| 抽象 | GetRobotSystem: | 返回自动化接口的实现 |
| 抽象 | SetRobotSystem | 设置自动化接口的实现 |
| 抽象 | GetCopyRight | 返回版权接口的实现 |
| 抽象 | SetCopyRight | 设置版权接口的实现 |
| 抽象 | NewWorkSpace | 为操作界面新建一个工作区域 |
| 抽象 | GetDir | 返回特定的文件名路径 |
| 抽象 | GetParamValue | 获得系统参数的值 |
| 抽象 | GetLogined | 返回是否登录 |
| 抽象 | RefreshSysParams | 更新系统参数的值 |

2）主要的服务接口定义的说明

|  |  |  |
| --- | --- | --- |
| **接口** | **方法** | **说明** |
| IHelpSystem | ShowHelp | 显示帮助 |
| IExceptionHandler | HandleException | 处理意外 |
| IDialogSystem | Confirm | 确认对话框1 |
| Confirm2 | 确认对话框2 |
| Warning | 警告对话框，要求确认 |
| ShowMessage | 显示信息 |
| ShowError | 显示错误 |
| ShowError2 | 显示错误，提示是否继续 |
| ShowWarning | 显示警告 |
| ShowWaiting | 显示等待窗口 |
| CloseWaiting | 关闭等待窗口 |
| DoSingleStationCheck | 单机双柜员复核对话框 |
| ShowHintFor | 显示提示小窗口 |
| ShowSuccess | 显示成功信息 |
| InputText | 要求输入一行文字（包括密码） |
| IConfigSystem | ReadString | 读取字符串类型的配置 |
| ReadInteger | 读取整数类型的配置 |
| ReadBoolean | 读取布尔类型的配置 |
| ReadFloat | 读取浮点数类型的配置 |
| WriteString | 保存字符串类型的配置 |
| WriteInteger | 保存整数类型的配置 |
| WriteBoolean | 保存布尔类型的配置 |
| WriteFloat | 保存浮点数类型的配置 |
| IWorkSpace | GetContainer | 获得放置操作界面的载体 |
| FitSize | 调整工作区域的大小 |
| UIStatusChanged | 通知工作区域，操作界面已经发生改变 |
| Release | 释放 |
| IUIInstance | CanClose | 释放可以关闭 |
| GetCaption | 获得标题 |
| SetCaption | 设置标题 |
| GetHelpID | 获得上下文帮助ID |
| Init | 初始化 |
| InitEx | 高级初始化 |
| CanDoAction | 是否可以完成特殊操作 |
| DoAction | 完成特殊操作 |
| SystemNotify | 通知系统变化 |

### 操作界面和主程序之间的接口

每个具体的操作界面被嵌套在主程序里面，展示给用户操作。所有的操作界面都必须实现标准的操作界面接口。该接口规定了操作界面程序需要实现的基本行为。

同样主程序提供给操作界面使用的工作区域也需要实现一定的接口，该接口规定了工作区域所需要的基本行为。

**实现**：在“**KSFrameWorks.pas**”里面“**IUIInstance**”定义了操作界面接口，“**IWorkSpace**”定义了工作区域接口。

**实现**：在“**KSFrames.pas**”里面的“**TKSFrame**”对象是操作界面的一个标准父类，集中交易程序里面所有的操作界面都是从该对象继承下来的。该对象实现了“**IUIInstance**”接口。

### 程序组织和菜单

一个菜单对应一个操作界面。每个操作界面有一个唯一标识该界面的ID。每个操作界面还对应一个界面类工厂的实例。由界面类工厂的实例负责根据主程序的需要创建出操作界面的实例。具体的步骤如下：

1）程序包在装载的时候，在每一个操作界面的代码的初始化部分，创建一个界面类工厂的实例，并且将界面ID和该界面的实现类传递给该界面类工厂。

2）界面类工厂自动在框架程序里面注册。

3）程序运行的时候，用户选择某一个菜单。主程序根据菜单ID，在已经注册的界面类工厂里面找到合适的界面类工厂。

4）主程序要求界面类工厂创建出一个操作界面的实例。

5）界面类工厂根据前面得到的界面的实现类创建出具体的操作界面的实例。

6）根据前面介绍的操作界面接口和工作区域接口将操作界面和主程序关联起来。使得主程序和操作界面之间可以互相沟通。

**实现：**在“**KSFrameWorks.pas**”里面“**TUIClassFactory**”、“**TUIStdClassFactory**”是界面类工厂的实现类。

### 程序启动完成的工作

1、主程序创建应用程序上下文对象。

2、应用程序上下文对象完成初始化。

3、主程序创建公共数据模块

4、主程序根据配置文件读取需要装载的程序包

5、依次装载程序包

6、程序包被装载以后，会自动创建：

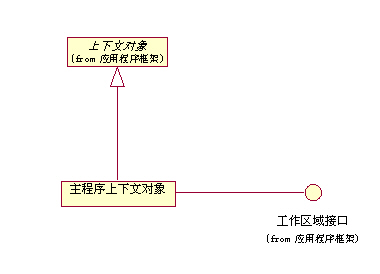
a）数据模块，里面包含命令描述对象、命令处理对象。在创建的时候，完成命令描述对象和命令处理对象在上下文对象里面的注册

b）界面类工厂，会自动注册到应用程序框架。

7、主界面根据菜单文件，创建菜单。

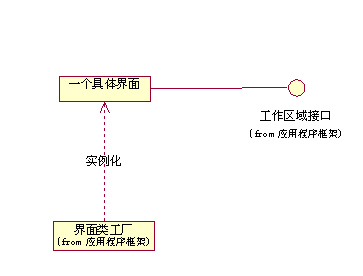
## 6主程序

主程序里面实现了在应用程序框架里面定义的抽象的上下文对象及其提供的各种接口。



## 7具体功能模块

在具体的功能模块里面实现各种操作界面。操作界面必须实现专门的接口。



# 十二、组件视图

该视图里面描述了在运行时候，各个组件之间的关系。其中“基础程序包”包括了最下层“工作视图模型”、“命令模型”、“通用数据访问”三个部分的实现。“应用程序框架包”包含了“应用程序框架”的实现。“主程序”和“具体功能模块”两个程序包分别对应“主程序”和“具体功能模块”的实现。

主程序和各个具体功能模块之间是松耦和的，它们通过应用程序框架程序包进行通讯，相互之间没有直接的依赖关系。各个具体功能模块之间也是松耦和的，相互之间也没有直接的依赖关系。

因为整个应用程序是松耦和，所以功能模块可以根据业务需要自由组合、搭建。充分体现了面向组件（Component）的优势。



# 第三部分 开发入门部分

# 十三、操作界面程序开发一般流程

## A、几个基本概念和基本对象类

### 1、工作视图、工作字段、控件

一个工作视图包含多个工作字段，每个工作字段代表一个数据项目。控件是数据的表现和录入工具。当工作字段的值发生改变的时候，这些值自动/手动地传递给控件，让控件显示出正确的数据。当用户在控件输入数据以后，控件自动/手动地将改变以后的数据传递给工作字段。通过这些机制保证工作字段的值和控件一致。

### 2、数据访问组件和交易编码

数据访问组件TKCDataset和TKCDatabase封装了通过通讯平台访问应用服务器的细节。

交易编码是一个业务请求的定义，包括输入参数，输出参数，处理逻辑，交易代码等等。

TKCDataset的参数（Params）代表了交易编码输入参数，结果集的字段（Fields）代表交易编码的输出参数。TKCDataset的ReturnCode表示返回码，返回码＝0表示没有错误，>0表示错误。

### 3、命令对象、命令描述对象、命令处理对象、上下文对象等

引入命令对象的目的是为了分割界面和实现，达到松耦和的目的。一个命令对象包含了多个输入参数和多个输出参数。命令对象通过ID和Version标志功能。

命令描述对象是命令对象的元类，用来描述一个ID和Version标志的命令具有的各个参数的类型，以及创建一个命令对象的实例（特定的ID和Version）。

命令工厂负责管理注册的命令对象描述对象的实例。

命令处理对象实际处理一类（通过ID和Version标志）命令对象，即根据命令对象的输入参数，（广义的）计算出命令对象的输出参数。

命令总线对象将一个命令对象实例发送给合适的命令处理对象的实例，让命令处理对象完成处理，然后把处理结果（在命令对象的输出参数里面）返回给调用者。

上下文对象为命令对象、命令描述对象、命令处理对象提供一个运行的环境。

### 4、命令请求对象

命令请求对象的作用是：创建命令，将工作视图的字段（工作字段）的值传递给命令对象的参数；等命令对象执行完成以后，将命令对象的参数的值传递回工作视图的字段。

### 5、特殊的命令处理对象

TKCWVProcessor和TKCWVQuery都是命令处理对象类的具体子类。他们的作用都是：将命令对象的（输入）参数传递给数据访问对象的参数，然后打开/调用数据访问对象，最后将数据访问对象的返回结果集的字段传递给命令对象的（输出）参数。

他们的区别是，TKCWVProcessor对应交易类型的处理，返回简单数据；TKCWVQuery对应查询类型的处理，要返回结果集对象。

## B、开发新程序的一般步骤

### 1、准备开发环境

参看“二、环境”“2、开发环境”“环境建立”部分。

### 2、准备编写新程序

创建新的程序包

1）创建新的Package 到工程组

2）新建数据模块

3）为新界面定义新的菜单

4）修改配置文件，以装载这个新的程序包

### 3、设计数据模块

1）理解交易编码文档

2）增加数据集对象

3）增加命令定义对象

设置命令参数

4）增加命令实现对象

选择Processor 或者Query

参数绑定

### 4、设计新的界面

1）选择继承的父类模版

2）注册新的界面工厂

3）为工作视图定义工作字段

4）将命令请求对象的输入/输出参数和工作字段关联起来

5）增加输入/输出控件，和工作字段关联起来

### 5、运行和调试

1）编译程序

2）运行程序

3）不能出现新的界面的几种问题

a）没有装载新的程序包

b）没有注册界面类工厂

4）查看和了解日志文件

### 6、进一步的完善

1）字段的有效性控制

2）显示反馈信息

3）调整界面布局和输入Tab顺序

## C、工作流程检查表

|  |  |
| --- | --- |
| **内容** | **检查** |
| 1、根据业务确定界面。 |  |
| a)确定单选（下拉框）或者复选（列表框） |  |
| b)目标营业部是否需要选择 |  |
| 2、找到正确、最新的交易编码。**注意**：一个界面可能需要多个交易编码。将交易编码保存为文本形式，使用“列拷贝”的形式，使得参数中文名和英文字段名在同一行上面，为以后的复制作好准备。 |  |
|  |  |
| 3、创建数据模块，**注意**：从TCommandModule继承下来。增加一个TKCDataset对象。 |  |
| 4、根据以交易编码为主，结合界面，确定命令定义对象（TWVCommandTypeInfo）。 |  |
| a）为每个交易编码增加一个命令定义对象。 |  |
| b）设置命令对象的*ID*，一般为交易编码的名字 |  |
| c）根据交易编码的输入参数，增加命令对象的输入参数。可以利用批量增加的形式。数据来源于第2步保存的文本文件。**注意**：命令对象的输入参数不包含“*发起营业部*”“*职工代码*”“*网卡地址*”三个系统参数。 |  |
| d）增加“*返回码*”“*返回值*”作为输出参数。 |  |
| e）对应查询类型的交易编码，增加“*查询结果集*” 作为输出参数。 |  |
| f）否则，对其他类型的交易编码，根据交易编码的输出参数，增加输出参数。可以利用批量增加的形式。数据来源于第2步保存的文本文件。 |  |
| g）检查数据类型是否正确 |  |
| h）允许不传递具体值的命令对象的输入参数，应该设置缺省值。例如：对一些查询交易编码，输入参数“币种”一般不指定，那么设置缺省值为“”，表示查询所有的“币种”。 |  |
| 5、增加命令处理对象 |  |
| a)为每个查询类型的交易编码，增加一个TKCWVQuery；为每个其他类型的交易编码，增加一个TKCWVProcessor，注意设置TKCWVProcessor.Dataset。 |  |
| b)设置命令处理对象的*ID*和前面的命令请求对象相同。 |  |
| c)设置*RequestType* |  |
| d)根据交易编码的输入参数，设置输入参数绑定。可以使用批量增加的形式。注意：“*发起营业部*”“*职工代码*”“*网卡地址*”对应到正确的系统参数“*#营业部代码*”、“*#职工代码*”和“*#网卡地址*”上面。 |  |
| e)将“*返回码*”“*返回值*”的输出参数正确绑定到“*@Return*”和“*vmess*”上面。 |  |
| f)如果是查询交易，将“*查询结果集*”绑定到“*@Dataset*”上面 |  |
| g)否则对其他交易编码，根据输出参数定义，设置绑定。可以利用批量增加的形式。 |  |
| 6、检查第3、4、5步，完成数据模块的设置。注意：不要关闭该模块，方便下面界面模块的设置。 |  |
|  |  |
| 7、开始设置界面模块 |  |
| 8、为工作视图对象(TWorkView)增加工作字段(TWVField)。 |  |
| a) 根据交易编码的输入参数，结合界面，确定需要增加的字段。可以利用批量增加的形式。注意：“*发起营业部*”“*职工代码*”“*网卡地址*”不应该作为字段。  对查询界面，需要增加一个“*查询结果集*”字段，数据类型是“*kdtObject*”，*OwnObject*=True。  对于需要显示反馈信息（例如客户资料）的界面，需要增加对应这些反馈信息的字段。 |  |
| b）**注意**一个界面对应多个交易编码，所以需要考虑为每个交易编码设置字段。但是必须仔细考虑，哪些字段是各个交易编码共用的（即它们使用的值是相同的），哪些不是。例如几个交易编码共用一个“币种”，那么只需要一个“币种”字段。但是如果查询和修改都需要“币种”字段，但是我们不希望修改的时候改变了查询需要的“币种”字段，那么需要两个币种字段，建议分别命名为“查询.币种”和“设置.币种”。 |  |
| c）设置工作字段的数据类型。 |  |
| d）设置工作字段的有效性检查，可以直接设置“*Checker*”属性，或者利用“*DomainName*”，或者利用“*OnCheckValid*”事件。包括根据数据库的要求设置长度限制、数值范围限制、允许输入的字符限制等等。 |  |
| e）复杂的控制需要设置“*MonitorXXX*”属性。 |  |
| f）设置按“Enter”和“\*”改变输入焦点。 |  |
| g）为一些字段设置*DataField*属性，这些字段需要与来自数据集的字段同步。设置的时候，一定要明白数据来源的交易编码是什么（是查询，还是客户验证、取股东代码信息），然后根据建议编码的输出参数定义来设置。 |  |
| h）关键的一步：设置*GroupIndex*。将相关（相同交易编码的输入/输出参数）的字段设置为一组。这样可以避免不同组之间的互相干扰。 |  |
| i）检查上面的设置，完成对工作字段的设置。 |  |
| 9）增加工作视图的*FieldsMonitors*，控制输入有效的时候，确认按键有效。  **注意**：如果应该检验股东代码是否存在。有时候，虽然通过客户号输入控件输入一个正确的客户号，但是该客户可能没有指定市场的股东代码。 |  |
|  |  |
| 10、为每个交易编码增加一个命令请求对象（TWVRequest） |  |
| a）设置*WorkView*和*ID* |  |
| b）设置*Bindings*属性。先使用“Add All Existed Request Bindings”，根据命令定义对象的参数设置绑定。然后进行调整，对各种字段的名字进行必要的修改。 |  |
| c）注意客户号、开户营业部、股东代码一般来源于通过客户号输入控件反馈给工作视图的字段。 |  |
|  |  |
| 11、设置各种控件 |  |
| a）设置*WorkView*和*WorkField*属性 |  |
| b）对于下拉框和列表框，通过“*ItemDataEntry*”属性设置数据字典条目，可以根据数据字典初始化列表。 |  |
| c）对于选择数据字典的列表框，设置*DataPresentType*=IDOBJECTS, *Style*=csDropDownList |  |
| d）对于其他的列表框，设置“*SeperatedStr*”属性 |  |
| e）对于编辑框，输入输入数字，可以考虑设置“*SynchronizedWhenExit*”属性为True。 |  |
| f）对于CheckBox，注意：*CheckedValue*、*UnCheckedValue*、*DefaultChecked*、*PresentType*、*SeperateChar*、*CharIndex*属性。 |  |
| g）对于TWVCheckListBox，注意：*CheckedValue*、*UnCheckedValue*、*DefaultChecked*、*PresentType*、*SeperateChar*、*SeperatedStr*、*SynchronizeWhenExit*属性。 |  |
| h）对于形式数据字典的标签，需要设置*DataPresentType*=IDOBJECTS和*DataPresentParam*＝数据字典条目。 |  |
| 12、如果需要输入客户，使用客户输入控件。 |  |
| a）**注意**一下属性：*AllowMethods*、*LabelCtrl*、*SwitchKey*、*FirstMethodChar*、*NeedPassword*、*NeedReinput*、*NeedMainAccounts*、*NeedMoneyAccounts*、*PasswordType*、*WorkView*、*CustInfoGroupIndex*、*AccountInfoGroupIndex*、*MoneyInfoGroupIndex*、*GoNextWhenPressEnter*、*GoPrevChar*。 |  |
| b）特别**注意**：*CustInfoGroupIndex*、*AccountInfoGroupIndex*、*MoneyInfoGroupIndex*。检查工作视图里面的工作字段是否正确指定了*GroupIndex*：和客户资料、股东资料、资金情况相关的字段的*GroupIndex*是否和客户号输入控件指定的相同，其他无关的字段的*GroupIndex*应该不同。 |  |
| c）**注意**：和客户资料、股东资料、资金情况相关的字段的*DataField*字段是否正确指定。 |  |
|  |  |
| 13、对于查询类型的界面， |  |
| a）找到查询对应的交易编码的输出参数定义。 |  |
| b）增加一个TWVDBBinder对象。设置*WorkView*、*FieldName*(一般是“查询结果集”)、*DataSource* |  |
| c）对需要根据数据字典进行翻译的字段，设置Bindings属性，设置*FieldName*（数据集的字段名，参看对应交易编码），*DataPresentType*=“IDOBJECTS”，*DataPresentParam*＝数据字典条目。 |  |
| d）设置DBGrid里面列标题和字段名，可以使用批量增加的形式。调整好列的宽度和顺序。 |  |
| 14、对于参数设置界面，增加一个TWVDataSource控件，设置好*WorkView*、*DataSource*、*GroupIndex*属性。 |  |
|  |  |
| 15、将控件对齐，位置摆放整齐。 |  |
| 16、调整好*TabOrder* |  |
| 17、调整好界面的大小，注意Grid和Panel的*Align*属性。按键的*Anchor*属性。 |  |
|  |  |
| 18、编写其他代码： |  |
|  |  |
| 19、调试 |  |

# 十四、完全开发实例讲解

## 输入模板的使用(例：单个客户业务控制)

### 准备工作

### 定义菜单

规定菜单名称为“单个客户业务控制演示”，修改文本形式的菜单文件menus.mnt，最后加密为二进制菜单文件menus.mn。

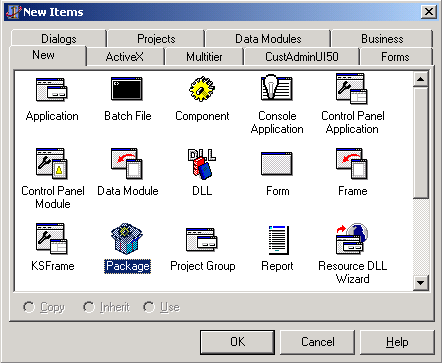
### 创建包

1、新建包

方法1：点选菜单“File”下的“New…”项目

方法2：直接点选快捷按钮“new”

即出现“New Items”对话框，如下图一所示



(图一)

2、“确认”后的需要做的工作：

1> 在包编辑器的“Requires”列下，点击快捷按钮“Add”增加两个引用的包文件：

KSFranework50.dcp 和 KSGTPlatform50.dcp

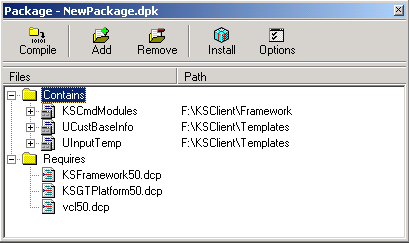
2> 在在包编辑器的“Contains”列下，点击快捷按钮“Add”增加三个模板文件：

KSCmdModules.pas：数据模块模板

UCustBaseInfo.pas： 客户基本信息模板

UInputTemp.pas： 输入模板

如下图二所示。



(图二)

1. 包保存

1> 存盘，命名为“NewPackage.dpk”

2> “NewPackage.dpk”保存到“DBDemos”子目录下面。

3> 设置程序包的目标文件路径和引用的程序包。

新建包完成

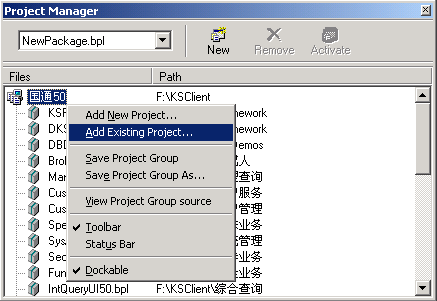
1. 添加包

1> 将工程包“国通50.bpg”设置为可写的。

2> 打开“国通50.bpg”，点击菜单“View”下的“Project Manage”，出现“Project Manage”的对话框。

3> 在“Project Manage”下，给“国通50.bpg”项目添加包“NewPackage.dpk”。

如下图三所示。



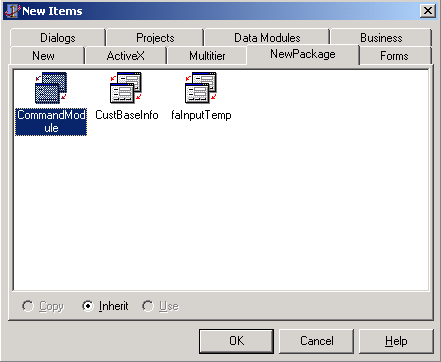
(图三)

1. 继承模板

1> 在“Project Manage”下，双击包“NewPackage.bpl”，即将当前活动的工程设置为“NewPackage.bpl”

* 1. 然后操作如1步骤，选择“New”，在“New Items”对话框里面的选择“PackageTemp”页，选择正确的父类。

如下图四所示。



(图四)

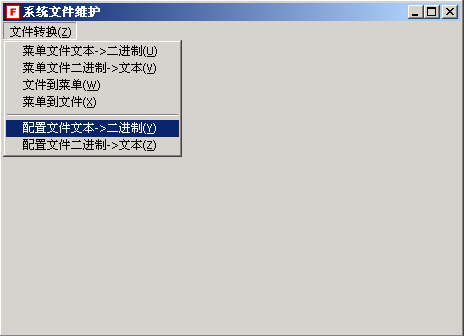
**注意：** 对话框“NewPackage”页面中的操作选项Copy / Inherit / Use中一定要选择“Inherit”项（其实它已是默认值）

3>“确认”后，继承于数据模块模板的新的数据模块CommandModule1就生成了

**注意：**在CommandModule1相应的单元文件Unit1中，可以在uses部分看到KSCmdModules数据模块模板文件。

### 修改配置文件及其他

1. 改配置文件ksclient.ini，装载该程序包，并且在Comfirm部分增加一个确认的条目。最后利用SysFileMan50.exe将其加密为二进制配置文件ksclient.in，如下图五所示。



(图五)

[Packages]

Count=32 🡪 **Count=33**

1=DBDemo.BPL

2=KSDebug50.BPL

3=KSRobot50.BPL

4=KSDbgPrjRep50.bpl

5=KsGTCopyRight50.bpl

……………………………

……………………………

30=ManagerQueryUI50.bpl

31=ReportUI50.BPL

32=ReturnCmsUI50.bpl

**33=NewPackage.bpl**

[Confirm]

简单委托=1

选帐号委托=1

境外委托=1

补委托单=1

………………

………………

**客户业务控制=2**

**注：**1，2 是不同风格的对话框

2、找到相应的交易编码，复制出输入/输出参数文本，对文本进行格式化。

交易编码：110401

功能描述：设置客户的业务控制

相关数据表：客户基本资料（CUST\_BASE\_INFO）

输入参数：

客户号 scust\_no

业务控制串 scust\_limit

营业部代码 sbranch\_code0

职工代码 semp

网卡号 sstation

输出参数：

原来的业务控制串scust\_limit

## 设计数据模块

### 数据模块定义

在前面创建的新的数据模块CommandModule1中，做如下设计：

1. 在单元的“Initialization”部分编写代码，创建该数据模块。

CommandModule1 := T CommandModule1.Create (Application);

2、增加对象控件且用“**命名规范**”命名

在数据模块上面增加一个数据集对象KCDataset1，命名为“Dataset”

在数据模块上面增加一个命令定义对象WVCommandTypeInfo1，命名为“ctCtrlCust”

在数据模块上面增加一个命令处理对象KCWVProcessor1，命名为“cpCtrlCust”

**注意：**在设置命令定义对象和命令处理对象的时候，可以利用前面从交易编码里面复制出来的文本，进行批量添加。批量添加以后再逐个调整一下。

### 命令定义对象和命令处理对象

1. **命令定义对象（ctCtrlCust）**
   1. 增加输入/输出参数

输入参数：

客户号 scust\_no

业务控制串 scust\_limit

---------------------------------------------

营业部代码 sbranch\_code0

职工代码 semp **这部分不要**

网卡号 sstation

---------------------------------------------

输出参数：

原来的业务控制串scust\_limit

再新添两个输出参数：

返回码 @return

返回信息 vsmess

所以，**实际**增加的参数如下：

客户号 scust\_no

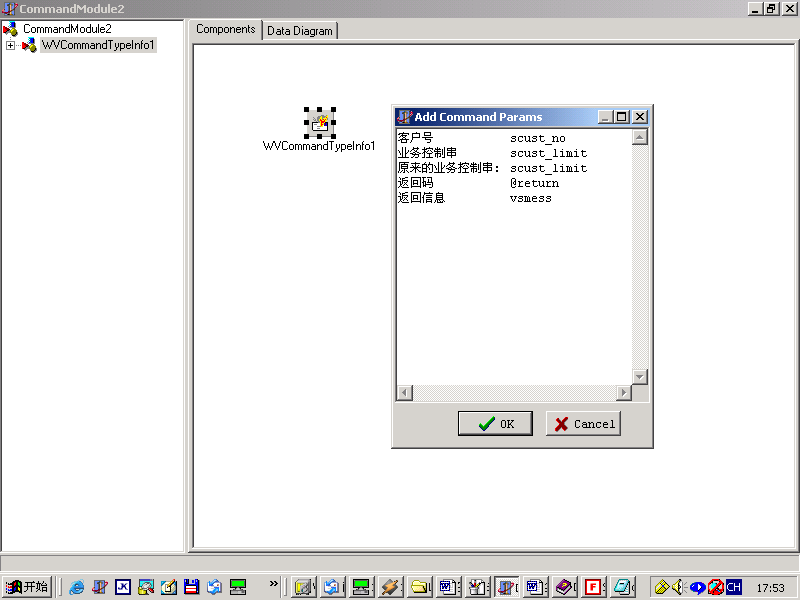
业务控制串 scust\_limit

原来的业务控制串scust\_limit

返回码 @return

返回信息 vsmess

如下图六所示



（图六）

* 1. 参数属性设置

“返回码”、“返回信息”和“原来的业务控制串”的ParamType属性设置为ptOutput，其余参数的ParamType属性设置为ptInput（默认值）

3> 对象属性设置

将ID属性设置为“**设置客户的业务控制**”，手动输入

1. **命令处理对象（cpCtrlCust）**
   1. 增加输入参数

输入参数：

客户号 scust\_no

业务控制串 scust\_limit

------------------------------------------------------------------------------------

营业部代码 sbranch\_code0 🡪 #营业部代码 sbarnch\_code0

职工代码 semp 🡪 #职工代码 semp

网卡号 sstation 🡪 #网卡地址 sstation0

------------------------------------------------------------------------------------

所以，实际**增加**的输入参数如下：

客户号 scust\_no

业务控制串 scust\_limit

#营业部代码 sbarnch\_code0

#职工代码 semp

#网卡地址 sstation0

* 1. 增加输出参数

输出参数：

原来的业务控制串scust\_limit

再新添两个输出参数：

返回码 @return

返回信息 vsmess

所以，**实际**添加的输出参数如下：

原来的业务控制串scust\_limit

返回码 @return

返回信息 vsmess

* 1. 对象属性设置

>>将ID属性设置为“**设置客户的业务控制**”，如命令定义对象ID属性设置了，

则可直接点选设置；

>>将KCDataset属性设置为“**Dataset**”

>>将RequestType属性设置为“**110401**”

### 模块命名

将模块命名为“UCMCtrlCust”

## 设计界面

### 选择适合的界面模板继承创建界面并进行界面设置

1、 选择适合的界面模板继承建立界面

利用前面介绍的方法，在程序包“NewPackage.dpk”里面新建一个Frame，从“**TfaInputTemp**”继承下来。

在单元的“**Initialization**”部分编写代码，创建界面工厂，注册该界面。

**initialization**

TUIStdClassFactory.Create ( '单个客户业务控制演示' , TfaInputTemp1 ) ;

2、界面设置

Frame的caption属性设置为“单个客户业务控制演示”；

Frame的HelpID属性设置为“单个客户业务控制演示”；

Frame的Name属性设置为“faCtrlCust”；

**注意：**将初始化部分的代码

TUIStdClassFactory.Create ( '单个客户业务控制演示' , **TfaInputTemp1** ) ; **改为**

TUIStdClassFactory.Create ( '单个客户业务控制演示' , **TfaCtrlCust**) ;

3、保存界面单元为“**UCtrlCust**”

### 工作视图定义

1、增加字段并对字段分组

为工作视图增加字段

一部分来自交易编码： 字段组别（GroupIndex属性） 客户号 scust\_no 0

业务控制串 scust\_limit 0

原来的业务控制串 scust\_limit 2

另外一部分是客户信息：

客户.营业部 sbranch\_code0 1

客户.客户号 scust\_no 1

客户.客户姓名 sname 1

客户.客户证件类别 lcert\_code 1

客户.客户证件 scert\_no 1

客户.客户状态 sstatus0 1

客户.代理姓名 sname2 1

客户.代理证件类别 lsafe\_level 1

客户.代理证件 scert\_no2 1

客户.代理级别 scust\_limit2 1

客户.代理期限 sdate0 1

根据要求增加与客户相关的此字段：

客户.业务控制串 scust\_limit 1

**分组说明：**一般地，作为输入参数的字段组别为“0”；作为输出参数组别为“2”；

作为客户**（基本）**信息的字段组别为“1”。

2、字段的有效性检验和约束性控制

对字段的Constrained属性进行设置，如字段需要约束性控制，则将Constrained属性设置为True

3、字段监控，目的：控制界面中“**确认**”键的Enabled属性（即**确认**是否可按下）

工作字段FieldsMonitors属性中

增加一子项，点选设置MonitorValidChangedFields属性

即MonitorValidChangedFields := '**|客户.客户号|客户的业务控制串|**'

### 命令请求对象的定义

1、设置命令请求对象的ID

如数据模块是打开状态，可点选设置为“**设置客户的业务控制**”

2、参数绑定

点选MainRequest命令请求对象，按鼠标右键，点选“Add All Existed Request Bindings”自动绑定参数

1. 手工调整

1> 移除“返回码”和“返回信息”项；

2> 将“客户号”binding项的FileName属性从“客户号”改为“客户.客户号”，参看后面编码部分的说明（**字段"客户.客户号"和客户号控件的有效性同步**）就可明白这样设置的意义所在。

### 增加控件

1. 增加客户号控件，命名为“**edCustNo**”
   1. 设置CustInfoGroupIndex属性（默认值为1），可根据需求设置任意整数

**注意：**在工作视图中，客户**（基本）**信息的字段组别（GroupIndex属性）均为“1”。

* 1. 和工作视图关联

参看后面编码部分的说明（**字段"客户.客户号"和客户号控件的有效性同步**）

* 1. 设置LabelCtrl属性，即可和一标签控件关联

1. 增加其他控件，列表框控件（WVCheckListBox1），命名为“**clbCustBusinessStr**”
   1. 和工作视图关联

先设置Workview属性为“**Workview**”；

再设置FieldName属性字段为“**业务控制串**”即可。

* 1. 设置SeperatedStr属性字段为“**-**”
  2. 设置ItemDataEntry属性为“**1021**”，即可以从数据字典中取得所有的**业务控制串**

3、在界面上面添加客户基本信息模板

1> 将当前活动的工程设置为“PackageTemp.bpl”（该工程包含所有的客户资料模板）。

2> 在Form上面增加Frame，选择需要的客户基本信息模版**UCustBaseInfo**即可。

### 界面设计

1. 控件**TabOrder**属性的设置
2. 控件**Width，Heigh**t属性的设置
3. 控件**间隔**的设置 & 控件的**合理化摆放**

## 编码

### “确认”键的Enabled属性监控

procedure TfaCtrlCust.ValidChanged (Sender: TWVFieldMonitor; Valid: Boolean);

begin

**inherited;**

btnOK.Enabled := **Valid**;

end;

### 取原来的客户业务控制串

procedure TfaCtrlCust.ValueChanged (WorkField: TWVField);

var

CtrlStr: String;

begin

**inherited;**

CtrlStr := WorkView.FieldByName('客户.业务控制串').Data.Asstring;

WorkView.FieldByName('业务控制串').Data.SetString(CtrlStr);

end;

### 重载“DoIt”方法，实现确认对话框

**protected**

procedure DoIt; **override**;

procedure TfaCtrlCust.DoIt;

var

CheckerNo, CheckerPassword,CustNo,CurrentType : string;

begin

**// 判断“客户的业务控制串”是否被重新设置**

if WorkView.FieldByName ('客户的业务控制串').Data.Asstring =

WorkView.FieldByName ('业务控制串').Data.Asstring then

begin

**Context.GetDialogSystem.ShowWarning** ('业务控制串未做改变！');

Exit;

end;

**// 再次确认对话框显示**

if not DoPrompt ('客户业务控制',Format('客 户 号：%s'#13#10#13#10,

[WorkView. FieldByName ('客户.客户号').Data.Asstring]) + '设置客户业务控制？') then

begin

**SelectFirst**; // 将焦点移到第一个控件项上，**一定要有这行代码**

Exit;

end;

**// 单机双柜员复核**

CustNo := WorkView.FieldByName ('客户.客户号').Data.AsString;

if not **ComCmdDefinition.DoSingleStationCheck** ('客户业务控制',CheckerNo,

CheckerPassword,742,CustNo,CurrentType ) then Exit;

//

**inherited**;

Context.GetDialogSystem.ShowSuccess('修改成功！');

Reset; // 可参看“**TfaInputTemp**”模板里的同名过程

end;

### 字段"客户.客户号"和客户号控件的有效性同步

procedure TfaCtrlCust.CheckValid(WorkField: TWVField);

begin

**inherited;**

WorkField.Valid := **edCustNo.Valid** ;

end;

### 事件的命名

将以上的事件按**事件命名规范命名**

procedure TfaCtrlCust.**ValidChanged** (Sender: TWVFieldMonitor; Valid: Boolean);

procedure TfaCtrlCust.**ValueChanged** (WorkField: TWVField);

procedure TfaCtrlCust.**CheckValid**(WorkField: TWVField);

依次命名为

procedure TfaCtrlCust.**DoUpdateOKBtn** (Sender: TWVFieldMonitor; Valid: Boolean);

procedure TfaCtrlCust.**DoGetBusinessCtrlOnPreBusinessCtrlChanged** (

WorkField: TWVField);

procedure TfaCtrlCust.**DoCheckCustomerValid** (WorkField: TWVField);

## 收尾工作

## 查询模板的使用(例：查询银行对帐标志)

### 准备工作

### 定义菜单

规定菜单名称为“查询银行对帐标志演示”，修改文本形式的菜单文件menus.mnt，最后加密为二进制菜单文件menus.mn。

### 创建包

创建命令包和界面包

1、创建命令包。打开“程序包模板”，选择“PackageTemp2.bpl”（该工程包含所有的合适的模版）。选择菜单[File]|[Save As]，将该模板包另存为Query.dpk，放在“DBDemos”目录下面。该包就是我们创建的命令包。

2、同样方法创建界面包QueryUI.dpk。

3、将命令包和界面包加到“国通50.bpg”程序组中。方法是：选择[Project]|[Add Existing Project]，找到命令包Query.dpk和QueryUI.dpk，分别加入。

### 其他

1. 找到相应的交易编码310432,，复制出输入/输出参数文本，对文本进行格式化。如下表：

功能描述： 查询营业部对应银行代码

相关数据表：

营业部对应银行代码(BRANCH2BANK)

输入参数：

发生营业部 sbranch\_code0

查询营业部 sbranch\_code1 (可以为空)

货币代码 scurrency\_type (可以为空)

银行代码 sbank\_code (可以为空)

职工代码 semp

网卡地址 sstation0

输出参数：

营业部代码 sbranch\_code0

银行代码 sbank\_code

货币代码 scurrency\_type

连通标识 sstatus0

交易对帐标识 sstatus1

非交易对帐标识 sstatus2

开户缺省模板代码 lvol1

设置人员 semp

设置日期 sdate0

设置时间 stime0

目的网关 lvol2

主功能号 lvol3

银行名称 sall\_name [81]

银行类别 lvol4

外围号 sbank\_code2 [7]

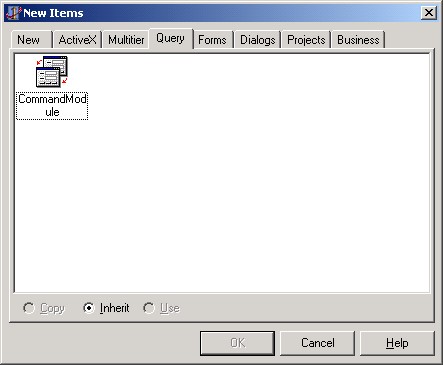
2、将工程包“国通50.bpg”设置为可写的。将“程序包模版”里面的“PackageTemp2.\*” 设置为可写的。

## 设计数据模块

1、打开命令包Query.dpk。

2、在命令包里添加CommandModule模板。方法是：选择菜单[Project]|[Add to Project]，选取“FrameWork”目录下的“kscmdModule.pas”，加入命令包中。

1. 从“CommandModule”继承一个数据模块。选择菜单[File]|[New]，在弹出的界面里找到命令包所在选项卡“Query”，选取“CommandModule”。表示从“CommandModule”开始继承。点击[ok]按钮。这样就新建了一个从“CommandModule”继承的数据模板，将该模板保存并命名为“UCMQueryBankCheckFlag”。



（新建一个数据模块）

4、模块各属性命名。如：将Name命名为“dmQueryBankCheckFlag”。

5、在单元的“Initialization”部分编写代码，用于注册该数据模块。

代码如下：

dmQueryBankCheckFlag := TdmQueryBankCheckFlag.Create(Application);

6、在数据模块上面增加一个命令定义对象(TWVCommandTypeInfo)，并命名为“ctQueryBankCheckFlag”。设置ID,ID设置“查询银行对帐标志演示”。

接着要设置命令参数，即把交易编码里的输入和输出参数加到命令定义对象中。

注意：对于查询型的交易编码，在命令定义对象中添加参数时，不必把输出参数加进去，只需用另加一个“查询结果集”输出参数即可。该“查询结果集”代替了所有输出参数。

具体方法是：选中命令定义对象ctQueryBankCheckFlag。点击右键，选择“Add Command Params”，将交易编码中的输入参数（对于查询类交易编码，不必加输出参数）分别添加到弹出界面的空白框中。（要求：每行表示一命令参数，名称在前，对应Cpack的英文字段在后，中间用空格格开）。交易编码中的三个固定参数“发生营业部”，“职工代码”，“网卡号”不必加进去。

再做手工调整。主要是加上另外三个重要参数，即查询结果集、返回码和返回信息。具体方法：双击命令定义对象ctQueryBankCheckFlag，在弹出的界面里点击界面上方工具栏中的第一个图标(Add New)，即增加了一个参数。在该参数的可视属性编辑器(Object Inspector)里，分别给各属性赋相应值。例如要添加“查询结果集”，paramDataType应设成对象型（kdtObject）,ParamName设为“查询结果集”，ParamType设成输出参数类型（PtOutPut）。添加“返回码”时，paramDataType应设成整数型（kdtInteger）,ParamName设为“返回码”，ParamType设成输出参数类型（PtOutPut）。添加“返回信息”时，ParamDataType应设成字符型(kdtString)，ParamName应设成“返回信息”，ParamType设成输出参数类型(PtOutPut)。

7、在数据模块上面增加一个查询型的命令处理对象(TWVQuery)，并命名为“cqQueryBankCheckFlag”。设置命令处理对象。

设置ID，选取刚才在命令定义对象“ctQueryBankCheckFlag”的ID。只有这样，命令处理对象才和命令定义对象关联起来了（通过ID关联）。

设置RequestType(请求功能号)，即交易编码的代码。在这里，RequestType设为“310432”。

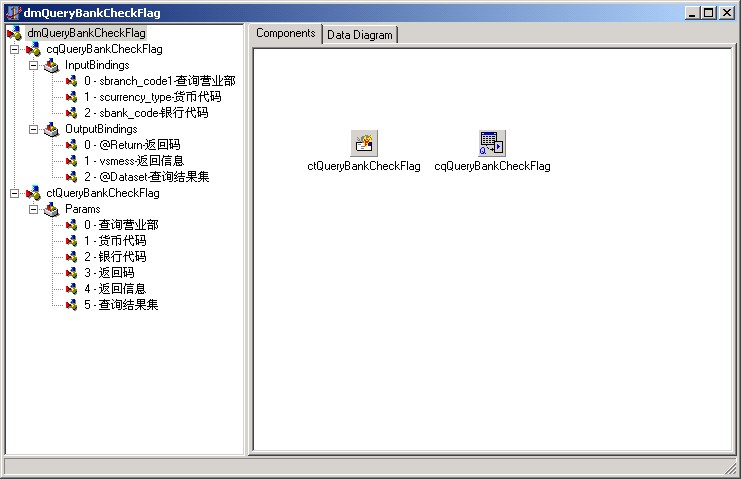
接着，和设置命令定义对象一样，要设置命令参数，即把交易编码里的输入和输出参数加到命令定义对象中。

注意：对于查询型的交易编码，在命令处理对象中添加参数时，和命令定义对象一样，不必把输出参数加进去。只需把“查询结果集”和@DataSet绑定即可。该“查询结果集”代替了所有输出参数。

和命令定义对象设置不同的是：命令处理对象添加输入和输出参数时，需分两步，要把加入输入参数和加入输出参数分开。

第一步是加入输入参数。具体方法是：选中命令定义对象cqQueryBankCheckFlag。点击右键，选择“Add Input Binding”，将交易编码中的输入参数）分别添加到弹出界面的空白框中。（要求：每行表示一命令参数，名称在前，对应Cpack的英文字段在后，中间用空格格开）。交易编码中的三个固定参数“发生营业部”，“职工代码”，“网卡号”不必加进去。

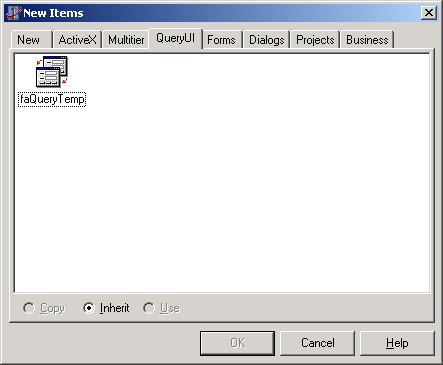
第二步是加入输出参数。在这里，只有“查询结果集”，“返回码”和“返回信息”。不必拷贝交易编码里面的输出参数，而是一个参数一个参数的加。具体方法是：双击命令定义对象cqQueryBankCheckFlag。在弹出的界面里点击界面上方工具栏中的第一个图标(Add New)，即增加了一个参数。在该参数的可视属性编辑器(Object Inspector)里，分别给各属性赋相应值（实际上是将命令定义对象和Cpack中的英文字段绑定起来）。



(创建好的数据模块)

## 设计界面

1. 打开命令包QueryUI.dpk。
2. 在命令包里添加faQueryTemp模板。方法是：选择菜单[Project]|[Add to Project]，选取“Template”目录下的“UQueryTemp.pas”，加入命令包中。
3. 从“faQueryTemp”继承一个数据模块。选择菜单[File]|[New]，在弹出的界面里找到命令包所在选项卡“QueryUI”，选取“faQueryTemp”。表示从“faQueryTemp”开始继承。点击[ok]按钮。这样就新建了一个从“faQueryTemp”继承的数据模板，将该模板保存并命名为“UQueryBankCheckFlag”。



（新建查询模板）

1. 给该模板Frame里的各属性命名。如：将Name命名为“faQueryBankCheckFlag”，

Caption和HelpID设为“查询银行对帐标志演示”。注意，HelpID必须和以后为该界面做的联机帮助文件的文件名相同。

1. 在单元的“Initialization”部分编写代码，创建界面工厂，注册该界面。

程序代码为：

Initialization

TUIStdClassFactory.Create('查询银行对帐标志演示',TfaQueryBankCheckFlag );

在这里，第一个参数是菜单ID,必须与菜单配置文件menus.mnt中定义的一致，

第二个参数是本界面的类名，用“T”加该界面的Name(我们已经将其命名为“faQueryBankCheckFalg”)组成。

另外，需要引用KsFrameworks单元。因此，需在implementation部分加引用语句：

uses KsFrameworks;

1. 为工作视图（WorkView）增加字段。

具体方法是：选中工作视图WorkView。点击右键，选择“Add WorkView Fields”，将交易编码中的输入参数（对于查询类交易编码，不必加输出参数）分别添加到弹出界面的空白框中。（要求：每行表示一命令参数，名称在前，对应Cpack的英文字段在后，中间用空格格开）。交易编码中的三个固定参数“发生营业部”，“职工代码”，“网卡号”不必加进去。

再做手工调整。主要是加上另外三个重要参数，即查询结果集、返回码和返回信息。具体方法：双击工作视图WorkView，在弹出的界面里点击界面上方工具栏中的第一个图标(Add New)，即增加了一个工作字段。在该工作字段的可视属性编辑器(Object Inspector)里，分别给各主要属性赋相应值。例如要添加“查询结果集”，Caption和Name应设成“查询结果集”,DataType设为对象类型“KdtObject”，OwnObject设为True（注意：对于查询结果集这种对象类型的工作字段，一定要设置该属性）。添加“返回码”时，Caption和Name应设为“返回码”,DataType设为整数型“KdtInteger”。添加“返回信息”时，Caption和Name应设为“返回信息”，DataType应设成“KdtString”。



（设计好的工作视图）



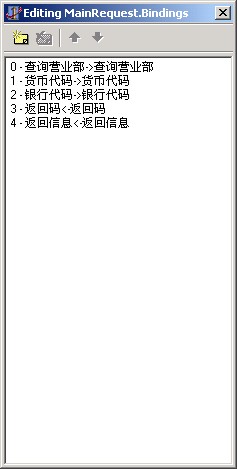
（查询结果集的属性编辑器）

1. 设置命令请求对象。

首先设置ID，在这里，需与命令包里数据模板中命令定义对象的ID一致，为“查询银行对帐演示”。

然后自动绑定一些参数。具体方法是：选中命令请求对象MainRequest，点击右键，选择“Add All Existed Request Bindings”，这样命令请求对象和命令定义对象就关联起来了。

最后手工调整。有可能你在工作视图WorkView中和在命令定义对象中定义的字段名不太相同（系统无法自动绑定）或者你不需要将工作视图里所有字段都与命令定义对象绑定，这时需要手工调整。在我们这个例子里，不需要手工绑定。但你可以检查刚才自动绑定是否完全正确。具体方法是：双击命令请求对象MainRequest，检查绑定。



（设计好的命令请求对象）

1. 设置表格(DbGrid)显示字段。

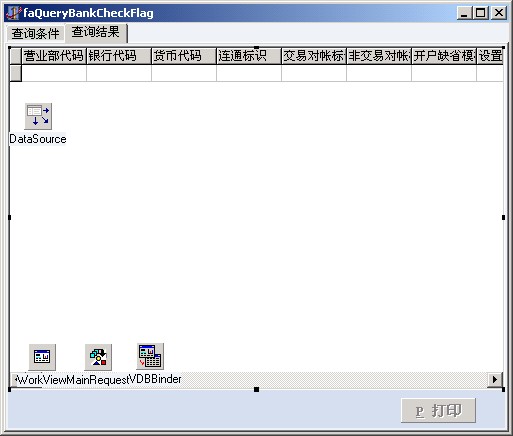
我们可以注意到，在上面的所有操作中，没有具体地将输出的各个字段与界面联系起来。在这里，就是把输出的各个字段与查询结果显示表格(DbGrid)联系起来。

具体方法：

在界面“查询结果”页的选项卡上选中Grid。点击右键，选择“Add Columns”，将交易编码中的输出参数分别添加到弹出界面的空白框中。（要求：每行表示一命令参数，名称在前，对应Cpack的英文字段在后，中间用空格格开）。

然后再手工调整。如果交易编码中有些输出字段不需要显示出来（因有些是供后台自己用）时，你需删除这些字段。具体方法是：双击Grid，在弹出界面中选择要删除的字段，点击上方工具栏中的第二个图标(Delete Selected)。

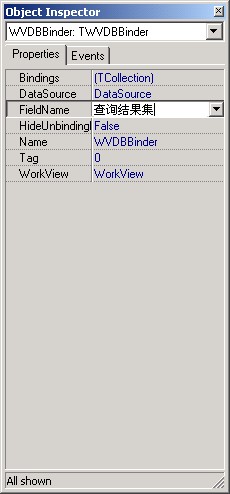
还有些是因为交易编码中字段的命令不太规范，你需要重新命名。具体方法是：双击Grid，在弹出界面中选择该字段，然后在该字段的可视属性编辑器(Object Inspector)中设置Title。即点击“Title”前的“+”号，将Title属性展开。设置Tile下的Caption属性。该属性即在Grid显示的列标题。



（设计好的Grid）

10、设置输出绑定（注意：不要忘了这一步，否则查询结果无法显示出来）。

主要是把工作字段“查询结果集”和WVDBBinder中的显示字段FieldName联系起来。具体方法是：选中输出绑定对象WVDBBinder，在其可视属性编辑器(Object Inspector)中设置FieldName属性，将该属性设置为“查询结果集”。

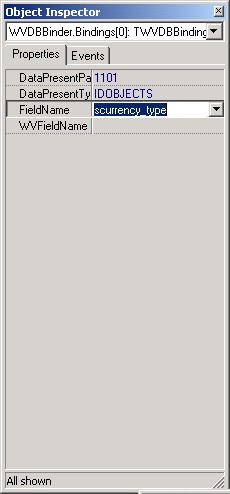


（在输出绑定里设置FieldName为“查询结果集”）

另外，对于一些需要翻译的字段，需要设置其数据字典条，以便于系统翻译。

在这里，以“货币代码”（“币种”）为例。、

具体方法是：双击WVDBBinder，在弹出的界面里点击上方工具栏中的第一个图标(Add New)。然后在该字段的可视属性编辑器(Object Inspector)中设置各属性。如添加“货币代码”的翻译绑定时，需设置DataPresentParm为其数据字典条目“1101”，DataPresentType设为“IDOBJECTS”，FieldName设置为对应交易编码中的英文字段“Scurrency\_Type”。



（设置“货币代码”的输出绑定，以便翻译）

11、在界面里“查询条件”页选项卡上放置各种控件，并将控件和工作视图WorkView中的工作字段关联起来。

以添加“银行代码”为例。

首先，银行代码应该是让用户多选一，因此需用下拉框(TWVCombox)。

选取一个label和一个下拉框控件。Label的caption设为“银行”。下拉框控件（一定要是TWVCombox）的主要属性应该这样设置：WorkView为“WorkView”，FieldName设为对应工作视图的“银行代码”。这样控件和工作视图关联起来了。因为“银行”需从数据字典里取数。从数据字典里取数一定要设置三个属性，DataPresentType、ItemsDataEntry,Style。具体值为：DataPresentType恒定设为“IDOBJECTS”，ItemsDataEntry设为“银行”对应的数据字典条目“\*1601”，星号表示“银行”可以允许为空，Style恒定设为“csDropDownList”。



（“银行代码”的属性编辑器）

1. 编写代码。

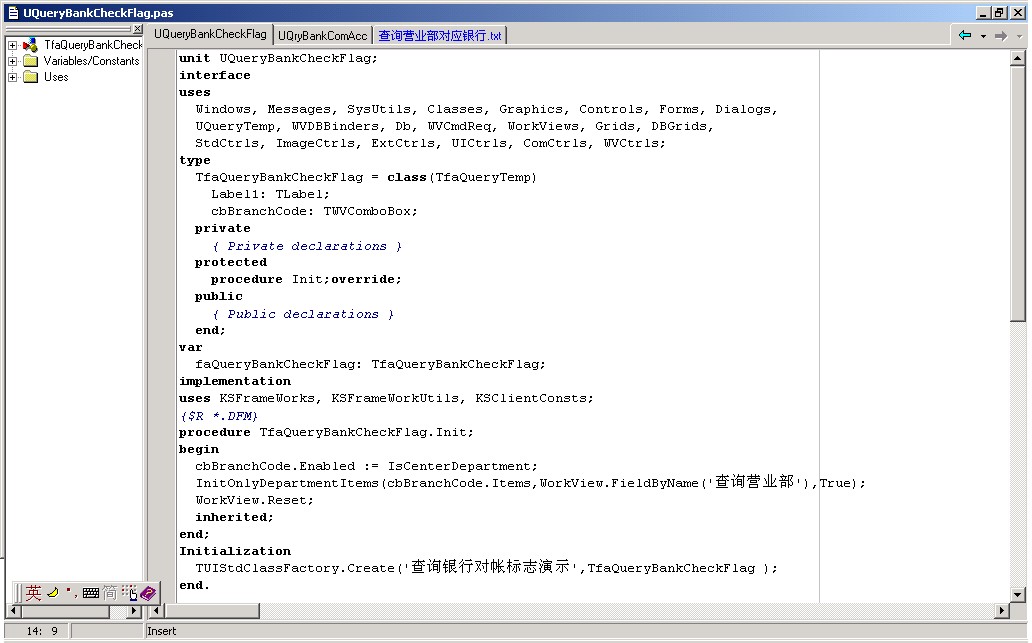
适当编写一些代码。在这里，主要是对“查询营业部”要进行特殊控制。因为，如果是以营业部操作员身份登录，那么不能让操作员选择“查询营业部”，如果是以总部操作员身份登录，则允许其选择营业部。我们需要重载（override）父类的Init方法。在重载的Init加如下代码：

cbBranchCode.Enabled := IsCenterDepartment;

InitOnlyDepartmentItems(cbBranchCode.Items,WorkView.FieldByName('查询营业部'),True);

WorkView.Reset;

其中，InitOnlyDepartMentItems是个公共函数，第一个参数是“查询营业部”对应的下拉框控件的名字（Name），第二个参数是“查询营业部”工作字段，第三个参数表示是否在下拉框中加“所有营业部”一项。



(代码页雏形)

1. 控制和优化。

对界面的一些控制和优化。

* 1. 调整TabOrder顺序和调整各个控件的摆放位置。
  2. 对各工作字段进行有效性控制。设置各工作字段的Checker和Contrained。如果需要用到域(Domain)控制的工作字段则需要设置相应的域。
  3. 监控各工作字段的有效性，并控制“查询”按钮正显或者灰显。

采用的方式是：在工作视图WorkView中，新建一个监控有效性的Monitor。

具体方法是：选中工作视图WorkView，在工作视图WorkView的可视属性编辑器中，选择FieldsMonitors，点击其后面带三个圆点的小按钮。在弹出的界面里，点击上方工具栏中的第一个图标(Add New)，这样就新建了一个Monitor。接着需要设置该Monitor的各属性。在其可视属性编辑器中，设置其说明属性Description，设置为“有效性校验”（此属性并不是必须设置的，但设置了便于以后维护）。然后选择MonitorValidChangedFields，该属性设置的是需要被监控的工作字段。点击该属性后带三个圆点的小按钮，在弹出的界面的多选框(CheckListBox)中选取需要被监控的工作字段。一般选取全部字段。(在这个程序里，实际上我们不需监控，每个字段都会是有效的。) 然后在该Monitor的可视属性编辑器的事件(Event)页选项卡里为OnValidChanged事件编写代码。代码一般如下：

BtnQuery.Enabled:=Valid;

BtnQuery为“查询”按钮的名字(Name)。

## 收尾工作

### 移去模板

分别从命令包和界面包中移去模板。

方法：

1. 选择[File]|[Close All]，将所有当前活动界面均关闭。然后重新打开“国通50.bpg”。

注意：一定要关闭一次，否则下面移去模板时会报错。

1. 打开命令包Query.dpk。选中该包活动窗口中“contains”下的“KsCmdModule”，点击右键，选择[Remove from project]，将“KsCmdModule”模板移去。
2. 用同样方法将界面包QueryUI.dpk里的“UQueryTemp”模板移去。

### 其它

1、将命令包和界面包加入KsClient.ini中，并转换成加密文件。

2、编译、运行。

## 参数设置模板的使用(例：市场设置)

### 准备工作

### 定义菜单

规定菜单名称为“市场设置演示”，修改文本形式的菜单文件menus.mnt，最后加密为二进制菜单文件menus.mn。

### 创建包

新建一个程序包，名字为“ParamsSet.dpk”保存到“系统管理”子目录下面。设置程序包的目标文件路径和引用的程序包。

### 创建从已有模版继承下来的模块

1、将当前活动的工程设置为“PackageTemp.bpl”（该工程包含所有的合适的模版）。

1. 然后选择“New”。在对话框里面的选择“PackageTemp”页，然后选择正确的父类。最后确定。
2. 将新建的文件从PackageTemp.bpl里面删除。然后增加到实际要被包含的程序包ParamsSet.dpk中。

### 修改配置文件及其他

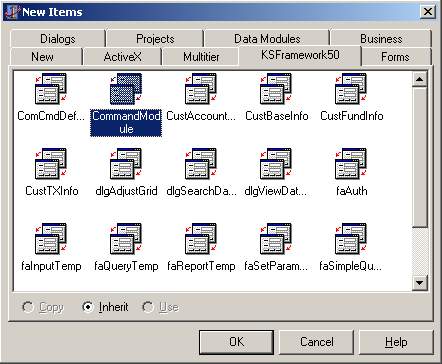
1、修改配置文件ksclient.ini，装载该程序包。

2、将工程包“国通50.bpg”设置为可写的。将“程序包模版”里面的“PackageTemp.\*” 设置为可写的。

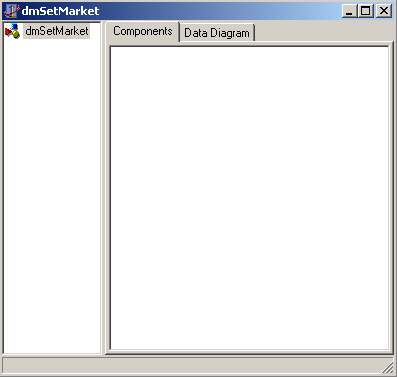
## 设计数据模块

1. 利用前面介绍的方法，在程序包“ParamsSet.dpk”里面新建一个数据模块，从“CommandModule”继承下来，然后保存。文件名为：UCMSetMarket.pas，模块名为：dmSetMarket。

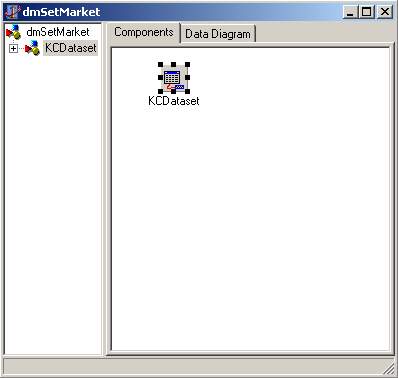
第一步：



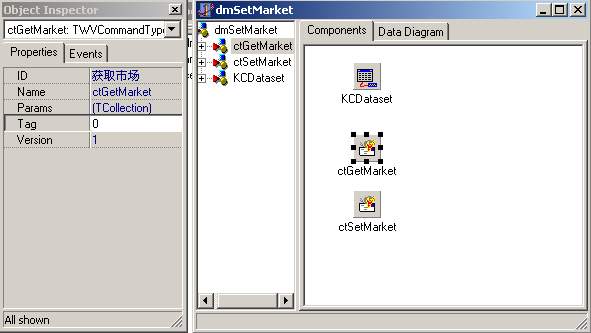
第二步：



1. 在单元的“Initialization”部分编写代码，创建该数据模块。initialization dmSetMarket := TdmSetMarket.Create(Application);
2. 在数据模块上面增加一个数据集对象。



1. 因为我们需要增加，或先查出数据修改，删除，在数据模块上面增加两个命令定义对象ctGetMarket，ctSetMarket。ctGetMarket的ID为“获取市场”，ctSetMarket的ID为“设置市场”。



1. 分析交易编码：此界面需要用到两个交易编码：查询市场 (310417) ，设置市场(310416)，为ctGetMarket增加输出参数：

为ctGetMarket增加参数：

市场代码

返回码

返回信息

查询结果集



为ctSetMarket增加参数：

市场代码

币种

市场名称

交割天数

席位连通标志

回购计息天数

证券账号长度

交易所时差

撤单允许标志

委托允许标志

当前交易日期

当前清算周期

操作类别

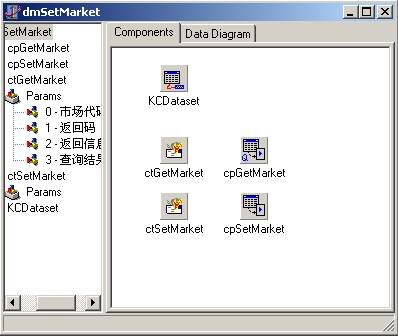
返回码

返回信息



以上增加参数可以用拷贝的方式。

1. 在数据模块上面增加两个命令处理对象cpGetMarket（TWVQuery），cpSetMarket(TWVProicessor)。cpGetMarket的ID为：“获取市场”，RequestType为：310417，cpSetMarket的ID为：“设置市场”，RequestType为：310416，还要设置DataSet属性。



1. 将ctGetMarket，ctSetMarket中定义的参数与送往后台的变量连接，连接在命令处理对象中实现，只要命令处理对象的ID与命令定义对象的ID连接即可。

ctGetMarket输入参数：

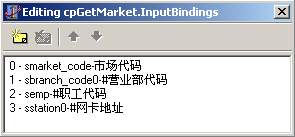
smarket\_code 市场代码

{以下代警号的是系统可取出的信息前台只要连接对即可}

sbranch\_code0 #营业部代码

semp #职工代码

sstation0 #网卡地址



ctGetMarket输出参数：

@return 返回码

vmess 返回信息

@DataSet 查询结果集{交易编码的输出参数不用联结他们会通过数据集返回}



ctSetMarket输入参数：

smarket\_code 市场代码

scurrency\_type 币种

sname 市场名称

lvol1 交割天数

sstatus0 席位连通标志

lvol2 回购计息天数

lvol3 证券账号长度

lvol4 交易所时差

sstatus1 撤单允许标志

sstatus2 委托允许标志

sdate0 当前交易日期

lvol5 当前清算周期

lvol0 操作类别

sbranch\_code0 #营业部代码

semp #职工代码

sstation #网卡地址



ctSetMarket输出参数：

@return 返回码

vmess 返回信息



1. 注意：在设置命令定义对象和命令处理对象的时候，可以利用前面从交易编码里面复制出来的文本，进行批量添加。批量添加以后再逐个调整一下。

## 设计界面

1. 利用前面介绍的方法，在程序包“ParamsSet.dpk”里面新建一个Frame，从“TfaDSetParamTemp”继承下来，然后保存在“系统管理、交易参数、USetMarket” Frame的名字为faSetMarket。在单元的“Initialization”部分编写代码TUIStdClassFactory.Create('市场设置演示',TfaSetMarket);，创建界面工厂，注册该界面。

TfaDSetParamTemp是一个两页的界面，第一页显示查询的数据，第二页是增加、修改或删除已查询出的数据。必须在第一页放置可以输入查询条件的控件，例如查询条件为“查询.市场代码”可放置一数据输入控件将条件输入给“查询.市场代码”，因为本例数据量很小因而此查询条件可以为空，就是说不用添加输入控件直接点击查询按钮即可。第一页上有一个表格和五个按钮，表格是用来显示查询出来的数据用的，细节按钮是将表格中的数据的每一项显示在第二页的对应的数据输入控件内，这样看起来比较清晰，修改按钮和删除按钮也是对查询出来的数据进行操作，显然如果数据没有查到这单个按钮应不能工作，无论有无查询数据查询按钮和增加按钮是可以工作的。对于某些菜单功能营业部只能对本营业部数据查询而不能进行增加删除和修改，为实现此功能TfaDSetParamTemp提供一个工作模式的概念即总部模式和营业部模式，总部模式还分能否操作其他营业部两种模式。对于我们的界面可以不设置因为TfaDSetParamTemp默认为总部模式。

1. 为工作视图（WorkView）增加字段，一部分来自交易编码，另外一部分是客户信息（打开dmSetMarket）。

查询.市场代码 //此参数将和ctGetMarket的“市场代码”连接

查询结果集 //查询市场返回的结果。DataType为：kdtObject；OwnObject为：True.

以下是输入的参数，他们的DataField必须与查询市场 (310417)交易编码的输出参数变量对应，查询出来的数据会通过DataField到WorkView中。

字段名 DataField GroupIndex

市场代码 smarket\_code 1

币种 scurrency\_type 1

市场名称 sname 1

交割天数 lvol1 1

席位连通标志 sstatus0 1

回购计息天数 lvol2 1

证券账号长度 lvol3 1

交易所时差 lvol4 1

撤单允许标志 sstatus1 1

委托允许标志 sstatus2 1

当前交易日期 sdate0 1

当前清算周期 lvol5 1

操作类别 0



1. 设置命令请求对象。先是ID，然后自动绑定一些参数，最后手工调整。命令请求对象由4个QueryRequest查询请求，AddRequest 增加请求DeleteRequest删除请求 ChangeRequest修改请求，对于后3个命令请求对象我们设置他们的ID为“设置市场”怎么区分呢？因为设置市场(310416) 交易编码中有一输入参数“操作类别---1增加，2修改，3删除”，我们在调用这3命令请求对象之前将“操作类别”赋上不同的值即可，具体实现方法后面会有例子。

对象 ID

QueryRequest 获得市场

QueryRequest的数据绑定：

查询.市场代码 ->市场代码 //输入

查询结果集<-查询结果集 //输出



对象 ID

AddRequest 设置市场

AddRequest的数据绑定：

市场代码 ->市场代码

币种 ->币种

市场名称 ->市场名称

交割天数 ->交割天数

席位连通标志 ->席位连通标志

回购计息天数 ->回购计息天数

证券账号长度 ->证券账号长度

交易所时差 ->交易所时差

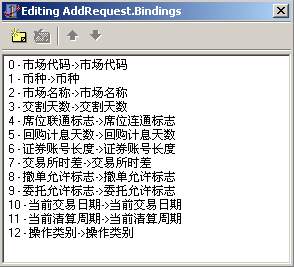
撤单允许标志 ->撤单允许标志

委托允许标志 ->委托允许标志

当前交易日期 ->当前交易日期

当前清算周期 ->当前清算周期

操作类别->操作类别



对象 ID

ChangeRequest 设置市场

ChangeRequest的数据绑定：

市场代码 ->市场代码

币种 ->币种

市场名称 ->市场名称

交割天数 ->交割天数

席位连通标志 ->席位连通标志

回购计息天数 ->回购计息天数

证券账号长度 ->证券账号长度

交易所时差->交易所时差

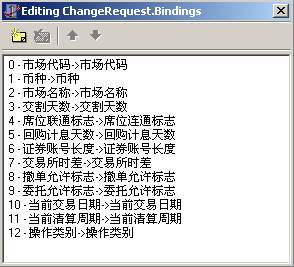
撤单允许标志 ->撤单允许标志

委托允许标志 ->委托允许标志

当前交易日期 ->当前交易日期

当前清算周期 ->当前清算周期

操作类别 ->操作类别



对象 ID

DeleteRequest 设置市场

DeleteRequest的数据绑定：

市场代码 ->市场代码

币种 ->币种

市场名称 ->市场名称

交割天数 ->交割天数

席位连通标志 ->席位连通标志

回购计息天数 ->回购计息天数

证券账号长度 ->证券账号长度

交易所时差->交易所时差

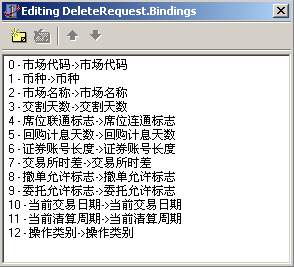
撤单允许标志 ->撤单允许标志

委托允许标志 ->委托允许标志

当前交易日期 ->当前交易日期

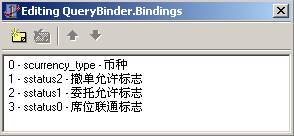
当前清算周期 ->当前清算周期

操作类别 ->操作类别



通过以上的连接我们可以将数据与命令连接在一起。

设置QueryBinder的FieldName为“查询结果集”，可以将“查询市场”的数据显示在表格中， 设置WVDataSource的GroupIndex为：1系统便会faSetMarket便会将“查询市场”的数据写入WorkView中GroupIndex为1的字段便于修改和删除。在QueryBinder中增加表格中需要显示中文意义的“查询市场”的数据的字段，如：Scureency\_Type的DataPresenType为IDOBJECTS，DataPresentParam为1101，那么显示数据时01将被显示为“人民币”，02将被显示为“港币”，03将被显示为“美元”。



1. 增加其他控件，设置他们的WorkView属性和工作视图关联起来，设置他们FielName为WorkView中的字段就可以将用户的输入写入WorkView中的字段。为了实现在这些控件回车/\*可以切换倒下一个/上一个控件，需要设置与这些控见相连的WorkView中的字段的GoNextWhenPressEnter 为True，GoPrevCahr为\*。
2. 设置QueryRequest，AddRequest，ChangeRequest，DeleteReques的AfterExec为CheckReturnData对返回数据进行验证。

### 编写程序

1. 设置默认值

对此界面上出现的几个输入项：

市场代码 ，市场名称，货币代码，交割天数，回购计息天数，需要默认值。对于交割天数，回购计息天数可在属性中直接设置为0，对市场代码，货币代码需要在faSetMarket的Init函数中实现，重载Init如下：

procedure TfaSetMarket.Init;

begin

inherited;

WorkView.FieldByName('市场代码').DefaultValue := GetFirstItemID(deMarket);

WorkView.FieldByName('币种').DefaultValue := GetFirstItemID(deCurrency);

WorkView.Reset;

UpdateButton;

end;

对于“市场名称”需要在字段“市场代码”的OnValueChanged事件中写

inherited;

if WorkField.Valid and (WorkField.Data.AsString <> '') then

WorkView.FieldByName('市场名称

').Data.SetString(GetIdObjectName(deMarket,WorkField.Data.AsString));

实现增加，删除，和修改字段“操作类别”有不同的值

重载BeforeAdd函数

procedure TfaSetMarket.BeforeAdd;

begin

inherited;

WorkView.FieldByName(‘操作类别’).Data.SetInteger(1);

end;

重载BeforeDelete函数

procedure TfaSetMarket. BeforeDelete;

begin

inherited;

WorkView.FieldByName(‘操作类别’).Data.SetInteger(3);

end;

重载BeforeChange函数

procedure TfaSetMarket.BeforeBeforeChange;

begin

inherited;

WorkView.FieldByName(‘操作类别’).Data.SetInteger(2);

end;

实现修改时不能修改关键字段而增加时可以：

重载BeforeChange函数

procedure TfaSetMarket.BeforeBeforeChange;

begin

inherited;

Controler.Enabled := False;

end;

重载BeforeAdd函数

procedure TfaSetMarket.BeforeBeforeAdd;

begin

inherited;

Controler.Enabled := True;

end;

2、有效性验证

对必输的数据必须设置Constrianed为True，有效性验证是通过设置WorkView中字段的Checker属性或设置DomainName来达到的。

以下列出界面上将要验证的字段的设置：

字段名 Constrained DomainName 长度

市场代码 True 1

市场名称 True 20

货币代码 True 2

交割天数 True 正整数

回购计息天数 True 正整数

交易所时差 False

证券账号长度 False

席位连通标志 True 1

撤单允许标志 True 1

委托允许标志 True 1

当前交易日期 False

当前清算周期 False

操作类别 False

设置字段的Hint表明这些字段的输入格式。设置字段的OnCheckValid为CheckEditFieldValid后，在程序运行时在接受用户输入的界面上会显示当前输入无效的数据，而在浏览界面或删除界面这些无效的数据巨响是不会被提示出错的，表示可以操作。

现在设置数据输入无效时确认按钮无效。

为WorkView的FieldsMonitors增加一字段并设置此字段的MonitorValidFieldsChanged为“|市场代码|币种|市场名称|交割天数|席位联通标志|回购计息天数|撤单允许标志|委托允许标志|”，再为此字段的OnCheckValid事件增加

btnOK.Enabled := Valid;Valid的值为“市场代码|币种|市场名称|交割天数|席位联通标志|回购计息天数|撤单允许标志|委托允许标志”内所指WorkView中字段的有效性的逻辑“与”，弱国所有字段有效确认按钮就能工作。

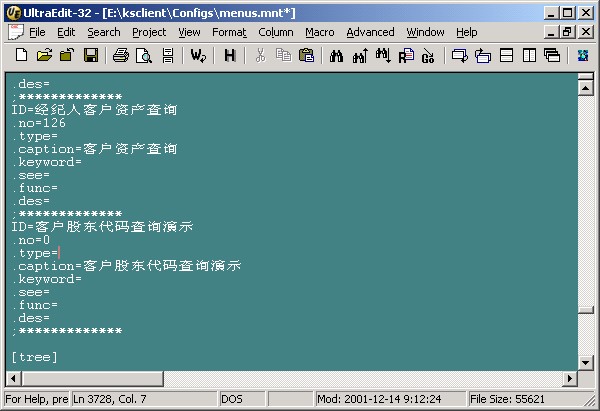
### 收尾工作

1、编译、运行。

## 客户信息模板的使用(例：客户股东代码查询)

### 准备工作

1. 定义菜单：规定菜单名称为“客户股东代码查询演示”，在configs目录下修改文本形式的菜单文件menus.mnt,如下图，最后加密为二进制菜单文件menus.mn。



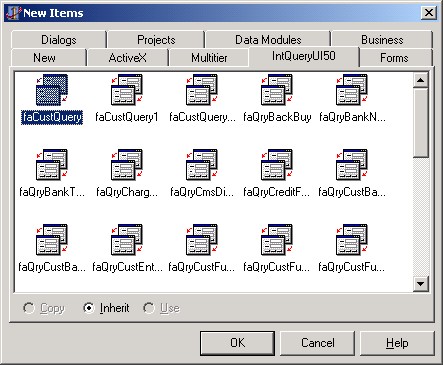
并在下面的营业部菜单中(或者总部)添加菜单名



2、将工程包“国通50.bpg”和综合查询的界面包和PackageTemp包设置为可写的。

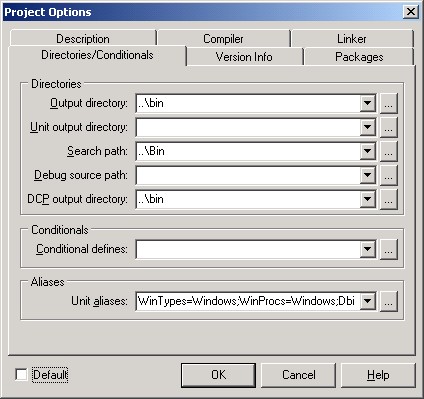
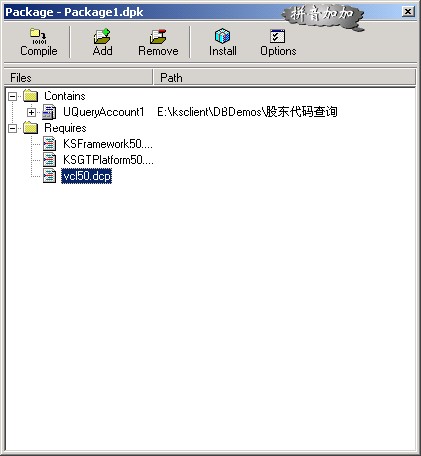
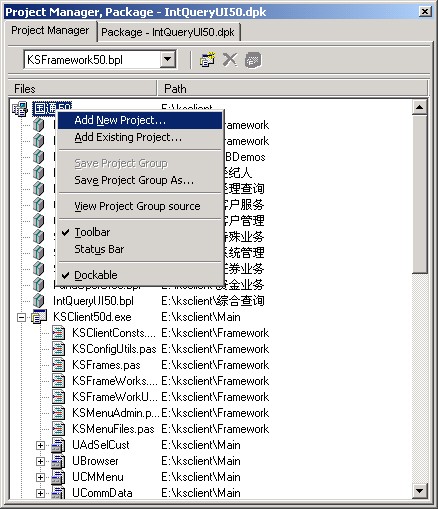
1) 将当前活动的工程设置为“IntQueryUI50.bpl”（该工程包含所有的合适的模版CustQueryTemp）。

2) 然后选择“New”。在对话框里面的选择“IntQueryUI50”页，然后选择faCustQuery。最后确定。

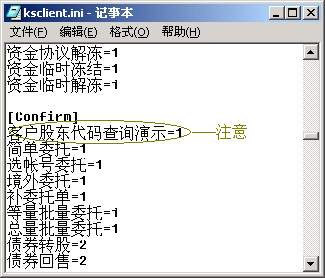
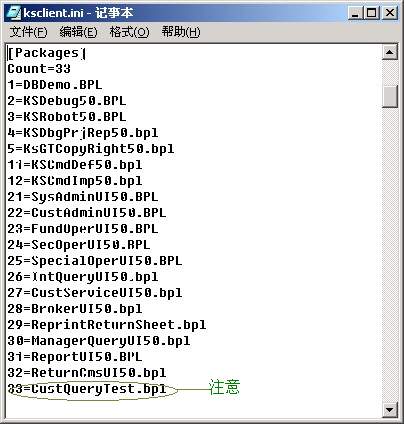


3) 将新建的模块保存，安装命名规范定义文件名和模块名。

3、创建程序包：新建一个程序包，如下图名字为“CustQueryTest.dpk”保存到“DBDemos”子目录下面。设置程序包的目标文件路径和引用的程序包。



4、修改配置文件ksclient.ini，装载该程序包，并且在Comfirm部分增加一个确认的条目。最后加密为二进制配置文件ksclient.in。



5、找到相应的交易编码，复制出输入/输出参数文本，对文本进行格式化。

相关的交易编码:100306, 界面应该包括的输入输出项目定义如下:

|  |
| --- |
| 功能描述： 取得股东代码资料 |
| 相关数据表：  HOLDER\_ACC |
| 输入参数：  客户号 scust\_no  市场代码 smarket\_code ‘’=查询所有市场，否则查询指定市场  主代码标志 lvol0 1=查询所有代码，2=只查询主代码  营业部代码 sbranch\_code0  股东代码 sholder\_ac\_no[17]  职工代码 semp  网卡号 sstation0 |
| 输出参数：  客户号 scust\_no  市场代码 smarket\_code  股东代码 sholder\_ac\_no  报盘代码 sholder\_ac\_no2  主代码标志 smain\_flag  指定标志 smain\_flag2  指定席位 sstock\_code  报盘席位 sstock\_code2  回购指定 sstatus0  股东代码帐户类别 sholder\_type  股东代码状态 sstatus1  代码姓名 sname  代码级别 lvol0  证件类型 lcert\_type  证件号码 scert\_no |

复制出输入/输出参数文本，对文本进行格式化。

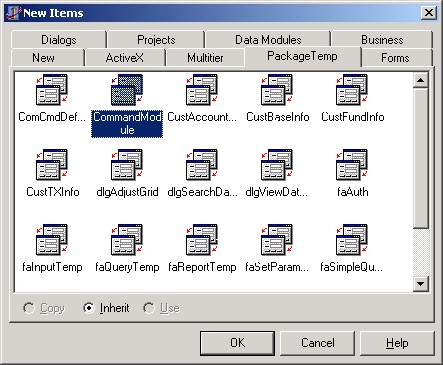
6、涉及的公共函数是:GetSecAccounts 在KSComCmdDefs单元中

procedure GetSecAccounts(const CustomerNo : string; const Market : string; OnlyMainAccounts : Boolean; SecAccounts : TObjectList); 根据客户号获取股东代码

### 设计数据模块

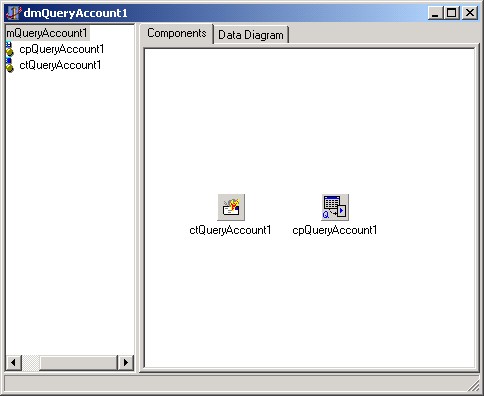
1、首先要从PackTemp中继承数据模型,先激活PackTemp,点选新建按钮,在PackTemp页中选择Command Module。

如图:

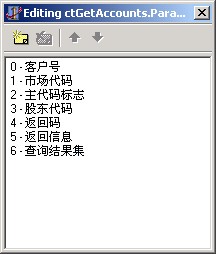


2、在数据模块中放置命令定义对象和命令实现对象。

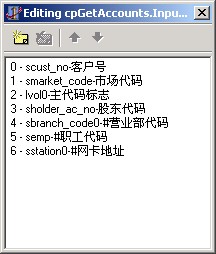
如图:



3、设置命令定义对象的ID为 “股东代码查询”,选中命令定义对象,右键在弹出菜单中选择Add Command Params,批量添加输入参数,输出参数以及返回码,返回信息和查询结果集。如图:



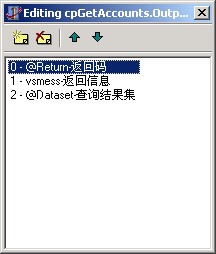
4、设置命令实现对象的ID下拉选择 “股东代码查询”,设置RequestType属性为100306,选中命令实现对象,右键在弹出菜单中选择Add Input Bindings,批量添加输入参数,对于网卡号,职工代码和营业部代码系统底层有默认设置,在字段的Param Name中选择,分别为:#网卡地址,#职工代码,#营业部代码全部设置完成以后,如图:



再次选中命令实现对象,右键在弹出菜单中选择Add Output Bindings,批量添加输出参数,对于本例来说输入的参数是放在查询结果集中的,所以只要添加查询结果集,返回码和返回信息就可以了,注意设置其字段的Param Name和FieldName ,分别是

返回信息   vsmess  
返回码          @Return  
查询结果集   @Dataset

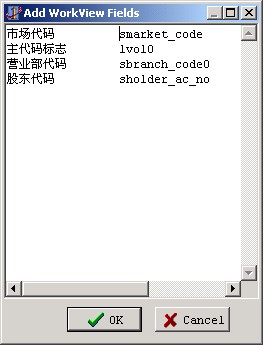
如图:



### 定义界面

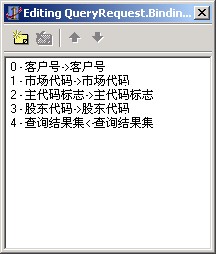
1、为工作视图增加字段，来自交易编码，另外客户号和查询结果集是模板里继承过来的。

选择WorkView,右键在弹出菜单中选择Add WorkView Fields,将输入参数拷贝进去,具体做法如图:



2、设置命令请求对象。先是ID，设为股东代码查询,然后设置WorkView属性,右键选择Add All Exited Request Bindings自动绑定一些参数，最后手工调整。

如图:



### 设计界面

1、利用前面介绍的方法，在程序包“Package1.dpk”里面新建一个Frame，从“TfaCustQuery”继承下来，然后保存。在单元的“Initialization”部分编写代码，创建界面工厂，注册该界面。

TCustQueryFactory.Create('客户股东代码查询演示', TfaCustQuery1,'股东代码查询演示');

值得注意的是最后一个参数是和嵌入式页面相关,参见[本例注意点](#_本例注意点)。

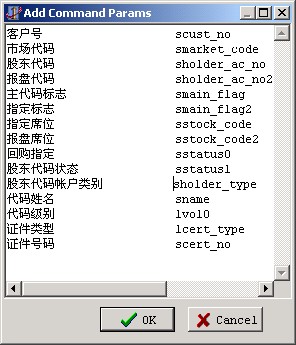
2、增加其他控件，和工作视图关联起来，设置字段。

注意:设置控件的WorkView,FieldName,如显示市场的控件的FieldName为“市场代码”

本例中三个输入框都是下拉框,还要设置控件的style为csDropDownList,SeperatedStr属性为“-”,因为市场代码是从数据字典里取到的,所以对于显示市场的下拉框还要特别设置,设置其DataPresentType属性为IDOBJECTS。

3、添加DBGrid中的输出字段。

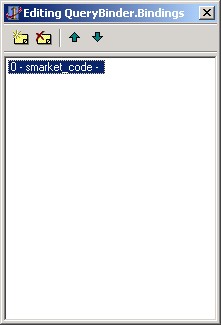
选中DBGrid,右键在弹出菜单中选择Add Columns,可以批量添加输出字段,将输出参数拷贝进去,如图:



4、对DBGrid中的输出字段进行绑定翻译中文。

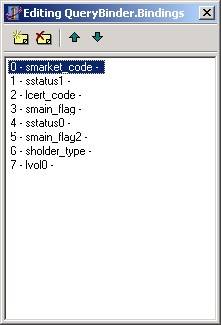
例如,市场字段绑定设置,双击QueryBinder,在弹出的编辑界面中新建一个字段项目,设置该字段项目的FieldName为smarket\_code(根据交易编码而来),在Data Present Type属性设置为IDOBJECTS,Data Present Params设为1301(数据字典中市场的编号)

如图:



同理设置股东代码状态, 股东代码帐户类别, 主代码标志,指定标志, 回购指定, 证件类别, 代码级别

选中DBGrid,右键在弹出菜单中选择Add Columns,可以批量添加输出字段,将输出参数拷贝进去,如图:



5、全部完成界面设计后如图:



### 编码

1、将框架UCustQueryTemp中的客户号输入控件的有效性传入界面中并控制'查询'的有效性。

Procedure TfaQueryCustAccount.DoCheckQueryBtn(Sender:TWVFieldMonitor; Valid: Boolean);

begin

inherited;

InputValidChanged(Valid);

end;

2、对返回码和返回信息进行检验。

procedure TfaQueryCustAccount.QueryRequestAfterExec(Request: TWVRequest;

Command: TWVCommand);

begin

inherited;

CheckCommandReturnValue(Command);

end;

3、根据客户号和市场代码取得股东代码。GetSecAccounts函数在KSComCmdDefs中。

ProcedureTfaQueryCustAccount.DoGetAccountOnMarketAndCustChanged(Sender: TWVFieldMonitor; Valid: Boolean);

var

Acc: TObjectList;

CustNoField,MarketField,AccountNoField:TWVField;

begin

CustNoField:=WorkView.FieldByName('客户号');

MarketField:=WorkView.FieldByName('市场代码');

AccountNoField:=WorkView.FieldByName('股东代码');

inherited;

if Valid and (CustNoField.Data.AsString<>'') and (MarketField.Data.AsString<>'') then

begin

Acc := TObjectList.Create;

cbstockholdnum.Enabled :=True;

AccountNoField.Data.Clear;

try ComCmdDefinition.GetSecAccounts(CustNoField.Data.AsString,MarketField.Data.AsString ,False,Acc);

cbstockholdnum.Items.Clear;

GetObjectsItems(cbstockholdnum.Items,Acc);

cbstockholdnum.Items.Insert(0,'-');

cbstockholdnum.ItemIndex:=0;

finally

Acc.Free;

end;

end

else

begin

cbstockholdnum.Items.Clear;

AccountNoField.Data.Clear;

cbstockholdnum.Items.Insert(0,'-');

cbstockholdnum.ItemIndex:=0;

cbstockholdnum.Enabled :=False;

end;

end;

3、继承模板的Init过程,重置WorkView。

procedure TfaQueryCustAccount.Init(const AUICode: string);

begin

inherited;

Workview.Reset;

end;

### 收尾工作

### 说明

因为客户资料查询是一种嵌入式的页面,他要和其他的嵌入式界面一起在客户综合信息查询界面中显示,所以别忘了在IntQueryUI50包里的UcustIntQuery.pas单元里要对本界面进行注册,修改PageCount=10,在实现的uses中加入界面所在的单元UQueryAccount1 另外

CustQueryPageClasses[9]:= TfaCustQuery1;

UICodes[9]:='股东代码查询演示';

Captions[9]:='股东代码查询演示';

另外还要修改UcustQueryFrame的TCustQueryFactory.CreateInstance函数。

### 小结

涉及到的技巧: 。

根据客户号和市场取得股东代码，填充到下拉框或者多选框。

使用下拉框方法。“-”“,”等等。

在表格里面根据数据字典翻译。

工作字段的缺省值。

控制市场的不选的时候，限制股东代码的输入。

如何增加一个菜单。

如何增加一个新界面。

有关单个客户的资料查询。

根据数据字典填充列表。

## 报表模板的使用（例：经营情况综合报表1）

### 准备工作

### 准备开发环境

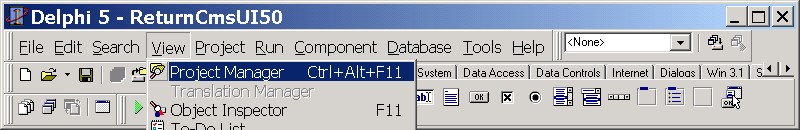
进行开发必须首先建立开发环境，这部分工作和具体的交易编码和编程实现无关，只是为了准备编程和调试运行的环境。

1、确保安装了Delphi和补丁1，安装了设计时包并正确的设置了path，需要安装的设置时包和需要设置的路径请参考相关的文档，在此不多做说明了。

2、确保已经创建好了工作目录（假设就叫做KSClient），并从版本管理工具（SourceSafe）获得了源代码和支持文件，基本的子目录至少包括：bin目录(可执行文件的输出目录),config目录（包含配置文件和菜单文件），FrameWork目录（包含框架程序），Templetas目录（包含模板），这些目录都是工作目录下的一级子目录。

3、在创建好的工作目录（假设就叫做KSClient），找到一个叫做国通50.bpg的文件，双击打开该文件，这是一个工程组文件，包含多个可执行文件和delphi的包（bpl）文件。

双击该文件后会自动启动delphi,然后选择deiphi的主菜单的view目录下的Project Manager(见下图)，或者使用快捷键‘Ctrl+Alt+F11’。



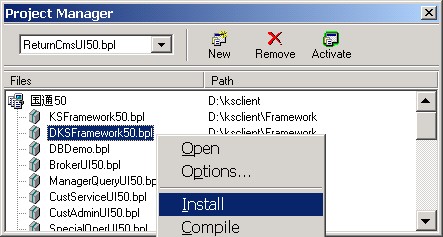
然后就会出现项目管理器Project Manager(见下图)。



在上图中出现的五个项目一般是必需的。KSClient50.exe是主程序，SysFileMan50c.exe是用来转换菜单文件和配置文件的可执行程序，KSFramework50.bpl是运行时使用的框架程序包，DKSFramework50.bpl是设计时使用的框架程序包，PackageTemp.bpl是设计时使用的模板程序包。

在保证这些文件都存在的前提条件下，选择deiphi的主菜单的Project目录下的Build All Projects，确保所有的程序都能够编译通过。

如果所有的程序都能够编译通过，下面应该安装(install)设计时框架包DKSFramework50.bpl(见下图)。



安装成功后会显示成功信息（如下图），表示包成功的安装好了。



4、如果所有的程序都能够编译通过，下面转到你的工作目录的Config子目录，看看没有没有ksclient.ini和ksclient.in两个文件（见下图），ksclient.ini是配置文件原文件，ksclient.in是经过转换的配置文件。

然后同样在这个目录内，看看没有没有munes.mns和munes.mn两个文件（见下图）。munes.mns是菜单原文件，munes.mn是经过转换的菜单文件。

5、在确保上述文件都存在的前提条件下，转到工作目录的bin子目录下，找到文件KSClient50.exe，双击启动程序(见下图)。如果能够启动成功，说明调试环境基本正常，可以进行下一步的开发和调试的工作。

6、以上的工作是为了确保调试环境可用，下面开始确保开发环境可以用。首先要求将国通50.bpg和 PackageTemp.bpk设置为可以修改，国通50.bpg在工作目录下，PackageTemp.bpk放在工作目录的‘程序包模板’子目录下。

7、如果以上的工作都能顺利的完成，就说明开发环境和调试环境基本无误，可以进行下一步的操作了。

### 分析业务需求

环境准备好以后，就可以开始分析业务需求，确定需要使用的交易编码，并根据业务需求选择模板并考虑界面需要进行的控制了。

经营情况综合报表的目的是生成证监会报表中的‘经营情况综合报表’,报表格式如下。

打印格式：

证券公司营业网点经营情况统计表

填报单位： 会员代码-深： 沪：

证券营业部名称： 营业部编码： 报告期： 年 月

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 项目名称 | | 本期买入 | | | | | | | 本期卖出 | | | | |
| 机构 | | | 个人 | | 合计 | | 机构 | 个人 | | 合计 | |
| A股 |  | | |  | |  | |  |  | |  | |
| 基金 |  | | |  | |  | |  |  | |  | |
| 国债 |  | | |  | |  | |  |  | |  | |
| 企业债 |  | | |  | |  | |  |  | |  | |
| 回购 |  | | |  | |  | |  |  | |  | |
| 新股认购 |  | | |  | |  | |  |  | |  | |
| 新股中签 |  | | |  | |  | |  |  | |  | |
| 其他 |  | | |  | |  | |  |  | |  | |
| 项目名称 | 个人 | | | | | 机构 | | | 合计 | | | |
| 营业部开设客户资金帐户总数（户） | | 本期开户 | |  | | |  | | |  | | |
| 本期销户 | |  | | |  | | |  | | |
| 期末户数 | |  | | |  | | |  | | |
| 上交所指定交易总户数（户） | | 本期指定 | |  | | |  | | |  | | |
| 本期撤消 | |  | | |  | | |  | | |
| 期末户数 | |  | | |  | | |  | | |
| 客户保证金 | | 本期存入 | |  | | |  | | |  | | |
| 本期取出 | |  | | |  | | |  | | |
| 本期余额 | |  | | |  | | |  | | |
| 期末网上新股认购未解冻资金 |  | | | | |  | | |  | | | |
| 托管股份总市值（A股+基金） |  | | | | |  | | |  | | | |
| 各种形式受托资产合计 |  | | | | | | | | | | | |
| 本期营业收入 | | |  | 本期营业支出 | | | | |  | | | |
| 其中： | | |  | 其中 | | | | |  | | | |
| 手续费收入 | | |  | 营业费用 | | | | |  | | | |
| 利息收入 | | |  | 利息支出 | | | | |  | | | |
| 资产管理收入 | | |  | 营业税金及附加 | | | | |  | | | |
| 其他业务收入 | | |  | 其他业务支出 | | | | |  | | | |
| 营业外收入 | | |  | 营业外支出 |  | | | | 净利润 | | |  |

负责人： 制表人： 联系电话： 报出日期： 年 月 日

整个报表分为两大部分，上半页表示本期的买入和卖出，按照不同的证券类别进行统计。下半页表示其他的相关信息。

为了生成该报表，需要给报表格式文件提供两个数据集，一个表示报表上半页的数据另一个表示报表下半页的数据，使用了两个交易编码，分别对应上面的两个部分。

交易编码220503 (经营情况综合报表1)用来取的报表上半页的数据

输入参数：

开户营业部 sbranch\_code1[4]

日期（月份） sdate0[9]

货币类型 scurrency\_type[3] 传’01’

职工所属营业部代码 sbranch\_code0[4]

职工代码 semp[7]

网卡号 sstation0[17]

输出参数：

证券统计类别 sstat\_type[3]

机构买入金额 damt0

个人买入金额 damt1

合计买入金额 damt2

机构卖出金额 damt3

个人卖出金额 damt4

合计卖出金额 damt5

交易编码220504 (经营情况综合报表2)用来取的报表下半页的数据

输入参数：

开户营业部 sbranch\_code1[4]

日期（月份） sdate0[9]

货币类型 scurrency\_type[3] 传’01’

职工所属营业部代码 sbranch\_code0[4]

职工代码 semp[7]

网卡号 sstation0[17]

输出参数：

本期开户个人 lvol0

本期开户机构 lvol1

本期开户合计 lvol2

本期销户个人 lvo3

本期销户机构 lvol4

本期销户合计 lvol5

期末户数个人 lvol6

期末户数机构 lvol7

期末户数合计 lvol8

本期指定个人 lvol9

本期指定机构 lvol10

本期指定合计 lvol11

本期撤销个人 lvol12

本期撤销机构 lcert\_code

本期撤销合计 lsafe\_level

期末户数个人 lsafe\_level2

期末户数机构 lbank\_acc\_type

期末户数合计 lbank\_acc\_type2

本期存入个人 damt0

本期存入机构 damt1

本期存入合计 damt2

本期取出个人 damt3

本期取出机构 damt4

本期取出合计 damt5

期末余额个人 damt6

期末余额机构 damt7

期末余额合计 damt8

期末新股认购未解冻个人 damt9

期末新股认购未解冻机构 damt10

期末新股认购未解冻合计 damt11

托管股份总市值个人 damt12

托管股份总市值机构 damt13

托管股份总市值合计 damt14

手续费收入 damt15

分析这两个交易编码可以发现，这两个交易编码的输入字段完全相同，他们的功能也基本相同，都是输入一定的条件数据来获得数据集。

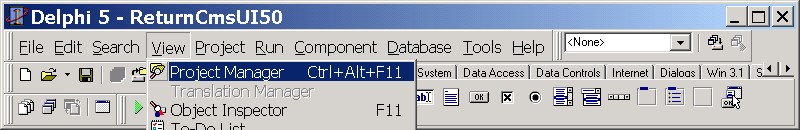
通过这些分析，我们可以确定将要使用报表模板作为界面程序的基类，该界面需要调用两个交易编码。并且我们需要按照报表规定的格式通过报表生成器制作一个报表格式文件（rpt文件）。

### 创建从已有模版继承下来的模块

完成了业务需求的分析以后，可以开始创建界面了。

1、在创建好的工作目录（假设就叫做KSClient），找到一个叫做国通50.bpg的文件，双击打开该文件，这是一个工程组文件，包含多个可执行文件和delphi的包（bpl）文件。

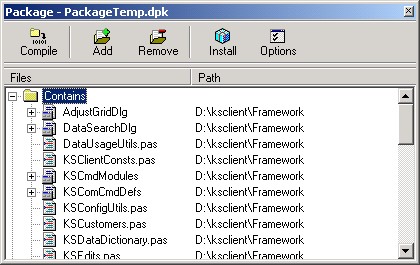
双击该文件后会自动启动delphi,然后选择deiphi的主菜单的view目录下的Project Manager(见下图)，或者使用快捷键‘Ctrl+Alt+F11’。



然后就会出现项目管理器Project Manager(见下图)。



双击显示框中的“PackageTemp.bpl”，将当前活动的工程设置为“PackageTemp.bpl”（该工程包含所有的可能要从此继承下来的模版）。然后就会显示包的说明界面（见下图）。

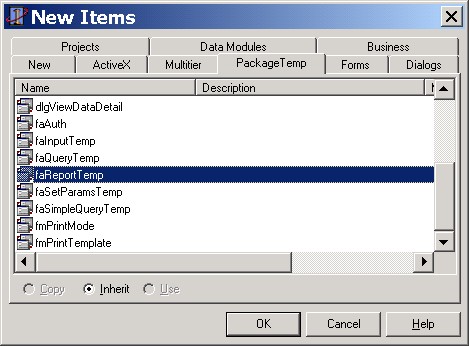


2、然后选择Deiphi的主菜单的“File”目录下的“New”(见下图)。



按下“New”出现Delphi的“Object Repository”。

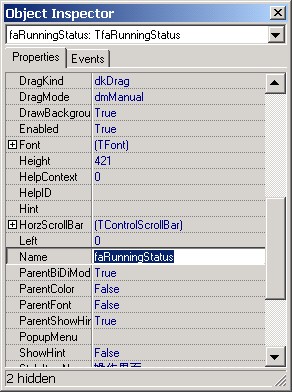
选择“PackageTemp”页的‘faReportTemp’项目，并单击‘OK’键。（见下图）



然后就会出现从报表模板继承下来的界面（如下图）。



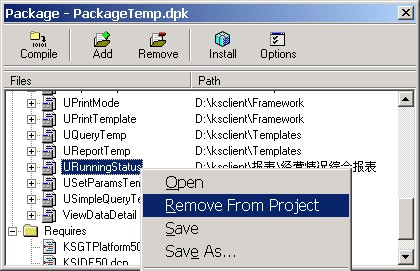
在Delphi的Object Inspector中将界面的Name改成faRunningStatus（见下图）



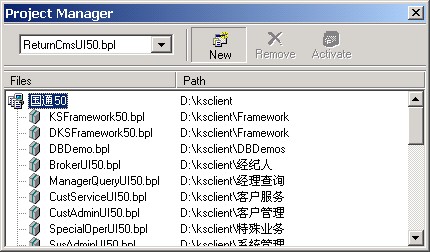
然后需要将界面程序保存到目录中去，由于该界面属于报表，所以就在工作目录下建立一个子目录，命名为报表。然后再在报表目录下建立一个子目录，命名为经营情况综合报表。将界面程序保存到该目录下，并且按照命名规范定义文件名为UrunningStatus。

Delphi会自动将pas文件和同名的dfm文件保存到该目录下。

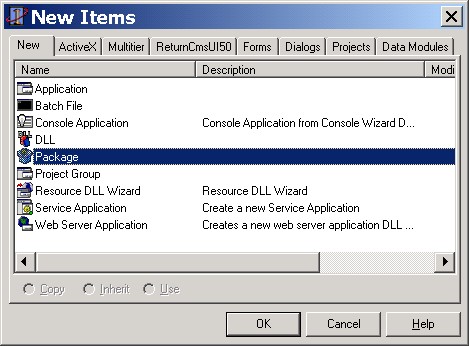
然后回到包PackageTemp.bpl的说明界面，单击右键选择Remove From Project将新建的文件从PackageTemp.bpl里面删除。（如下图）。



2、以上工作完成后，重新选择Project Manager 单击‘New’按钮（见下图）



出现Delphi的“Object Repository”并选择‘New’页的package项目新建立一个包（见下图）。



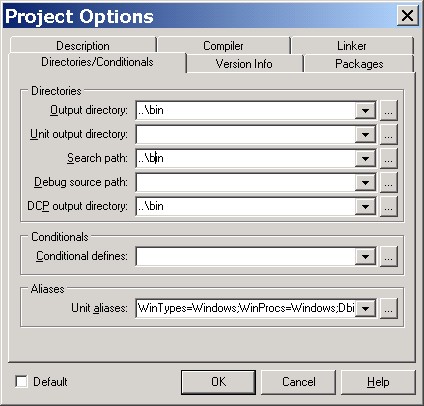
将新建立的包保存到工作目录的报表子目录下，并按照命名规范用ReportUI50.dpk为文件名保存。

下面可以回到Project Manager，单击右键并选择Option选项。设定包的编译选项。（见下图）



按下Option后会出现Project Option对话框

选择Project Option对话框的Directories/Condicionals页，将OutPut Directory，Search Path和DCP OutPut Directory都设置为‘..\bin’（见下图）。

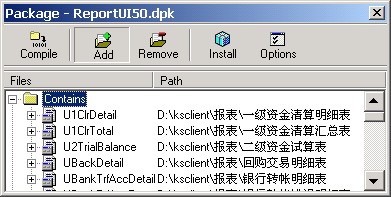


设置成功后按下‘OK’键并保存。

如果该包已经存在，这一步的工作就可以省略。

3、在确保包和程序文件都存在的前提下，需要将程序文件包含到包里。

重新选择Project Manager，双击显示框中的“ReportUI50.bpl”，将当前活动的工程设置为“ReportUI50.bpl”，然后在显示的包的说明界面中选择‘Add’。



然后在弹出的Add对话框中单击‘Browse’选择需要添加的程序文件

通过对话框查找刚才保存在工作目录的报表子目录的经营情况综合报表子目录下的UrunningStatus.pas文件，并按下‘打开’键。

选定了要包含的文件后，单击‘Add’对话框的‘OK’键将文件加入包（见下图）



这样我们就成功的将程序包含到包中了。现在可以编译一下这个包，确保可以编译通过了。

PS：说明：为什么要安装上面的步骤进行操作？

因为这些公共的父类模版都在“KSFramework50.BPL”程序包里面。我们需要的子类模块在我们特定的程序包里面。虽然Object Pascal本身并不禁止跨程序包继承，但是在IDE里面有这个限制。为了解决这个问题，我们预先建立了一个程序包“PackageTemp.bpl”，包含所有可用的公共的父类模版。然后在该模块里面通过继承的形式创建新的子类模块。创建好以后再把该模块移到真正需要它的程序包。

### 使得新建的程序出现在主程序中

完成了上面的工作以后，我们就可以开始设法使新的界面程序出现在主程序中了。

1、首先在包ReportUI50.bpl中找到我们新作的界面程序，双击打开，并按下键盘的‘F12’快捷键调出代码编辑器

在{$R \*.DFM}和end之间增加如下的代码。

initialization

TUIStdClassFactory.Create('经营情况综合报表',TfaRunningStatus);

对以上的代码的解释：

TUIStdClassFactory是在框架程序中定义的类，实际上起了类工厂的作用，负责将界面程序注册到程序环境中，该类构造函数Create需要两个参数，'经营情况综合报表'是界面的名字，也是标示符，TfaRunningStatus就是在本程序中定义的界面的类。

这段代码放在程序的initialization（初始化）部分，当这个程序被调入时会自动运行。

2、接下来需要修改配置文件KSClient.ini，使得该程序包被主程序加载。

在工作目录的Configs目录中找到配置文件KSClient.ini，可以用普通的记事本打开，找到Packages部分，确保包含了包ReportUI50.BPL条目，并且编号小于Count中所限定的数量，比比如向下面显示的那样。

[Packages]

Count=32

31=ReportUI50.BPL

修改成功后不要忘了使用配置文件的转换程序SysFileMan50c.exe（在bin目录下）将配置文件转换为加密文件KSClient.in。

3、接下来需要修改菜单文件Menus.mns，使得该程序条目出现在菜单中。

在工作目录的Configs目录中找到菜单文件Menus.mns，可以用普通的记事本打开，找到Menus部分，确保包含了如下信息

;\*\*\*\*\*\*\*\*\*\*\*\*\*

ID=经营情况综合报表

.no=0

.type=

.caption=经营情况综合报表

.keyword=

.see=

.func=0

.des=

;\*\*\*\*\*\*\*\*\*\*\*\*\*

在调试阶段可以将.no和.func都设置为0，ID和.caption都设置为菜单的名字。

然后在营业部菜单部分，确保包含了如下信息

-+证监会报表

--经营情况综合报表

表示‘经营情况综合报表’包含在‘证监会报表’这个组下。

修改成功后不要忘了使用菜单文件的转换程序SysFileMan50c.exe（在bin目录下）将菜单文件转换为加密文件menus.mn。

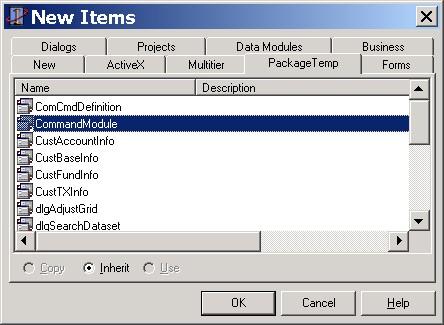
4、好了，完成了以上的工作后，可以启动主程序，在主程序的菜单部分找到‘经营情况综合报表’条目，单击就可以打开我们新做的界面了。

## 设计数据模块

下面我们就要开始设计数据模块，定义输入输出参数，这个部分是界面程序同后台程序通讯的桥梁。

1、将当前活动的工程设置为“PackageTemp.bpl”

2、选择Deiphi的主菜单的“File”目录下的“New”。按下“New”出现Delphi的“Object Repository”。选择“PackageTemp”页的“CommandModule”项目，并单击‘OK’键。（见下图）



然后就会出现从数据模块模板继承下来的界面（如下图）。



在Delphi的Object Inspector中将界面的Name改成dmRunningStatus，然后需要将其保存到工作目录的报表子目录的经营情况综合报表子目录下，并且按照命名规范定义文件名为UCMRunningStatus。

然后就可以像对界面程序一样，将该数据模块从“PackageTemp.bpl”包中删除并转移到“ReportUI50.bpl”包中去，具体方法同上面讲过的一样。

3、完成了数据模块的迁移工作后，现在要开始放置组件和编写代码了。

打开该界面后，在代码编译器中添加如下代码。

在代码的“{$R \*.DFM}”和“end.”之间增加如下代码。

Initialization

dmQueryBankCheckFlag := TdmQueryBankCheckFlag.Create(Application);

对以上的代码的解释：

dmQueryBankCheckFlag := TdmQueryBankCheckFlag.Create(Application)用于创建数据模块对象，数据模块模板在基类中实现了在对象被创建时，将其包含的所有的数据定义对象和数据实现对象注册到主程序的环境的操作。

这段代码放在程序的initialization（初始化）部分，当这个程序被调入时会自动运行。

4、下面开始放置组件和设置属性

首先在数据模块上面增加一个命令定义对象(TWVCommandTypeInfo)，并命名为“ctRuningStatus1”（TWVCommandTypeInfo组件在组件面板的WorkView页上）。并且在Delphi的Object Inspector中将其ID设置“经营情况综合报表”。

接着要设置命令参数，即把交易编码里的输入和输出参数加到命令定义对象中。

我们首先来看输入参数。

输入参数：

开户营业部 sbranch\_code1[4]

日期（月份） sdate0[9]

货币类型 scurrency\_type[3] 传’01’

职工所属营业部代码 sbranch\_code0[4]

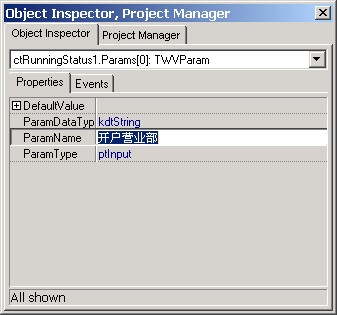
职工代码 semp[7]

网卡号 sstation0[17]

按照约定，‘职工所属营业部代码’，‘职工代码’和‘网卡号’三个参数由系统环境自动提供，所以不需要增加到数据定义对象中，所以在输入方面只需要增加‘开户营业部’，‘日期’和‘货币类型’三个参数即可。

具体操作步骤：双击命令定义对象ctRuningStatus1，在弹出的界面里点击界面上方工具栏中的第一个图标(Add New)，即增加了一个参数。在该参数的可视属性编辑器(Object Inspector)里，分别给各属性赋相应值。例如要添加“开户营业部”，ParamName表示参数名字，这里自然应该设置为“开户营业部”，ParamType表示参数类型，用于区分参数是输入还是输出或者既是输入又是输出，这里自然应该设置为“ptInput”表示是输入。ParamDataType表示参数的数据类型，分为字符串，整数，实数，对象类型等多种，由于“开户营业部”需要使用的实际的参数是sbranch\_code1，这个是同后台程序约定好的一个字符串，所以将其设置为“kdtString”，表示这个是一个字符串类型。

设置的结果见下图：



然后可以用相同的方法设置另两个参数。

下面我们来看输出参数。

由于我们的目的是为了获得整个的查询结果数据集，所以对于所有的输出参数，都不需要在命令定义对象中增加，取而代之的是一个称为“查询结果集”输出参数，该“查询结果集”代替了所有输出参数。

另外我们还需要增加另外两个约定好的输出参数‘返回码’和‘返回信息’。

返回码是一个整数，每个后台处理程序都需要返回该值，表示处理结果是否正确，返回码如果为0，表示正确，如果不为0，用各种不同的整数表示不同的错误。

返回信息是一个字符串，当返回码不为0时，用来输出错误信息。

然后使用同上面相同的方法来设置这三个参数，需要注意的是

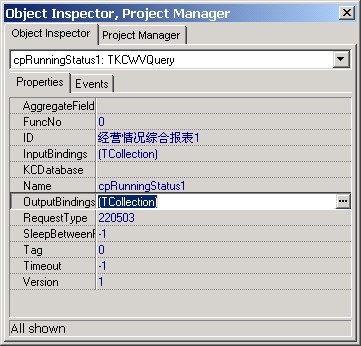
1. 这三个参数的参数类型ParamType都应该设置为“ptOutput”表示是输入。
2. 查询结果集的参数数据类型ParamDataType应该设置为kdtObject，表示是对象型。
3. 返回码的参数数据类型ParamDataType应该设置为kdtInteger表示整数型。

设置完成见下图：



然后需要在数据模块上面增加一个查询型的命令处理对象TWVQuery（在组件面板的KC页上），并命名为“cpRunningStatus1”。将命令处理对象的ID设置为同相应的命令定义对象“ctRunningStatus1”的ID相同。只有这样，命令处理对象才和命令定义对象关联起来了（通过ID关联）。

然后设置RequestType(请求功能号)，即交易编码的代码。在这里，RequestType设为“220503”（如下图）。



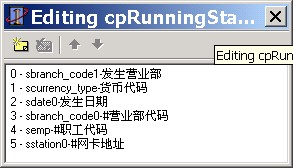
接着，和设置命令定义对象一样，要设置命令参数，即把交易编码里的输入和输出参数加到命令定义对象中。

和命令定义对象设置不同的是：命令处理对象添加输入和输出参数时，需分两步，要把加入输入参数和加入输出参数。

具体方法是：在Object Inspector中双击命令处理对象cpRunningStatus1的InputBindings属性。在弹出的界面里点击界面上方工具栏中的第一个图标(Add New)，即增加了一个参数。在该参数的可视属性编辑器(Object Inspector)里，分别给各属性赋相应值（实际上是将命令定义对象和Cpack中的英文字段绑定起来）。FieldName属性表示Cpack中定义的参数名，ParamName表示在命令定义对象中定义的字段名或者由系统环境定义的字段名（如下图）

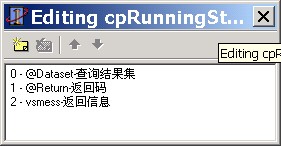


需要将6个输入参数都添加进去。结果见下图。



环境参数“#营业部代码”，“#职工代码”和“#网卡地址”，分别表示没有在命令定义对象中定义的三个参数。

然后在Object Inspector中双击命令处理对象cpRunningStatus1的OutputBindings属性。用同样的方法添加输出参数。（见下图）



“查询结果集”对应@DataSet。

“返回码”对应@Return。

“返回信息”对应vsmess。

这三个特殊的参数都是约定好的，请不要更改。

就这样，我们就完成了经营情况综合报表1的数据模块。

然后可以用同样的方法完成经营情况综合报表2的数据模块。

## 设计界面

完成了数据模块的设计工作后，可以开始实现界面功能了。

1、给工作视图（WorkView）增加字段。

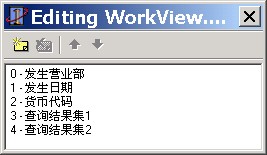
在工作视图中，增加如下字段。

开户营业部，日期，货币类型，查询结果集1和查询结果集2。

开户营业部，日期和货币类型是输入参数，查询结果集1和查询结果集2分别表示业务编码‘经营情况综合报表1’和‘经营情况综合报表2’的输出结果。

开户营业部，日期和货币类型都是字符串类型，查询结果集1和查询结果集2都是对象类型。

具体方法是：双击工作视图WorkView，在弹出的界面里点击界面上方工具栏中的第一个图标(Add New)，即增加了一个工作字段。在该工作字段的可视属性编辑器(Object Inspector)里，分别给各主要属性赋相应值。例如要添加“查询结果集1”，Caption和Name应设成“查询结果集”,DataType设为对象类型“KdtObject”，OwnObject设为True（注意：对于查询结果集这种对象类型的工作字段，一定要设置该属性）。添加“开户营业部”时，Name应设为“开户营业部”，DataType应设成“KdtString”。设置完成的情况如下图。

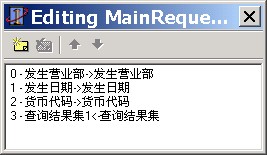


2、设置命令请求对象。

在界面上已经有一个命令请求对象MainRequest了，由于在这个界面上我们需要两个查询的结果，所以还需要增加另外一个命令请求对象，可以命名为MainRequest2。

对于命令请求对象MainRequest首先需要设置其ID，在这里，需与命令包里数据模板中命令定义对象的ID一致，为“经营情况综合报表1”。然后自动绑定一些参数。具体方法是：选中命令请求对象MainRequest，点击右键，选择“Add All Existed Request Bindings”，这样命令请求对象和命令定义对象就关联起来了。

最后手工调整。有可能你在工作视图WorkView中和在命令定义对象中定义的字段名不太相同（系统无法自动绑定）或者你不需要将工作视图里所有字段都与命令定义对象绑定，这时需要手工调整。在我们这个例子里，参数‘查询结果集1’就和命令定义对象定义的参数‘查询结果集’不同，所以需要手工修改。设置完成的结果见下图。



然后可以用基本相同的方法设置命令请求对象MainRequest2

3、设置输出绑定，将查询的结果放入数据源对象。

现在事实上我们已经完成了所有的向后台请求数据结果的工作。返回的结果已经可以储存在字段‘查询结果集1’和‘查询结果集2’中了。

由于我们需要将返回的结果以数据源的形式传送给报表格式文件，所以现在我们需要设置输出绑定。

首先需要在界面上放置一个DataSource对象，将其命名为DataSource1，然后需要放置一个WVDBBinder对象（在组件面板的WorkView页上）, 将其命名为WVDBBinder1。

将WVDBBinder1的DataSource属性设置为DataSource1，将FieldName属性设置为‘查询结果集1’，这样就将‘查询结果集1’字段和DataSource1绑定了。（见下图）

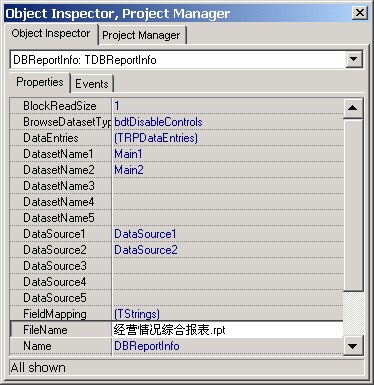


然后可以在界面上放置一个DataSource对象，将其命名为DataSource2，然后再放置一个WVDBBinder对象, 将其命名为WVDBBinder2。

然后可以用同样的方法将字段‘查询结果集2’和DataSource2绑定。

4、将数据源绑定到报表控件。

首先设置报表控件的FileName属性为‘经营情况综合报表.rpt’，表示打印所需要使用的报表的名字。然后需要将DataSource1和DataSource2加入报表控件并命名为Main1和Main2（注意：这也是在报表文件的数据源中使用的名字）。完成后见下图。



5、在界面上放置输入控件。

在界面里“查询条件”页选项卡上放置各种控件，并将控件和工作视图WorkView的工作字段关联起来。

以添加“发生营业部”为例。

首先，发生营业部应该是让用户多选一，因此需用下拉框(TWVCombox)。

选取一个label和一个下拉框控件。Label的caption设为“发生营业部”。下拉框控件（一定要是TWVCombox）的主要属性应该这样设置：WorkView为“WorkView”，FieldName设为对应工作视图的“发生营业部”。这样控件和工作视图关联起来了。因为“发生营业部”需从数据字典里取数。从数据字典里取数一定要设置三个属性，DataPresentType、ItemsDataEntry,Style。具体值为：DataPresentType恒定设为“IDOBJECTS”，ItemsDataEntry设为“银行”对应的数据字典条目“-8”，Style恒定设为“csDropDownList”。

如此，程序已经可以基本完成功能了。

## 收尾工作

1、编译、运行。

2、编写代码。

适当编写一些代码。在这里，主要是对“发生营业部”要进行特殊控制。因为，如果是以营业部操作员身份登录，那么不能让操作员选择“查询营业部”，如果是以总部操作员身份登录，则允许其选择营业部。我们需要重载（override）父类的Init方法。在重载的Init加如下代码：

cbBranchCode.Enabled := IsCenterDepartment;

InitOnlyDepartmentItems(cbBranchCode.Items,WorkView.FieldByName('查询营业部'),True);

WorkView.Reset;

其中，InitOnlyDepartMentItems是个公共函数，第一个参数是“查询营业部”对应的下拉框控件的名字（Name），第二个参数是“查询营业部”工作字段，第三个参数表示是否在下拉框中加“所有营业部”一项。

3、控制和优化。

对界面的一些控制和优化。

调整TabOrder顺序和调整各个控件的摆放位置。

对各工作字段进行有效性控制。设置各工作字段的Checker和Contrained。如果需要用到域(Domain)控制的工作字段则需要设置相应的域。

监控各工作字段的有效性，并控制“查询”按钮正显或者灰显。

采用的方式是：在工作视图WorkView中，新建一个监控有效性的Monitor。

具体方法是：选中工作视图WorkView，在工作视图WorkView的可视属性编辑器中，选择FieldsMonitors，点击其后面带三个圆点的小按钮。在弹出的界面里，点击上方工具栏中的第一个图标(Add New)，这样就新建了一个Monitor。接着需要设置该Monitor的各属性。在其可视属性编辑器中，设置其说明属性Description，设置为“有效性校验”（此属性并不是必须设置的，但设置了便于以后维护）。然后选择MonitorValidChangedFields，该属性设置的是需要被监控的工作字段。点击该属性后带三个圆点的小按钮，在弹出的界面的多选框(CheckListBox)中选取需要被监控的工作字段。一般选取全部字段。(在这个程序里，实际上我们不需监控，每个字段都会是有效的。) 然后在该Monitor的可视属性编辑器的事件(Event)页选项卡里为OnValidChanged事件编写代码。代码一般如下：

BtnQuery.Enabled:=Valid;

BtnQuery为“查询”按钮的名字(Name)。

# 十五、几种界面模版及其使用

## 1、输入模版

### 1.1、关于取消

界面上面“取消”按键的意义是将当前输入（还没有提交到后台）的数据清空，输入焦点恢复到第一个输入控件上面，让用户重新输入。一些输入界面有一个显示最近操作记录的表格框（ListView），那么在取消的时候不应该清空该表格。因为表格里面包含的是已经完成的操作，所以在取消的时候清空不适当。

在取消的时候调用的方法是“TfaInputTemp.Reset”。在该方法的实现里面清空了客户号控件、将工作视图（TWorkView）复位（TWorkView.Reset），最后将输入焦点放在第一个输入控件上面。根据前面的说明，在该方法里面不应该清空实现最近操作记录的表格。

### 1.2、关于确认

输入模版TfaInputTemp在确认的时候调用“TfaInputTemp.DoIt”。在“DoIt”里面会调用“SendCommand”。“SendCommand”方法在处理完毕以后，不论是否处理成功（产生意外），都会将将输入焦点放在第一个输入控件上面。

如果操作成功（没有意外产生），那么输入模版的子类在重载“DoIt”方法的时候，一定要调用“Reset”方法，将数据清空，输入焦点恢复到第一个控件上面。注意不是直接调用“TWorkView.Reset”，而是父类（输入模版）的“Reset”方法。

### 1.3、关于Enter键自动下移

在按下Enter键的时候，如果字段设置了GoNextWhenPressEnter，并且字段有效，那么输入焦点自动移动到下一个控件。如果想暂时禁止该特性，例如弹出一个提示价格超过涨跌停价格对话框，同时希望关闭对话框以后输入焦点暂时保留在原来的控件上面。处理的方法是设置字段的“AutoGoNext”属性。

如果设置了GoNextWhenPressEnter，那么每次按下Enter的时候，都会先设置AutoGoNext:=True，然后将控件的数据同步到工作字段。如果控件的数据发生改变，那么字段的数据发生改变，触发“OnCheckValid”和“OnValueChanged”事件。在这些事件里面可以修改字段的AutoGoNext属性。这些事件处理完毕以后。程序根据字段的有效性（Valid）和AutoGoNext属性控制是否自动切换到下一个控件。如果AutoGoNext:=False，保留焦点在该字段对应的控件上面。用户没有修改任何数据再次在该控件上面按下“Enter”，因为字段的值没有改变，那么不会触发字段的事件，AutoGoNext的值是自动设置的True。输入焦点会自动切换到下一个控件。

什么时候需要设置“AutoGoNext:=False”？如果在字段的“OnCheckValid”和“OnValueChanged”事件里面弹出一个警告性质的对话框，那么一般在紧接着后面的代码设置“AutoGoNext:=False”。这样的例子有：客户开户时候的证件号码（提示证件号码已经存在）、委托价格（提示超过涨跌停价格）等等。

## 2、参数设置模版

### 2.1、界面布局

参数设置界面分为两页。第一页包括：查询条件、显示查询结果的表格、以及查询、增加、修改、删除、细节几个功能按键。初始状态是在第一页。用户输入查询条件以后，点击查询按键，查询出数据。点击增加、修改、删除、细节几个功能按键，切换到第二页。用户完成数据录入工作以后，点击“确认”或者“取消”，完成处理以后，切换回第一页。

### 2.2、参数设置界面的状态和状态转换

参数设置界面包括以下几种状态：

// 参数设置界面的状态

**type**

TSettingStatus = (

ssBrowse, // 浏览状态

ssAdd, // 增加状态

ssChange, // 修改状态

ssDelete, // 删除状态

ssViewDetail // 浏览细节数据状态

);

状态通过属性“Status”表达。修改该属性，将调用“SetStatus”方法。

**property** Status : TSettingStatus **read** FStatus **write** SetStatus;

用户按下：“增加”、“修改”、“删除”、“细节”键以后，进行从浏览状态到增加、修改、删除、细节状态的切换。下面图示了状态转换的时候调用的过程。

BeforeAdd, StatusChanged

ssBrowse ------------------------------------------------------------>ssAdd

BeforeChange, StatusChanged

ssBrowse ------------------------------------------------------------>ssChange

BeforeDelete, StatusChanged

ssBrowse ------------------------------------------------------------>ssDelete

BeforeViewDetail, StatusChanged

ssBrowse ------------------------------------------------------------>ssViewDetail

用户在第二页按下“取消”，简单地将状态恢复到ssBrowse。

用户在第二页按下“确定”，如果ConfirmEdit返回True，那么根据当前状态分别调用： DoAdd、DoChange、DoDelete。如果状态成功的切换会浏览状态（操作成功，根据AutoRefresh属性自动重新查询数据。

在状态切换过程中调用到的各个方法的说明：

|  |  |  |
| --- | --- | --- |
| **方法** | **调用的其他方法** | **说明** |
| BeforeAdd | EnableEditCtrls(True);  WorkView.Reset(WVDataSource.GroupIndex); | 1、所有的输入项目可以编辑  2、初始化所有输入项目  (**由子类重载该方法，允许主键可以编辑**) |
| BeforeChange | EnableEditCtrls(True); | 所有的输入项目可以编辑(**由子类重载该方法，禁止主键可以编辑**) |
| BeforeDelete | EnableEditCtrls(False); | 所有的输入项目都不可以编辑 |
| BeforeViewDetail | EnableEditCtrls(False); | 所有的输入项目都不可以编辑 |
| StatusChanged | 如果是修改、删除、细节状态，调用DisplayData。  如果是增加状态并且要求以当前数据为模版，也调用DisplayData |  |
| DoAdd | SaveModifiedData;  SendEditCommand(AddRequest); | 同步修改的数据到工作视图，提交请求，如果成功(FEditSucceed=True而且没有意外抛出)，恢复状态为浏览 |
| DoChange | SaveModifiedData;  SendEditCommand(ChangeRequest); | 同步修改的数据到工作视图，提交请求，如果成功(FEditSucceed=True而且没有意外抛出)，恢复状态为浏览 |
| DoDelete | SaveModifiedData;  SendEditCommand(DeleteRequest); | 同步修改的数据到工作视图，提交请求(FEditSucceed=True而且没有意外抛出)，，如果成功恢复状态为浏览 |
| ConfirmEdit | 缺省情况下返回True。 | 返回是否允许修改。在按下确认的时候被调用。可以在这里弹出对话框提示确认或者复核。  **某些子类需要重载该函数，根据单机双柜员复核情况返回True/False。** |

其他相关方法说明

|  |  |
| --- | --- |
| **方法** | **说明** |
| EnableEditCtrls | 将第二页的输入控件全部有效或者无效。当增加和修改的时候这些控件一般有效，以后可以输入数据；在删除和浏览细节状态，都不可以输入。  注意：**一些子类需要重载该方法，控制特殊控件的有效或者无效。** |
| DisplayData | 依次调用WorkView.Reset(WVDataSource.GroupIndex)  WVDataSo urce.DataSourceToWorkView;  将数据从数据源同步到工作视图（再到各种控件）。  注意：**如果使用特殊的控件，子类需要重载该方法，控制特殊控件的数据同步。** |
| SendEditCommand | 发送请求到服务器 |
| SaveModifiedData | 调用WorkView.SynchronizeCtrlsToFields; 将修改的数据同步到工作视图。  注意：**如果使用特殊的控件，子类需要重载此方法，控制特殊控件的数据同步**。 |

### 2.3、权限控制

#### 概念

系统参数分为两种类型：

一种是只有全公司的统一数据，这些数据只能由总部统一维护（增加、修改、删除）。营业部只能浏览这些数据。在维护这种类型的参数时，界面上面没有营业部选择下拉框。

另外一种是总部设置缺省的参数，每个营业部还可以单独设置自己的参数，也可以不设置。如果不设置，营业部查询出来的数据是总部的缺省数据；如果设置了，查询出来的数据是营业部设置的数据。在维护这种类型的参数时，界面上面有营业部选择下拉框，列出所有的营业部。当发起营业部是总部的时候，允许选择目的营业部，但是当目的营业部不是总部的时候，不许修改数据，只能查询。当发起营业部不是总部的时候，不允许选择目的营业部，目的营业部只能是发起营业部。如果查询出来的数据是营业部自己设置的数据，允许增加、修改、删除；如果查询出来的数据是总部设置的缺省参数，不允许修改和删除、只允许增加。

**表格1：权限汇总**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **发起营业部** | **目的营业部** | **全公司的统一数据** | **营业部还可以单独设置自己的参数** | |
| **总部设置的缺省参数** | **营业部设置的数据** |
| **总部** | **总部** | 查询、增加、修改、删除 | 查询、增加、修改、删除 | 查询、增加、修改、删除 |
| **总部** | **营业部** | <无> | 查询 | 查询 |
| **营业部** | **营业部** | 查询 | 查询、增加 | 查询、增加、修改、删除 |

#### 基本实现

在USetParamsTemp.pas里面TfaSetParamsTemp（参数设置界面模版）的完成基本的实现。

**表格2：控制按键的有效性**

|  |  |
| --- | --- |
| **按键** | **控制Enabled的参数逻辑** |
| btnAdd | CanEdit and CanAdd |
| btnChange | CanEdit and CanChange |
| btnDelete | CanEdit and CanDelete |

**表格3：控制参数的设置**

|  |  |  |
| --- | --- | --- |
|  | **全公司的统一数据** | **营业部还可以单独设置自己的参数** |
| CanEdit | 发起营业部＝总部 | 发起营业部=目的营业部 |
| **CanAdd** | True | True |
| **CanChange** | True | 查询出来的数据是本营业部的数据 |
| **CanDelete** | True | 查询出来的数据是本营业部的数据 |

**表格4：TfaSetParamsTemp相关的字段和方法**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **说明** | **重载** |
| FDepartmentFieldName | 字段（string） | 查询出的Dataset里面代表营业部的字段名称，缺省为“sbranch\_code0”。如果为“”，表示不判断结果集里面的营业部。用于校验查询出来的数据是总部的数据还是营业部自己的数据。 |  |
| FDepartmentWVField | 字段（TWVField） | WorkView里面代表目的营业部的字段，用于检验发起营业部和目的营业部是否相同。  缺省为nil。 |  |
| FMaintainType | 字段（TParamMaintainType） | 参数维护类型：  pmtCenter：总部统一维护  pmtDepartment：营业部维护  缺省为pmtCenter。 |  |
| Init | 方法 | 初始化部分，子类重载该方法，设置FDepartmentFieldName、FDepartmentWVField、FMaintainType几个字段。 | 可以 |
| InternalGetCanEdit | 方法 | 判断CanEdit的状态。一般不需要重载该方法。少数特殊的界面，例如数据字典维护，在选择所有条目的时候不允许修改，需要重载该方法。 | 可以 |
| UpdateButton | 方法 | 更新按键的Enable状态。当一些数据（例如目的营业部）发生改变的时候，调用该方法。 |  |

其他从该模版继承的参数设置界面应该重载父类的一些方法，完成对权限的控制。

**表格5：子类需要重载的内容**

|  |  |  |
| --- | --- | --- |
| **数据维护方式** | **子类需要设置的内容** | **说明** |
| **全公司的统一数据** | FMaintainType:= pmtCenter; | 不需要重载Init方法，因为缺省设置为pmtCenter |
| **营业部可以单独设置参数** | FMaintainType:= pmtDepartment;  FDepartmentFieldName := 'sbranch\_code0';  FDepartmentWVField := WorkView.FieldByName('设置营业部'); | 需要重载Init方法。注意'sbranch\_code0'和'设置营业部'应该是该界面上面的实际值。 |

**补充**：一般需要删除子类原来重载的InternalGetCanEdit方法。

**表格6：各种常见的配置**

|  |  |
| --- | --- |
| **配置** | **说明** |
| FMaintainType:= pmtCenter;不可以选择营业部 | 只有总部的一套参数数据需要维护。 |
| FMaintainType:= pmtCenter;可以选择营业部 | 每个营业部有一套独立的数据需要维护，但是维护的权限只有总部才有。 |
| FMaintainType:= pmtDepartment;  FDepartmentFieldName := 'sbranch\_code0';  FDepartmentWVField := WorkView.FieldByName('设置营业部'); | 每个营业部维护自己独立的一套数据，不允许跨营业部操作。  查询出的数据包含一个“营业部”的字段，查询出来的数据可能是总部的，也可能是营业部，依照“营业部”的字段确定。 |
| FMaintainType:= pmtDepartment;  FDepartmentFieldName := '';  FDepartmentWVField := WorkView.FieldByName('设置营业部'); | 每个营业部维护自己独立的一套数据，不允许跨营业部操作。  查询出的数据不包含“营业部”字段，必然都是该营业部的数据。 |

#### 细节

关键的方法

|  |  |  |
| --- | --- | --- |
| **方法** | **功能** | **何时被调用** |
| UpdateButton | FCanEdit := InternalGetCanEdit;  然后根据FCanEdit、FCanAdd、FCanDelete、FCanChange更新按键的有效性，参看“**表格2：控制按键的有效性**” | 被CheckDataCanEdit调用 |
| CheckDataCanEdit | 1、无查询结果数据，禁止修改、删除，允许增加；  2、有查询数据，如果是营业部维护数据，那么需要根据返回数据检查营业部，只能修改和删除本营业部的数据; 否则不需要检查营业部，允许编辑  3、调用UpdateButton更新显示  注意：该方法只根据当前数据集的数据和维护类型控制FCanAdd、FCanDelete、FCanChange几个属性，最终按键的有效性还要根据FCanEdit控制。参见“**表格2：控制按键的有效性**” 、“**表格3：控制参数的设置**”和InternalGetCanEdit方法。 | 初始化的时候（Init）；数据集发生改变、数据集光标移动到新的一行 |
| InternalGetCanEdit | 根据数据维护类型和当前营业部控制FCanEdit属性，参见“**表格3：控制参数的设置**”。  注意：少数子类可以重载该方法，提供特殊的控制。 | UpdateButton |

## 3、报表模版

### 3.1、查询数据

按下“查询”按键，如下处理：

1、将数据从控件同步到工作视图。

2、调用DoIt

相关方法说明：

|  |  |
| --- | --- |
| **相关方法** | **说明** |
| DoIt | 1、关闭以前的结果集。调用“**CloseDatasets**”。  2、如果**SendCommandWhenDoIt**=true调用“**InternalSendCommand**”发送命令，  3、如果没有意外发生，切换到打印页。打印按键有效。 |
| **SendCommand** | 显示等待消息框，然后调用**InternalSendCommand。**保证关闭等待消息框。 |
| **InternalSendCommand** | 调用MainRequest.SendCommand。  注意1：部分子类需要重载该方法，增加其他的命令调用。  注意2：部分子类需要重载该方法，一个命令都不调用。 |
| **CloseDatasets** | 释放当前打开的数据集，将“打印”按键设置为无效。实现的机制是将工作视图里面包含TDataset的工作字段全部清空。  当界面上面的输入条件发生变化的时候，应该调用“**CloseDatasets**”方法，释放以前的数据集，并且将“打印”按键设置为无效。具体方法是在工作视图里面增加一个监控器（monitor），监控所有输入字段的值的变化。然后在监控器的OnValueChanged里面调用“**CloseDatasets**”。 |

例子见“汇总对帐单”。

### 3.2、打印

按下“打印”按键，调用“**DoPrintReport**”。相关方法如下：

|  |  |
| --- | --- |
| **相关方法** | **说明** |
| DoPrintReport | 1、调用**BeforePrint**  2、调用DBReportInfo.PrepareReport  3、调用AddSystemParams，将系统参数（系统日期、操作员等等）加入到打印变量里面。  4、调用**AfterPreparePrint**  5、完成打印或者预览。  6、调用**AfterPrint** |
| BeforePrint | 在打印以前被调用，子类可以重载该方法，完成其他必要的操作。 |
| **AfterPrint** | 在打印以后被调用，子类可以重载该方法，完成其他必要的操作。 |
| **AfterPreparePrint** | 在打印以前被调用。  注意：子类可以重载该方法，设置其他的打印变量。 |
|  |  |

例子见“汇总对帐单”。

### 3.3、批量打印的处理

处理批量打印的方法一般是重载“**DoPrintReport**”和“**InternalSendCommand**”方法。

在“**InternalSendCommand**”确定需要循环打印的范围。根据业务需要，可能是调用一个交易编码获得，那么应该将该交易编码设置到MainRequest上面，不需要重载“**InternalSendCommand**”方法。如果是前台根据输入数据范围自动确定，那么在“**InternalSendCommand**”里面禁止MainRequest，方法是重载该方法，然后不调用“**inherited**”。

在“**DoPrintReport**”里面需要如下处理：

//开始while 或者for循环

try

// 获得需要打印的数据

inherited; // 调用缺省的方法。

except

// 显示错误消息（使用Context.GetDialogSystem.ShowError2对话框），询问是否继续，如果不继续，那么退出

end;

//结束循环

例子参见“批量打印交割单”。

另外参见“**长时间操作的进度显示**”文档，描述了更好的处理循环打印的方法。

### 3.4、更复杂的关于批量打印的处理

1、因为批量打印一般需要反复查询数据，所以一般不需要在按下“查询”按键的时候进行查询。解决的方法是重载Init方法，设置**SendCommandWhenDoIt**属性是False。

2、在批量打印的时候重载DoPrintReport方法。然后在重载的该方法里面根据内容循环调用一个单独（根据当前循环的项目）的打印过程（该过程要每个程序员自己书写，不是模版上面的虚方法），例如名字叫“DoPrintOneReport”。注意格式应该使用长时间处理的形式，举例如下：

procedure MyFrame.DoPrintReport;

var

Canceled : Boolean;

begin

Canceled := False;

ShowProgress('操作名称', True, SCancelPrompt);

try

while do // 设置循环条件

begin

UpdateProgress('当前操作');

if IsProgressCanceled then

begin

// 如果在进度窗口取消，那么退出循环

Canceled := True;

Break;

end;

try

DoPrintOneReport**(一些参数，用于获取查询数据)**;

except

// 如果在完成一个操作的时候产生一个意外，询问用户是否继续处理下一个

// 如果不处理，退出循环

on E : Exception do

if IsProgressCanceled or

(Context.GetDialogSystem.ShowError2(

Format(SConfirmIgnoreError, [E.Message])

)<> uaYes) then

begin

Canceled := True;

Break;

end;

end;

end;

finally

CloseProgress; // 保证关闭进度窗口

end;

if Canceled then

Context.GetDialogSystem.ShowMessage('xxx操作被用户终止');

end;

3、在DoPrintOneReport界面里面先查询数据，然后调用“**DefaultPrint**”完成实际的打印。

procedure MyFrame.DoPrintOneReport**(一些参数，用于获取查询数据)**;

begin

SendCommand; // 获得数据

**DefaultPrint; // 完成实际的打印。**

end;

## 4、查询模版

# 第四部分 具体编码部分

# 十六、其他编码问题

## 1、关于数据字典

### 内部表示

参看“KSDataDictionary.pas”里面的TIdObject和TIdObjectList。

1）TIdObjectList.GetText 处理TField.GetText事件。

2）利用TIdObjectList.Find查找指定的子项目。

NeedIdObjectList : 根据数据字典条目，返回TIdObjectList对象。

### 数据来源

1）初始化读取数据字典（部分条目）

UCommData.pas TdmCommData.start

2）其他条目：在需要的时候读取。

NeedIdObjectList和UCommData.pas 的GetIdObjects、dmCommData.GetDataDictionary。

### 根据数据字典的内容自动初始化TStrings列表

1）调用KSFrameWorkUtils.pas的InitIdObjectItems或者InitDepartmentItems函数。

TStrings每个条目(字符串)都包含一个TIdObject对象。

参看：TStrings.AddObject、TStrings.Objects[]

2）对于TWVComboBox和TWVCheckListBox设置ItemDataEntry属性。属性值是数字，表示数据字典条目(例如“1011”表示“校验方式”)。如果数字前面有“\*”表示增加“-”，否则不增加。

原理：TWVStringsMan（WorkViews.pas）对象负责初始化TStrings列表。注意TWVStringsMan .GetStrings。

具体实例：KSComCmdDefs.pas的TComCmdDefinition的GetDataDictionaryItems。

参看：function WVGetStrings

### 根据数据字典的内容，处理各种控件

标签、下拉框、列表框、表格(Grid)：显示、输入（控件到字段）、同步（字段到控件）。

1、WVCtrls.pas里面的控件一般包含DataPresentType和DataPresentParam属性。

DataPresentType定义了一种数据的表达形式。DataPresentParam辅助参数。

实现：

WorkViews.pas里面的TWVFieldPresent对象。负责在控件和字段之间双向同步数据，即数据怎么在控件里面表达出来。

CtrlToField：控件的数据同步到工作字段。

FieldToCtrl：工作字段的数据同步到控件。

SetFieldEventHanlder：设置TField.OnGetText事件的处理函数。

参看：

function WVCtrlToField

function WVFieldToCtrl

function WVSetFieldEventHanlder

2、具体实例：

KSComCmdDefs.pas的TComCmdDefinition的CommonDataPresent。定义了“DataPresentType”=“IDOBJECTS”。

a）TComCmdDefinition.CommonDataPresentCtrlToField

针对TWVComboBox，并且Style=csDropDownList

b）TComCmdDefinition.CommonDataPresentFieldToCtrl

针对TWVComboBox，并且Style=csDropDownList

针对TWVLabel，并且DataPresentParam＝数据字典条目。

c）TComCmdDefinition.CommonDataPresentSetFieldHandler

利用了TIdObjectList.GetText。Param是数据字典条目。

3、注意：TWVDBBinder.Bindings里面如下设置，才能处理DBGrid里面的显示。

FieldName：来源于交易编码的输出参数，和DBGrid里面的FieldName相同，都是TDataset的字段名。

DataPresentType=“IDOBJECTS”

DataPresentParam＝数据字典条目。

#### 使用TWVCheckBox

属性如下：

WorkView-工作视图

FieldName-工作字段名称

DataPresentType-数据表现形式

DataPresentParam－数据表现形式的参数

CheckedValue-选中的时候的值

UnCheckedValue-未选中的时候的值

DefaultChecked-缺省是否选中

PresentType-判断是否选中的方式。

cdpEqual：是否和CheckedValue相等。

cdpContain：是否包含CheckedValue。

cdpIndexedChar：指定位置的子字符串是否等于CheckedValue。

SeperateChar-用于cdpContain方式的分割字符，例如“,”。等于#0时表示没有分割的字符。

CharIndex-用于cdpIndexedChar方式指定位置。实际对应字符串的下标为 (CharIndex-1)\*Length(CheckedValue)+1。

#### 使用TWVCheckListBox

属性如下：

WorkView-工作视图

FieldName-工作字段名称

DataPresentType-数据表现形式

DataPresentParam－数据表现形式的参数

CheckedValue-选中的时候的值

UnCheckedValue-未选中的时候的值

DefaultChecked-缺省是否选中

PresentType-判断是否选中的方式。

cdpContain：是否包含Items{I]文字的位于SeperatedStr前面的部分。

cdpIndexedChar：指定位置(Items的下标)的子字符串是否等于CheckedValue。

SeperateChar-用于cdpContain方式的分割字符，例如“,”。等于#0时表示没有分割的字符。

ItemsDataEntry-用于初始化Items

SeperatedStr-用于cdpContain方式，分割Items里面每个文本的字符串。位于SeperatedStr前面的部分被作为每个项目的比较基准。一般用“-”

SynchronizeWhenExit-焦点失去的时候同步。

## 2、关于客户号输入控件的问题

### A、基本说明

#### 1、允许输入的账号类别

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 标志 | 类别 | 类型 | 补齐 | 是否需要重输 |  |
| 1 | 沪A股东代码 | 股东代码 | A000000000 | Y |  |
| 2 | 深A股东代码 | 股东代码 | 0000000000 | Y |  |
| 3 | 沪B股东代码 | 股东代码 | C000000000 | Y |  |
| 4 | 深B股东代码 | 股东代码 | 0000000000 | Y |  |
| 0 | 客户号 | 客户号 |  | Y |  |
| \* | 牛卡账号 | 客户号 |  | N |  |

#### 2、账号输入控件的基本要求

1）使用F2切换输入账号类别。

2）可以限制可以输入的账号类别。

3）是否需要重输账号，是可以配置的。（对上表中，“Y”的部分，“N”的部分固定不重输，因为刷卡）。

4）是否要求用户输入密码，是可以配置的。

5）对于在不同营业部有相同账号的情况，要求操作员选择营业部。

#### 3、账号输入控件的基本流程

0）初始状态，账号是**基本无效**和**未验证**的。

1）输入焦点进入账号输入控件：

保存当前的账号类别和输入内容

2）输入焦点在账号输入控件里面：

用户按F2切换输入账号类别，可以修改输入内容。

按Enter输入焦点离开该控件。

3）输入焦点离开账号输入控件：

a）如果账号值非空，根据补齐内容，自动补齐。

b）如果账号类别和输入内容与保存的以前的账号类别和输入内容有不同，认为是一个**新输入**的账号值。账号是**未验证**的。

c）如果账号值是**新输入**的，如果账号不是空串，认为是一个**基本有效**的账号值，否则认为是一个**基本无效**的账号值。

d）如果账号值是**未验证**的，并且**基本有效**（非空），执行下面的操作

i）如果要求重输，出现对话框，如果输入值不相同，或者操作员取消，退出处理，保持**未验证**状态。否则继续下一步。

ii）根据账号类别和账号值，向服务器请求客户信息。(多行记录，每行一个营业部、客户号、客户其他信息)。如果账号无效（无数据，0行），退出处理，保持**未验证**状态。

iii）如果返回多个营业部（超过一行），要求操作员选择营业部，如果操作员取消，退出处理，保持**未验证**状态。否则继续下一步。

iv）根据选中的营业部，保存客户号和客户信息。

v）如果要求客户输入密码，出现对话框，要求客户输入密码。如果客户或者操作员取消，退出处理，保持**未验证**状态。

vi）如果要求客户输入密码，向服务器请求验证客户。如果验证失败，回到v）步。

注意：这个时候客户已经有效，下面读取其他信息，然后同步到WorkView。

vii）如果需要取各个市场的主股东代码，那么根据客户号，取各个市场的主股东代码，保存。返回结果集每行一个市场及其主股东代码。

viii）如果需要取资金情况，那么根据客户号，取各个币种的资金情况，保存。返回结果集每行一个币种。

iX）显示客户信息（来自第ii）步）。设置为**已验证**状态。

#### 4、取得账号输入的值

在账号已经验证的情况下，可以取得输入账号的值。

如果要求取得某个市场的股东代码，而且市场和账号类别相同，那么返回输入的账号，否则返回对应市场的主股东代码。

### B、抽象的客户、股东账号、资金账号对象

参看：KSCustomers.pas

TKSAbstractCustomer<- TKSCustomer

TKSAbstractSecAccount<- TKSSecAccount

TKSAbstractMoneyAccount<- TKSMoneyAccount

都具有GetParamValue方法。

|  |  |
| --- | --- |
| 参数类别 | 来源 |
| 客户TKSAbstractCustomer | 验证客户 (100101)的输出参数名称 |
| 股东TKSAbstractSecAccount | 股东代码资料 (100306)的输出参数名称 |
| 资金TKSAbstractMoneyAccount | 取客户资金信息 (100313)的输出参数名称 |

### C、控件的基本属性

参看：KSEdits.pas

#### 1、设计时候的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 说明 | 缺省值 |
| AllowMethods | 允许的输入方法(0客户号，1沪A，2深A，3沪B，4深B，  #-银行代码，\*磁卡号)，例如“13”表示只能输入“沪AB”账号。“”表示所以账号。 |  |
| LabelCtrl | 连接到一个Label控件上面，显示文字。  注意Label的属性：  AutoSize=False，大小调好，Alignment=Right |  |
| SwitchKey | 切换键113表示F2 | 113 |
| FirstMethodChar | 缺省显示的输入方法 | 0(客户号) |
| GlobalFirstMethodChar | 缺省输入方法是否根据配置文件获得(False) |  |
| NeedPassword | 是否需要验证客户密码 | false |
| NeedReinput | 是否账号需要重新输入 | false |
| NeedMainAccounts | 是否需要自动获取各个市场的主账号  对和股东账号相关的界面，应该设置为true；与股东代码无关（只和客户相关），应该设置为false(优化程序)。 | true |
| NeedMoneyAccounts | 是否需要自动获取各个币种的资金账号的情况。对和资金相关的界面，应该设置为true；与资金无关（只和客户相关），应该设置为false(优化程序)。 |  |
| PasswordType | 客户密码类型（1-交易密码，2-取款密码，3-交易密码或者取款密码） | 1 |
| WorkView | 工作视图 |  |
| CustInfoGroupIndex | 工作视图里面显示客户信息的字段的分组 | 1 |
| AccountInfoGroupIndex | 工作视图里面显示股东信息的字段的分组 | 2 |
| MoneyInfoGroupIndex | 工作视图里面显示资金信息的字段的分组 | 3 |
| Market | 市场（用于控制工作视图里面显示的股东信息） |  |
| CurrencyType | 币种（用于控制工作视图里面显示的资金信息） |  |
| GoNextWhenPressEnter | 按Enter，下移 | True |
| GoPrevChar | 按特殊键，上移 | \* |
| OnValidChanged[事件] | 当输入账号的有效性发生改变的时候被调用 |  |
| Department | 当前营业部代码 |  |
| NeedPasswordOnCrossDepart | =True的时候表示，仅仅在客户不是本营业部的时候，要求客户输入密码 | False |
| AdvancedSelect | 高级选择客户（例如根据姓名、地址等等）。使用Alt+F2热键 | False |
| Required | 客户号是否可以为空（不输） | False |
| OnInputValidChanged | 当输入有效性(InputValid)发生改变的时候触发 |  |
| AllowCustomerStatus | 允许的客户状态。’’表示客户状态不是正常状态就警告。否则表示允许的状态串。 | ‘’ |
| NormalStatus | 客户正常状态 | ‘0’ |
| ProhibitCrossDepart | 是否禁止输入其他营业部的客户 | False |
| IsWarningCustomerStatus | 是否警告客户状态 | True |
| OnNeedPassword | 当需要密码的时候触发，确定是否需要输入密码 |  |
| OnException | 当意外时触发 |  |
| OnWarningCustomerStatus | 当警告客户状态时被触发 |  |
| ProhibitSelectDepart | 是否禁止选择不同营业部的客户。对自助交割一类的应用，应该设置为True。 | False |
| AllowCenter | 允许总部操作营业部的客户 |  |

#### 2、密码的控制

|  |  |  |
| --- | --- | --- |
|  | NeedPasswordOnCrossDepart  False | NeedPasswordOnCrossDepart  True |
| NeedPassword=False | 不需要密码 | 仅在跨营业部(不是总部)访问的时候需要密码。 |
| NeedPassword=True | 需要密码 | 需要密码 |

#### 3、控制操作跨营业部的客户

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **ProhibitCrossDepart** | **AllowCenter** | **NeedPasswordOnCrossDepart** |
| 允许跨营业部 | False |  | False |
| 跨营业部输密码 | False |  | True |
| 不允许跨营业部 | True | False |  |
| 允许总部 | True | True | 密码由NeedPassword确定 |

#### 4、运行时候的属性和方法：

|  |  |  |
| --- | --- | --- |
| **名字** | **类型** | **说明** |
| CurrentInputMethod | 属性 | 当前输入方法(TAccountInputMethod对象) |
| Valid | 属性 | 客户是否有效 |
| Customer | 属性 | 当前客户(TKSAbstractCustomer对象) |
| MainAccounts | 属性 | 各个市场的主账号列表（MainAccounts[I]）是TKSAbstractSecAccount对象。 |
| MoneyAccounts | 属性 | 各个币种的资金情况，MoneyAccounts[I]是TKSAbstractMoneyAccount对象 |
| FirstInputMethod | 方法 | 输入法切换（第一个输入法） |
| NextInputMethod | 方法 | 输入法切换（下一个输入法） |
| Reset | 方法 | 复位 |
| GetAccount | 方法 | 根据市场获取股东代码（当前输入的或者获取到的主股东代码），如果客户号无效或者对应市场无账号，返回nil。返回TKSAbstractSecAccount对象。 |
| InputValid | 属性 | 输入有效性。当Required=True的时候，等于Valid属性；当Required=False的时候，仅当客户号为空或者正确输入的时候返回True。 |
| CustomerPassword | 属性 | 客户密码 |
| SetCustomer | 方法 | 根据客户号指定一个客户。 |

注意：应该将该控件正确连接到工作视图对象和标签对象上面。

注意：应该正确设置工作视图的工作字段。工作字段的Name、DataField、GroupIndex应该正确指定。特别是和输入控件相关的显示信息的字段的GroupIndex。而且其他与显示客户信息无关（其他的输入/输出字段）的字段的GroupIndex应该不同。

注意：响应OnValidChanged事件，根据Valid属性，控制“确认”按键是否有效。

参看：UInputCustomer.pas。

### D、涉及到的交易编码

#### 1、验证客户 (100101)

|  |
| --- |
| 功能描述： 验证客户是否合法，是否存在这个客户 |
| 相关数据表：  股东代码表（HOLDER\_ACC）  客户基本资料表（CUST\_BASE\_INFO）  客户代理人资料（AGENT\_INFO） |
| 输入参数：  帐号类型 status0[2]（0客户号，1沪A，2深A，3沪B，4深B，  #-银行代码，\*磁卡号）  帐号 scard0[51]  柜员代码 semp  是否检查挂失状态 lvol2 1=检查，0=不检查  操作的营业部代码 sbranch\_code0  是否允许股东代码挂失 lvol3 1=检查，0=不检查  查询的货币代码 scurrency\_code  查询的市场代码 smarket\_code2 |
| 输出参数：  客户号 scust\_no  客户姓名 sname  代理姓名 sname2  代理证件类型 lsafe\_level  代理证件号 scert\_no2  代理权限 scust\_limit2  代理状态 sstatus3  代理到期日期 sdate0  可用市场集合 sphone[31]  客户类别 scust\_type  客户证件类型 lcert\_code  客户证件号 scert\_no  开户营业部 sbranch\_code0  开户营业部名称 scard0  委托权限 scust\_auth  客户号的状态 sstatus0  客户的业务控制串 scust\_auth2  如果查询的货币代码不为空，则返回：  可用金额 damt0  客户的利息 damt1  客户的协议冻结金额 damt2  如果查询的市场代码不为空，则返回：  指定市场的主股东代码 sholder\_ac\_no2    如果以股东代码验证，则返回如下参数：  验证的股东代码（规格化的） sholder\_ac\_no  验证的股东代码是否指定交易 sstatus1  验证的股东代码的状态 sstatus2 |

#### 2、客户密码 (100102)。

|  |
| --- |
| 功能描述： 客户密码验证 |
| 相关数据表：  客户基本资料（CUST\_BASE\_INFO） |
| 输入参数：  验证方式 lvol0 (1-交易密码，2-取款密码，3-交易密码或者取款密码)  客户号 scust\_no  客户密码 tx\_pwd  营业部代码 sbranch\_code  职工代码 semp  网卡号 sstation0 |
| 输出参数：  无 |
|  |
| 错误信息：  验证客户密码失败 05112  客户交易密码错误 05110  客户取款密码错误 05111 |

#### 3、股东代码资料 (100306)。

|  |
| --- |
| 功能描述： 取得股东代码资料 |
| 相关数据表：  HOLDER\_ACC |
| 输入参数：  客户号 scust\_no  市场代码 smarket\_code ‘’=查询所有市场，否则查询指定市场  主代码标志 lvol0 1=查询所有代码，2=只查询主代码  营业部代码 sbranch\_code0  职工代码 semp  网卡号 sstation0 |
| 输出参数：  客户号 scust\_no  市场代码 smarket\_code  股东代码 sholder\_ac\_no  报盘代码 sholder\_ac\_no2  主代码标志 smain\_flag  指定标志 smain\_flag2  指定席位 sstock\_code  报盘席位 sstock\_code2  代码姓名 sname |

#### 4、取客户资金信息 (100313)。

|  |  |
| --- | --- |
| 功能描述： 取得客户资金信息 | |
| 相关数据表：  客户资金（FUND） | |
| 输入参数：  开户营业部  发生营业部  客户号  资金帐号序号  货币类型  职工代码  网卡号 | Sbranch\_code1[4]  Sbranch\_code0[4]  Scust\_no[11]  Scurrency\_type2[3]  Scurrency\_type[3]  Semp[7]  Sstation0[17] |
| 输出参数：  开户营业部  客户号  权限标志(１显示，２不显示)  货币类型  资金余额  可用金额  可取金额  现金金额  支票金额  异常冻结金额  实际买入金额  买入冻结金额  卖出解冻金额  手工冻结金额  手工解冻金额  贷款金额  买入未过户金额  卖出未过户金额  信用额  信用额到期日  禁用资金  禁取资产  总市值  参考市值  质押金额  虚增保证金  现钞现汇标志  利息  罚息  贷款利息 | Sbranch\_code1[4]  Scust\_no[11]  Lsafe\_level  Scurrency\_type[3]  Damt4  Damt5  Damt6  Damt7  Damt8  Damt9  Damt12  Damt10  Damt11  Damt14  Damt15  Damt16  Damt18  Damt19  Damt23  Sdate1[9]  Damt24  Damt25  Damt29  Damt32  Damt0  Damt30  Sstatus０[2]  Damt26  Damt27  Damt28 |

### E、如何将数据同步到WorkView的字段

同步数据的一般流程：

1、根据GroupIndex，复位(Reset)工作字段。

注意：工作字段的GroupIndex，以免无关的字段被复位

2、根据该组的字段的DataField从抽象的账号对象里面读取数据（GetParamValue）

注意：对应工作字段的DataField应该设置正确。

|  |  |  |
| --- | --- | --- |
| 组别 |  |  |
| 客户信息 |  |  |
| 股东信息 |  |  |
| 资金信息 |  |  |

## 3、关于工作字段和工作视图

### 如何校验字段有效性

当字段的Constrained=True的时候，需要校验字段的有效性。

校验的顺序如下：

A）如果设置了字段的DomainName，而且对应的Domain存在：

1）根据Domain的Checker对象的属性，进行有效性校验；

2）然后调用Domain的OnCheckValid事件，完成补充的校验；

3）最后调用字段本身的OnCheckValid事件，完成补充的校验。

B）没有对应的Domian：

1）根据字段的Checker对象的属性，进行有效性校验；

2）然后调用字段本身的OnCheckValid事件，完成补充的校验。

Checker对象（TWVFieldChecker类）判断有效性的方法是：

1)首先设置Valid=DefaultValid

2)如果字段的值为空(Field.Data.IsEmpty)，Valid=not Required

3)否则

a)对于字符串，检查长度是否合适(如果MaxLength>0)；

b)对于数字，检查数值范围(如果Max-Min>0)。

注意：现在版本的校验顺序与以前不同，现在允许在使用特殊的OnCheckValid事件以前，使用Checker对象完成基本的校验工作。

## 4、关于新的报表组件(TDBReportInfo)

### 原理

1、利用Band的概念建立报表打印的模型。复制Band通过控制器名称和分组索引遍历指定的数据集。概念参见报表制作工具的文档。

2、标签上面的文本数据可以是静态的数据，也可以是动态的数据。动态数据根据字段名获取。字段名可以是数据集的字段名称，也可以是系统参数（例如打印日期、页数），还可以是用户自定义的数据。

### 基本使用方法

1、在操作界面放一个TDBReportInfo组件(KSReport页)。

2、设置DatasetName1~ DatasetName5和DataSource1~ DataSource5。如果只有一个结果集需要被打印，只需要设置DatasetName1和DataSource1，其他依次类推。特别注意DatasetName应该设置为和报表文件里面相同的名字。

3、注意：关于报表文件里面数据集和Controller的命名问题。

如果只有一个结果集需要打印，可以命名为“Main”。如果有多个结果集打印，为了便于识别最后使用有意义的中文名，例如客户对账单需要同时打印客户的资金和证券消息，那么需要两个结果集，分别命名为“资金”、“证券”。注意界面上面应该有两个TDataSource，分别对应相应的结果集。

4、设置BrowseDatasetType为bdtDisableControls，可以优化打印速度。

5、在操作界面的Init方法部分（重载TKSFrame的Init方法），调用KSFrameWorkUtils单元里面的GetReportFileName 初始化TDBReportInfo对象的FileName。例如

DBReportInfo1.FileName := GetReportFileName(‘测试.rpt’);

6、在需要打印的时候（点击了打印键），调用如下代码：

DBReportInfo1.PrepareReport;

DBReportInfo1.Preview; 或者DBReportInfo1.Print。

7、控制打印的页范围。

TDBReportInfo的Preview和Print可以带2个参数：

Preview(StartPage : Integer=0; EndPage : Integer=0)

Print(StartPage : Integer=0; EndPage : Integer=0)

0表示不限制。

8、控制部分控件打印/不打印

首先在用报表工具设计的时候，为需要控制打印/不打印的控件设置ID属性。

在调用TDBReportInfo.PrepareReport，以后调用TDBReportInfo. SetCtrlPrintable。

SetCtrlPrintable(const ID:string; Printable : Boolean)

在调用的时候参数ID等于该控件的ID。

9、通过VariantValues属性或者SetVariantValue方法设置用户定义的数据。

参看：\\sas\Publish\DelphiLib\Demos\Report\SimpleDemo1 里面的例子。

### 属性和方法的介绍

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **说明** | **举例** |
| Title | 属性 | 报表的标题，显示在打印状态对话框里面 |  |
| FieldMapping | 属性 | 字符串列表，说明字段名称的映射关系。可以将报表设计里面的字段名映射为实际的名字。格式是：  报表的字段名=实际字段名 | 包含：  客户号=scust\_no |
| VariantValues | 属性 | 字符串列表，包含用户定义的变量的值（文本类型），格式是：  字段名=值 | 包含：  营业部名称=南头营业部 |
| SetVariantValue | 方法 | 设置用户定义的变量的值。效果等于在VariantValues里面增加一条文本。 | SetVariantValue(‘营业部名称’,’南头营业部’); |
| SetCtrlPrintable | 方法 | 设置控件是否被打印出来 | SetCtrlPrintable(‘余额’,False); |
| FileName | 属性 | 报表格式文件名 |  |
| TextFormatFileName | 属性 | 输出文本格式的报表格式文件名 |  |
| DatasetName1 ～ DatasetName5 | 属性 | 数据集的名称 |  |
| DataSource1 ～ DataSource5 | 属性 | 数据源 |  |
| BrowseDatasetType | 属性 | 遍历结果集时候采取的优化：  bdtNormal：无  bdtBlockRead：设置数据集的BlockReadSize，优化操作，不影响明细(Master-Detail)。  bdtDisableControls：调用数据集的DisableControls，优化操作，影响明细(Master-Detail)。 | 建议一般设置为bdtDisableControls，除非包含明细结果。 |
| PrintToFile | 方法 | 将报表输出到文本文件 |  |

### 文本格式化

现在打印的标签支持文本的格式化处理，主要通过TextFormatType（文字格式类型）属性和TextFormat（文字格式）属性控制。

|  |  |
| --- | --- |
| **TextFormatType** | **说明** |
| tfNone | 无 |
| tfNormal | 使用Format()函数格式化 |
| tfFloat | 使用FormatFloat()函数格式化 |
| tfDateTime | 使用FormatDateTime()函数格式化 |

TextFormat的内容是FormatXXX()函数的格式参数。举例如下：

|  |  |  |
| --- | --- | --- |
| **TextFormatType** | TextFormat | **说明** |
| tfNormal | %s客户开户回单 | 文字合成 |
| tfNormal | 共%d页 | 文字合成 |
| tfFloat | ‘$’#,##0.00 | 控制货币的显示 |
| tfDateTime | yyyy’年’mm’月’dd’日’ | 控制日期的显示 |

### 打印回单

在DataUsageUtils单元（unit）有两个和回单打印相关的公共过程：

// 打印回单

procedure PrintReceipt(const BasedFileName : string; WorkView : TWorkView; AddDefaultSystemParams : Boolean=True);

// 打印回单。根据配置确定是否打印回单，如果要打印，出现对话框，询问操作员，是否打印。返回是否打印

function DoPrintReceipt(const AOperation : string; const BasedFileName : string; WorkView : TWorkView; AddDefaultSystemParams : Boolean=True):Boolean;

回单的数据主要来源于工作视图（WorkView），部分来源于系统变量。系统变量有：#营业部代码、#营业部名称、#职工代码、#职工姓名。

如果AddDefaultSystemParams = True，还有参数“Date”和“Time”分别表示日期和时间。

### 打印报表

使用报表组件(TDBReportInfo)打印报表的时候，数据来源于数据集或者系统参数。系统参数包括：#Date（打印日期）、#Time（打印时间）、#Page（当前页码）。如果希望打印#营业部代码、#营业部名称、#职工代码、#职工姓名这些系统变量，应该在打印以前调用KSFrameWorkUtils单元里面的AddSystemParams方法。例如：

AddSystemParams(DBReportInfo1.VariantValues);

### 高级属性

在打印设计控件的Options属性里面可以设置高级打印特性。属性是文字类型，可以使用“;”间隔多个特性。

|  |  |
| --- | --- |
| **特性名** | **说明** |
| #NoPrintOnFirstPage | 不在第一页打印 |
| #NoPrintOnLastPage | 不在最后一页打印 |
| #PrintOnFirstPage | 仅在第一页打印 |
| #PrintOnLastPage | 仅在最后一页打印 |

## 5、使用系统对话框

### 基本说明

通过Context.GetDialogSystem获得对话框接口，该接口实现一下对话框：

|  |  |
| --- | --- |
| **方法名称** | **说明** |
| Confirm | 出现确认对话框，缺省具有：是、否两个按键，设置AllowActions可以控制显示的按键，增加一个“取消”。 |
| Confirm2 | 出现确认对话框，使用“Y/N”键盘确认/取消。 |
| Warning | 显示警告对话框，显示是、否两个按键。 |
| ShowWarning | 显示警告对话框，只有“确认”一个按键。 |
| ShowMessage | 显示信息对话框，只有“确认”一个按键。 |
| ShowSuccess | 显示操作成功对话框，只有“确认”一个按键。 |

另外：

|  |  |
| --- | --- |
| **方法名称** | **说明** |
| ShowWaiting | 在一个操作以前，显示一个等待对话框。 |
| CloseWaiting | 关闭等待对话框。 |

### 使用指南

#### 1、需要确认的情况

确认对话框根据需要在Confirm和Confirm2里面选择一种。只有少数特殊关键的界面使用Confirm2，其他都使用Confirm。

建议使用公共函数DoPrompt，根据配置选择Confirm和Confirm2里面的一种。

#### 2、警告对话框

如果使用Warning，如果用户选择“否”，应该进行一些特殊处理，例如：客户开户界面，输入的证件号码已经存在，调用Warning弹出警告对话框，如果用户选择“是”，程序不做特殊处理，输入焦点移动到下一个框；如果选择“否”，清空证件号码，焦点保持在证件号码上面。

如果使用ShowWarning，简单一些，没有什么特殊处理。

建议使用Warning。

#### 3、显示错误信息

使用ShowMessage

#### 4、显示成功信息

使用ShowSuccess，不要使用ShowMessage！

#### 5、使用等待对话框

使用ShowWaiting和CloseWaiting，注意使用try...finally

例如：

try

ShowWaiting(SWaitingQuery);

QueryRequest.SendCommand;

finally

CloseWaiting;

end;

## 6、几种和客户相关的类别的含义和处理

### 6.1、几种和客户相关的类别

|  |  |  |
| --- | --- | --- |
| **类别** | **用途** | **维护** |
| **(总部)统计类别** | 用于针对客户的数据统计。全公司统一一套数据。 | 只有总部可以修改该数据。 |
| **(营业部)客户类别** | 每个营业部分别设置的类别，用于数据统计和权限控制。 | 数据由营业部自己维护。 |
| **客户组别** | 每个营业部分别设置的类别，用于数据统计和权限控制。 | 数据由营业部自己维护。 |

### 6.2、程序处理

#### 原来的处理

当总部操作员登录进去，可以选择营业部。但是原来程序里面的数据字典只有登录营业部（这个时候是总部）的一套数据。那么选择客户类别或者客户组别下拉框里面的数据没有根据营业部变化，同时查询处理的客户类别和组别的数据也没有根据营业部进行翻译。所有程序的处理是不正确的。

如果以营业部的操作员登录进去，因为不能选择营业部，查询的数据只是本营业部的数据，所以不会产生错误。

#### 新的处理

为了解决这个问题，程序里面有关数据字典的处理进行了修改，涉及到数据字典的函数增加了一个营业部参数。因为这个参数具有缺省值，所以对以前的调用没有影响。对营业部无关的数据字典，例如“币种”、“市场”等等，都没有影响。只有程序里面涉及到客户类别、客户组别的部分需要修改。

具体参见：**KSFrameWorkUtils**和**KSDataDictionary**。

#### 具体修改内容

修改的内容如下：

1、修改界面，将Label的Caption正确设置，分清除该界面到底使用的是统计类别还是客户类别还是客户组别。

2、如果有选择客户类别的下拉框或者列表框，需要如下处理：

1）将原来设置的ItemsDataEntry设置为“”，保留DataPresentType="IDOBJECTS"。有些列表框不需要"IDOBJECTS"。

2）在响应营业部代码改变的事件（OnValueChanged）里面编写代码，例子如下：

**procedure** TfaSetCustSecType.WorkViewWorkFields0ValueChanged(

WorkField: TWVField);

**begin**

// 选择的营业部发生变化

// 标准的处理

inherited;

// 关闭查询结果

CloseDataset;

// 更新按键的状态

UpdateButton;

// 新增加的处理客户类别的代码段

if WorkField.Data.AsString<>'' then

// 根据营业部初始化客户类别列表

InitDepartCustomerTypes(WorkField.Data.AsString,cbCustType.Items,True) else

// 营业部为空，或者表示所有营业部，将"-"加到列表里面

AddOnlyEmptyItem(cbCustType.Items);

// 复位客户类别的选择。(选择"-")

WorkView.FieldByName('查询客户类别').Reset;

// 强制更新下拉框，保证选中"-"

cbCustType.ForceUpdateText;

**end**;

3、如果查询结果里面有客户类别或者组别需要翻译，需要如下处理：

1）在TWVDBBinder.Bindings里面对应客户类别的项目的DataPresentType和DataPresentParam都设置为“”。（原来是IDOBJECTS和1043）。

2）为TWVDBBinder.Bindings里面对应客户类别的项目的OnBinding事件编写代码，例子如下：

**procedure** TfaSetCustSecType.QueryBinderBindings2Binding(

Binding: TWVDBBinding; ADBField: TField; AWVField: TWVField);

**begin**

inherited;

// TWVDBBinder.Bindings[].OnBinding 事件处理程序

// 将客户类别字段的OnGetText事件指定为GetCustTypeText

ADBField.OnGetText := GetCustTypeText;

**end**;

3）为该界面对象增加一个方法，例子如下：

**procedure** TfaSetCustSecType.GetCustTypeText(Sender: TField;

var Text: string; DisplayText: Boolean);

**begin**

// 注意TWVDBBinder.Bindings[].OnBinding 事件处理程序

// 响应客户类别字段的OnGetText事件，根据营业部代码转换数据到文字

if DisplayText then

KSDataDictionary.GetCustTypeText(Sender, 'sbranch\_code0', Text);

**end**;

例子参见：客户类别对证券类别委托权限设置。

### 6.3、小结

1、数据字典现在和营业部相关。

2、将数据集字段的文字在显示的时候进行转换的方法。

## 7、一些和日期相关的函数

### 7.1、系统日期、工作日期、本地日期的概念

|  |  |  |
| --- | --- | --- |
| **概念** | **含义** | **使用范围** |
| 系统日期 | 是来源于主机服务器的日期，该时间和北京时间（自然日期）相同。 | 1）报表页脚里面的打印日期 |
| 工作日期 | 是交易系统的处于的工作日期，该日期一般和“系统日期”相同，但是在特殊情况下也可以落后或者领先于“系统日期”。“工作日期”代表业务发生的时间。 | 1）界面输入日期的缺省值  2）回单打印里面的发生日期 |
| 本地日期 | 是运行前台程序的本地电脑的当前日期。日期值取决于该台电脑使用者的设定。 | 不使用 |
| 最近报表日期 | 可以打印的最近的报表查询日期 | 1）报表界面输入里面日期的缺省值 |

### 7.2、获取日期

系统启动的时候获得当前的“系统日期”和“工作日期”，保存在上下文对象的参数里面。这些日期是“yyyymmdd”形式的8位数字的字符串。可以通过“**GetSystemDate**”和“**GetToday**”获得这些日期。注意GetToday是定义在KSFrameWorkUtils单元里面的公共函数。

### 7.3、和日期相关的函数

在KSFrameWorkUtils单元里面定义了一些和日期相关的函数，包括：

|  |  |
| --- | --- |
| **定义** | **说明** |
| function **GetToday** : string; | 返回工作日期，格式yyyymmdd |
| function **GetTodayStr** : string; | 返回工作日期，格式yyyy年mm月dd日 |
| function **DBStrDateToDateTime**(const DateStr : string) : TDateTime; | 将数据库的字符串日期格式(yyyymmdd)转换为日期类型 |
| function **FormatDBStrDate**(const DateFormat,DateStr : string) : string; | 将数据库的字符串日期格式(yyyymmdd)格式化为指定的字符串格式。 |
| function **GetSystemDate** : string; | 获得当前系统日期(yyyymmdd)。 |
| function **GetSystemDateStr** : string; | 获得当前系统日期，格式yyyy年mm月dd日 |
| function TComCmdDefinition.**GetReportDay**: string; | 获得可以打印报表的最后一个工作日 |
| procedure **TryRefreshDates**(Silence : Boolean=True); | 尝试更新日期数据。  如果Silence=True，忽略意外。  否则出现错误对话框，询问是否继续，如果不继续，抛出EAbort。 |

### 7.4、和日期格式化相关的常量

在单元里面定义了一些和日期格式化相关的常量，包括：

NormalDateTimeFormat = 'yyyymmdd';

FullDateFormat = 'yyyy''年''mm''月''dd''日''';

### 7.5、在报表和回单里面使用日期

在报表设计里面设置字段名为“#系统日期”(yyyymmdd格式的日期)或者“#系统日期2” （“yyyy年mm月dd日”）可以获得**系统日期**的字符串形式。

因为回单格式需要同时支持标准的回单和回单补打，所以回单设计里面的日期应该使用字段“Date”((yyyymmdd格式)或者“Date2”(yyyy年mm月dd日)。回单打印在调用DoPrintReceipt或者PrintReceipt的时候，最后一个AddDefaultSystemParams参数的值为True，程序自动将当前**工作日期**设置到“Date”和“Date2”里面。回单补打的时候，调用DoPrintReceipt或者PrintReceipt的时候，最后一个AddDefaultSystemParams参数的值为False，需要编码设置“Date”和“Date2”字段。

注意到字段使用的是一个字符串变量的值，所以格式化的形式只能设置为tfNone或者tfNormal，设置为tfDateTime是没有意义的。设置为tfNormal的时候，注意使用“%s”表示该日期（字符串类型），例如设置“文字格式”为“打印日期 %s”。

其他非系统日期应该使用和日期相关的函数获取，然后通过自定义变量的形式设置。

### 7.6 更新日期

可以调用**“TryRefreshDates”**更新日期数据。如果发生意外，如果Silence=True，忽略意外。否则出现错误对话框，询问是否继续，如果不继续，抛出EAbort。

该函数被回单打印“**PrintReceipt**”和获得报表日期“**TComCmdDefinition.GetReportDay**” 自动调用。

### 7.7 （附录）关于日期处理的修改历史

在原来的版本里面没有使用使用“系统日期”，全部使用的是“工作日期”。为了兼容原来的大部分代码，将“GetToday”和“GetTodayStr”定义为“工作日期”，兼容原来的代码。报表里面的“#系统日期”(yyyymmdd格式的日期)或者“#系统日期2” （“yyyy年mm月dd日”）表示“**系统日期”**。回单里面的“Date”和“Date2”表示“工作日期”。

|  |  |
| --- | --- |
| **单元** | **修改内容** |
| KSClientConsts | 增加了日期参数的名称 |
| UContext | 支持更多的日期参数，目前包括“#系统日期”“#工作日期”“#报表日期”三种日期。  在启动的时候获得各种日期。 |
| KSFrameWorkUtils | 修改了GetToday、GetTodayStr重新对应“工作日期”。  增加了GetSystemDate、GetSystemDateStr。  修改了AddSystemParams，将“工作日期”改为“系统日期”。  增加了TryRefreshDates用于更新日期 |
| DataUsageUtils | PrintReceipt继续调用GetToday、GetTodayStr。但是增加对TryRefreshDates的调用，保证回单打印的工作日期正确。 |
| UCommData | 修改命令模式对象，支持三种日期。不再在登录的时候获得“工作日期”。 |
| KSComCmdDefs | 修改命令模式对象，支持三种日期。  修改了TComCmdDefinition.GetReportDay的实现。 |
| UPrintMode | 打印日期改为调用GetSystemDateStr。 |

## 8、长时间操作的处理

对一个需要较长时间进行处理的操作，一般需要在处理的过程中显示一个进度窗口，告诉用户现在进行到什么操作，并且很多时候允许用户取消正在操作。

在集中交易系统里面，对单个的长时间数据库操作，已经提供了进度显示。但是对循环调用类型的操作还没有提供一个整体的进度显示，允许用户取消循环。涉及到的业务包括一些批量打印（批量对账单等、需要多次循环（不是几个市场或者币种的小数量）打印的报表）、批量指定交易等等。

为实现进度显示，编码需要做如下调整：

1）在implement部分的uses里面增加：ProgressShowing和KSClientConsts单元。

2）修改原来循环部分如下：参见“长时间操作.txt”

uses ProgressShowing, KSClientConsts;

procedure MyLongOp;

var

Canceled : Boolean;

begin

Canceled := False;

ShowProgress('操作名称', True, SCancelPrompt);

try

while do

begin

UpdateProgress('当前操作');

if IsProgressCanceled then

begin

// 如果在进度窗口取消，那么退出循环

Canceled := True;

Break;

end;

try

// 完成一个操作

except

// 如果在完成一个操作的时候产生一个意外，询问用户是否继续处理下一个

// 如果不处理，退出循环

on E : Exception do

if IsProgressCanceled or

(Context.GetDialogSystem.ShowError2(

Format(SConfirmIgnoreError, [E.Message])

)<> uaYes) then

begin

Canceled := True;

Break;

end;

end;

end;

finally

CloseProgress; // 保证关闭进度窗口

end;

if Canceled then

Context.GetDialogSystem.ShowMessage('xxx操作被用户终止');

end;

对于批量打印，如果循环的次数少，可以使用“关于报表模版的使用”文档里面介绍的“批量打印”的方法。因为每一个单独的打印都可以被用户取消，抛出一个意外，那么可以在意外处理里面中断循环。

但是如果循环的次数多，例如按照客户号循环，应该使用上面介绍的方法处理。因为可能每个客户的数据少，在打印单个客户的时候由于速度快，根本无法取消单个打印处理。也就是说在用户操作中无法抛出意外，无法通过意外处理中断循环，所以应该使用上面介绍的方法。

## 9、一些编码注意事项

### 9.1、正确使用对话框

操作成功以后（除去参数设置界面，参数设置界面成功以后都不需要弹出对话框！），使用ShowSuccess代替ShowMessage。

### 9.2、关于重载Init方法的两个问题

1）在父类的Init方法里面，将键盘输入焦点移到TabOrder顺序里面（排除Enabled=False的控件）第一个控件上面。对一些涉及到根据登录营业部不是总部的情况下要将营业部下拉框（一般TabOrder=0）设置为Enabled=False的控件。如果使用如下的方式：

procedure XXX.Init;  
begin

inherited;  
// 初始化营业部下拉框  
cbDepartments.Enabled := IsCenterDepartment;

// 其他初始化  
end;

那么键盘焦点无法移到正确的控件上面：因为继承的方法将焦点移到营业部下拉框上面，但是营业部下拉框可能是无效的Enabled=False，造成初始的焦点丢失。

正确的初始化方法如下：

procedure XXX.Init;  
begin

// 初始化营业部下拉框  
cbDepartments.Enabled := IsCenterDepartment;

inherited;

// 其他初始化，例如 FMaintainType := pmtDepartment;  
end;

注意：inherited;应该在其他初始化部分以前，特别是系统参数设置界面，因为inherited部分会设置FMaintainType的初始值，这个初始值可能和该界面需要设置的不相同，所以要放在该界面的其他初始化部分前面执行。

2)如果重载了Init方法，最好在Init的最后调用

UpdateButton; 更新“增加”等按键的有效性。

### 9.3、参数设置与字段有效性

参数设置类型的界面，应该增加仅在增加和修改的时候的起作用的字段的有效性的判断，使得在浏览状态，这些字段的有效性始终是有效的。

实现方法1：是响应OnCheckValid事件，增加如下代码：

if not (status in [ssAdd, ssChange]) then

WorkField.Valid := True;

实现方法2：在IDE里面将OnCheckValid事件的处理方法设置为CheckEditFieldValid。

### 9.4、在正确的事件处理的地方编写代码

尽量在工作视图、工作字段、工作视图的监视器对象的事件上编写代码，不要直接在控件的事件上面编写代码。主要原因如下：

1）在控件的事件上面编写程序灵活性不够，当需要更换控件的时候，例如改换使用特定的数字输入编辑框，必须重新连接事件处理程序。

2）在控件的事件上面编写程序不准确，没有真实体现工作字段的性质（值或者有效性）变化。

3）在控件的事件上面编写程序存在一些问题，一些时候会破坏正常的Windows消息处理。

另外尽量使用数据域。

### 9.5、为代码书写注释

为代码书写基本的注释，例如：

“// 当输入字段有效性改变的时候，改变确认按键的有效性。”

“// 当证券代码或者市场发生改变的时候，取证券信息。”

注释语言的要点是：

1）说明事件被触发的条件，例如“当输入字段有效性改变的时候”、“当证券代码或者市场发生改变的时候”。

2）说明进行的操作，例如“改变确认按键的有效性”、“取证券信息”。

### 9.6、程序的处理有没有基本的优化

涉及到访问数据源的操作，例如“取证券信息”、“取客户资料”等等，在取数据以前应该判断输入数据是否基本有效（符合特定的格式）。例如，“取证券信息”的前提条件是“市场”和“证券代码”非空。“取客户资料” 的前提条件是“客户号”有效，并且符合“客户号”的格式。如果前提条件不满足，不应该向数据源发送请求，应该由本地的代码进行处理，例如清空证券信息、清空客户资料。

### 9.7、有没有对返回数据进行检查

缺省情况下，模版里面的主请求（TWVRequest对象）会自动检查命令的返回值，但是在操作界面里面用到的其他命令请求对象没有这个缺省的处理。应该在命令请求对象的AfterExec事件里面检查返回值。可以将这些命令请求对象的AfterExec事件连接到缺省的事件处理器上面。

### 9.8、检查编译器的警告和提示信息

将程序Build，检查编译器返回的警告（Warning）和提示（Hint）信息，应该尽量修改代码，排除掉任何潜在的错误。

常见的几种情况：

1）变量有定义但是没有被使用到：可能是以前的算法使用到的局部变量，如果新的算法不使用到该变量，将其从声明中去掉。也可能是代码里面出现错误，漏掉了对该变量的使用，应该赶紧检查程序，改正错误。

2）变量可能没有被赋值就被引用：一个非常可怕的潜在错误。一定要在变量赋值以后，才能引用它。特别注意下面错误的try finally的写法：

try

MyObject := TMyObject.Create;

DoSometing;

finally

MyObject.Free;

end;

如果在TMyObject.Create的时候发生意外，这个时候MyObject变量的值是不可预测的，这个时候调用MyObject.Free;会产生严重错误。正确的写法是：

MyObject := TMyObject.Create;

try

DoSometing;

finally

MyObject.Free;

end;

3）对象一个静态方法（不是virtual方法）和父对象的虚方法同名：如果想覆盖父对象的虚方法的行为，应该将子对象的该方法定义为“override”。如果是其他用法，应该将子对象的该方法重新命名。

4）函数的返回值可能没有定义：因为一个函数（function）的条件路径很多，对其中的一些路径，返回值Result可能没有被赋值。检查这些路径，特别注意else。

### 9.9、编码风格是否符合编码规范

## 10、要求操作员确认

## 11、打印回单

## 12、单机双柜员复核

## 13、列表框和滚动条

一些列表框例如银行帐号包含的文字特别宽，解决的办法是在将数据填充到列表框以  
后，调用CompUtils单元里面的过程“AutoHScrollListBox”，将列表框对象作为参数。  
procedure AutoHScrollListBox(ListBox : TCustomListBox);

## 14、改变输入焦点

改变输入焦点的标准做法是调用当前Frame上面的WorkView对应的方法：

TWorkView的方法

procedure SelectPrev(Current : TWinControl); //输入焦点移动到上一个控件

procedure SelectNext(Current : TWinControl); //输入焦点移动到下一个控件

注意参数：Current是当前控件。如果在控件的事件里面编写代码，那么可以使用“WorkView. SelectPrev(TControl(Sender));”的形式。

注意：应该统一到这种用法，其他用法都是不完全正确或者和标准的行为兼容。

## 15、确认时候取消

在用户按下确认键的时候，一般弹出确认对话框，如果用户选择“否”，那么应该将焦点移动到第一个输入控件上面。例子如下：

if not DoPrompt() then

begin

SelectFirst;

Exit;

end;

## 16、使用TObjectList

在公共函数里面，不少通过TObjectList对象作为参数传递多条数据。正确的使用TObjectList的方法是：（以委托帐号为例）

TMyFrame = class(TfaInputTemp)

procedure DoSomething(); // 事件触发，例如字段改变、监控器(Monitor)等等。

private

//保存客户的证券帐号

FAccounts : TObjectList;

protected

public

constructor Create(AOwner: TComponent); override;

destructor Destroy; override;

end;

constructor TMyFrame.Create(AOwner: TComponent);

begin

// 一定在inherited前面创建FAccounts，

// 因为在继承的构造方法里面会调用WorkView.Reset，可能会触发DoSomething

// DoSomething里面使用到FAccounts

// 一定要保证在调用DoSomething以前创建了FAccounts

FAccounts := TObjectList.Create;

inherited;

end;

destructor TMyFrame.Destroy;

begin

inherited;

// 一定要记得释放该对象

FreeAndNil(FAccounts);

end;

procedure TMyFrame.DoSomething();

var

I : Integer;

begin

lsAccounts.Items.Clear;

// 使用FAccounts

// 根据客户号获取股东代码(SecAccounts里面包含TKSAbstractSecAccount)，Market=''表示所有市场

ComCmdDefinition.GetSecAccounts(

WorkView.FieldByName('客户.客户号').Data.AsString,

WorkView.FieldByName('市场').Data.AsString,

False,

FAccounts);

// 使用FAccounts里面包含的数据

lsAccounts.Items.BeginUpdate;

try

for I:=0 to FAccounts.Count-1 do

begin

lsAccounts.Items.AddObject(TKSAbstractSecAccount(FAccounts[I]).Account,FAccounts[I]);

end;

finally

lsAccounts.Items.EndUpdate;

end;

end;

# 十七、编码技巧及其实例

## 技巧标题－书写规范

### 问题的提出

该部分对该技巧需要解决的问题进行说明，交待问题的背景。

### 示例界面

列举遇到该问题的界面。

### 解决方法

具体说明应用该技巧解决问题的方法，可能包括界面设置和书写代码。

### 进一步的说明

对技巧里面涉及到的关键公共函数进行详细的说明。

## 客户号控件的使用和客户号字段的验证：禁止跨营业部、客户状态提示、密码、市场和主股东信息、币种和资金信息。

### 问题的提出

客户号是集中交易系统中客户的唯一标示，并且也是日常工作中使用最为频繁的特殊的值。对于客户号，要求实现补齐，有效性校验，是否允许客户进行跨营业部的操作，客户状态的控制和提示，是否需要输入密码。获得各个市场的主股东信息，获得各个币种的资金信息等功能。

### 示例界面

简单委托，需要输入客户号后，校验客户的存在性，并且获得客户的股东帐号和资金情况。

### 解决方法

使用客户号控件。

客户号控件是一个增加了大量功能的文本输入控件。其属性和方法有很多。详细参见《国通集中交易前台程序开发指南》的“关于客户号输入控件的问题”部分（以下称《开发指南》）。

客户号控件是控件KsAccountEdit,位于UserCtrls控件页。

为什么不使用一般文本控件？因为客户号控件封装了取客户基本资料，而且可以按热键(F2)进行切换不同的帐号（如沪A帐号，深A帐号，牛卡帐号等），极大地方便了输入。在程序开发上，它简化了编程，封装了代码，便于升级和维护。

客户号控件的基本使用。从UserCtrls控件页选取一个KsAccountEdit控件，然后还要从Standard控件页选取一文本标签(Label)。将KsAccountEdit控件的LabelCtrl属性赋值为文本标签(label)的name属性值。该文本标签(label)用于显示帐号的名称(如“客户号”，“沪A帐号”等)。

理解客户号控件与工作视图的关联。可以注意到，客户号控件与其它输入控件相比，一个很大的不同，那就是其它输入控件都有一个FieldName属性，控件就是通过此属性与工作视图关联起来。而客户号控件没有此属性。客户号通过调用交易编码获取客户的各项信息。获取的这些信息分成三类，一类是基本信息，另一类是帐号信息，还有一类资金信息。基本信息包括客户号，客户以及代理人的营业部、姓名、证件类型及证件号码等。帐号信息包括帐号（深A帐号、沪A帐号、深B帐号、沪B帐号），市场等。资金信息包括可取金额、可取现金，可取支票和资金余额等。在客户号控件里，这三类信息分组序号可以在客户号控件里指定。基本信息的分组序号是属性CustInfoGroupIndex，默认为1。帐号信息的分组序号是属性AccountGroupIndex，默认为2,资金信息的分组序号是属性MoneyGroupIndex，默认为3。客户号控件虽然没有显示的FieldName，但是客户号对应的各项客户信息是通过调交易编码获得的，这些信息对应着相应的交易编码里的英文字段。客户号控件和工作视图就是通过这些英文字段跟工作视图里工作字段DataField属性关联起来，关联时要求客户号控件里分组序号与工作视图里的工作字段的GroupIndex相同。如：要把客户号的“客户状态”与工作视图的工作字段“客户状态”关联起来。那么首先确定“客户状态”是调哪个交易编码获得的（具体哪些客户信息是调哪个交易编码获得的，请具体参看《开发指南》）。可以看出，“客户状态”是调用“验证客户 (100101)”这个交易编码，“客户状态”对应“sstatus0”。然后判断“客户状态”是属于哪个分组。分组的判断方法是：调用100101获得的信息项表明是属于基本信息分组；调用100306获得的信息项表明是属于帐号信息分组；调用100313获得的信息项表明属于资金信息分组。找出（或者设置）该分组的分组序号（“客户状态”所在的基本信息分组的分组序号默认为“1”）。有了这些信息后，就可以在工作视图里新建工作字段“客户状态”，设置该工作字段的DataFied，DataField的值要与交易编码里英文字段值一样，在这里设为“sstatus0”，设置该工作字段的GroupIndex，GroupIndex要与分组序号相同，在这里设为“1”。

通常需要考虑设置的一些属性。

如果要设置禁止跨营业部，设置属性ProhibitCrossDepart为True。

如果需要客户输密码，需设置属性NeedPassword属性为True。如果仅仅是让客户在跨营业部时输密码，那么需设置属性NeedPasswordOnCrossDepart为True。

如果界面里要求不许切换帐号，或者只允许某些类型的帐号，需设置AllowMethods

如果界面需用特殊状态的客户，那么需设AllowCustomerStatus。客户号控件会自动检验该客户的状态是否正常，默认情况下，如果不正常会报状态不正常的警示信息。但在有些界面里，允许客户的状态为不正常状态，如在客户号解挂界面里，客户号状态为挂失状态才合理。这时，需将客户号控件的AllowCustomerStatus属性设为“2”（“2”对应数据字典里的挂失状态）。

在客户控件里，你可以有选择地取帐号信息或者资金信息(基本信息默认被取出)。需要取帐号信息，设置NeedMainAccounts属性为True；需要取资金信息，设置NeedMoneyAccounts属性维True。一个客户的帐号不止一个，一般与市场相关，因此取帐号信息时，需设置市场，即要给客户号控件的Market赋值，如果Market值为空，取出的是主帐号的帐号信息。同样，在取资金信息时需设置客户号控件的CurrencyType属性。

### 进一步的说明

客户号控件调用了三个公共函数，分别用来校验客户存在性，获得股东帐号信息和获得资金信息。

CheckExists 校验客户的存在性

GetSecAccounts 获得股东帐号信息

GetMoneyAccounts 获得资金信息

## CheckBox的使用

### 问题的提出

对于简单的选择是否的项目，需要使用CheckBox。

### 示例界面

添加银行帐号，设置自动转款标志时使用了TWVCheckBox的控件。

### 解决方法

TWVCheckBox控件是在TcheckBox上增强了功能的控件。同时和集中交易使用的其他输入控件一样，有WorkView 和 FieldName属性，在使用时需设置这两个属性。

其它常用属性有：

CheckValue和UnCheckValue。

CheckValue表示CheckBox选中时表示的值，UnCheckValue表示CheckBox未选中时表示的值。例如工作视图里有一个工作字段叫“允许撤单”，状态只有两种，一种表示允许，一种表示不允许，那么该工作字段适合于用CheckBox控件来实现。如果我们用“1”表示允许，“0”表示不允许，那么该CheckBox的CheckValue应设为“1”,UnCheckValue应设为“0”。

### 进一步的说明

该控件的所有的功能在基础代码库中定义，是属于整个的工作视图模型的一部分，同具体的业务无关。

## 多选框的使用

### 问题的提出

对于需要进行多选的情况，需要使用多选框CheckListBox

### 示例界面

二级资金试算表，在界面一开始显示和按下‘复位’键后，需要将所有的币种全选。

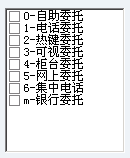
### 解决方法

TWVCheckListbox相当于TWVCheckBox的集合。但在具体使用上有很多不同之处

多选框里表示的数据有两种形式。

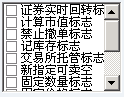
第一种形式是：普通形式

每一项由代码和名称组成，中间用分隔符连接(如“0-自助委托”)。这种形式的多选框很多见，例如[委托方式修改]里的委托方式。



第二种形式是：拼位串形式

每一项只有名称（如：）。每一项选中时表示一个值，未选中时表示一个值，选中时表示另外一个值。这种形式的多选框并不多见，只在[系统管理]|[交易参数]|[证券代码设置]和[证券类别参数设置里]使用到。如[证券代码设置]里的“标志”字段。



如上图中，多选框中的每一项表示一个标志，选中时表示该位为“1”，未选中表示该位为“0”，然后把所有的位拼成一个串。如选中“计算市值标志”，那么整个多选框表示的值位“010000000000”（第二位“计算市值标志”的位值为“1”，其余位的值均为“0”）。

多选框通常需设的属性。见前文中CheckListBox控件的使用。

全选和全不选的问题。

当光标停在多选框上时，按“+”键可以实现全选，按“-”可以实现全不选。

全选和全不选有时需要写程序控制。如需要在界面刚显示时就让多选框全选或者有时要让多选框全不选，那么需在初始化时调用SelectAll函数（全选和全不选共用同一个函数），函数原型为：

如在‘二级资金试算表’的界面中，有如下两段代码

procedure Tfa2TrialBalance.Init;

begin

InitOnlyDepartmentItems(cbDepartment.Items, WorkView.FieldByName('开户营业部'), True);

cbDepartment.Enabled :=IsCenterDepartment;

WorkView.FieldByName('日期').DefaultValue :=ComCmdDefinition.GetReportDay;

WorkView.Reset ;

inherited;

SelectAll(clbCurrencyType,true);

end;

在界面初始化时，调用公共函数SelectAll，将币种列表框全选。

procedure Tfa2TrialBalance.DoClear;

begin

inherited;

SelectAll(clbCurrencyType,true);

end;

在按下‘复位’键后，调用公共函数SelectAll，将币种列表框全选。

### 进一步的说明

调用的公共函数是在单元KSFrameWorkUtils中定义的。

函数的实现如下

procedure SelectAll(CheckListBox:TCheckListBox;SelectAll:Boolean);

var

I: Integer;

begin

for I:=0 to CheckListBox.Items.Count - 1 do

if CheckListBox.ItemEnabled[I] then

CheckListBox.Checked[I] := SelectAll;

if CheckListBox is TWVCheckListBox then

TWVCheckListBox(CheckListBox).InputDataChanged;

end;

首先用参数SelectAll来控制是全选或者全不选，然后如果控件是TWVCheckListBox对象，调用TWVCheckListBox对象的InputDataChanged方法进行数据同步。

## 将最近操作的内容显示在上方的历史记录表格里面。

### 问题的提出

在一些重要的操作界面，如资金和委托中，为了操作员便于查看最近的操作，我们系统里提供了将最近操作的内容显示在表格里的功能。

### 示例界面

见现金存取界面就需要实现这个功能。

### 解决方法

在具体应用时，其实很简单。

首先你需要在界面上加一个listView控件，比如取名为“ListView”。

然后要编写一个初始化记录的函数，比如取名为“InitRec”。在该函数里简单编程。一般如下：

procedure TfaXXX.InitRec(var Rec:Tstrings);

begin

Rec.Add (WorkView.FieldByName('客户.客户号').Data.Asstring);

Rec.Add (WorkView.FieldByName('客户.客户姓名').Data.Asstring);

…

end;

可以看出，该函数只是简单地调用Add方法。

最后剩下的就是编写主体调用代码了。一般应该在重载后的Doit里编程，而且注意应该编写在inherited，即等命令发送完成，而且顺利处理以后。

一般需加下面四行代码（Rec变量需事先声明）：

Procedure DoIt;

…

Rec:=Tstringlist.Create ;

InitRec(Rec);

AddRecToListview(listview1,Rec);

Rec.Free;

End;

### 进一步的说明

在这里调用的公共函数AddRecToListview定义在单元KSFrameWorkUtils中。

实现机理。我们是将历史记录记在一个TlistView类型控件里。我们提供了一个公用函数AddRecToListView。该函数是将一个Tstrings赋值给ListView，而这个Tstrings就是表示一条记录。因此AddRecToListView的功能就是将一条记录记到ListView。而怎样将工作视图里的字段赋值给Tstrings形成一条记录呢？你可以自己编写一个初始化记录的函数，我们一般将该函数取名为InitRec。在该函数里，你可以把工作视图里的字段一个个加进去就行了。

AddRecToListview函数如下，可以自己研究一下。

Procedure AddRecToListview(listview:Tlistview; const Rec:Tstrings;Const MaxRecCount:integer=3);

var

i,j:integer;

begin

with ListView do

begin

//Move to Upper Line with all.

if Items.count>=MaxRecCount then

for i:=0 to Items.Count-2 do

begin

items[i].caption:=items[i+1].caption;

Items[i].SubItems.Clear;

for j:=1 to Rec.Count -1 do

begin

Items[i].SubItems.Add(Items[i+1].SubItems[j-1]);

end;

end;

if Items.count<MaxRecCount then

Items.Add;

//Set Current Record to the Last Line.

items[Items.Count-1].caption:= Rec.strings[0] ;

Items[Items.Count-1].SubItems.clear;

for i:=1 to Rec.count-1 do

begin

Items[Items.Count-1].SubItems.Add(Rec.strings[i]);

end;

end;

end;

以上的代码实际实现了刷新ListView中的数据的工作。

## 根据市场和证券代码取得证券信息。

### 问题的提出

证券代码是系统中的另一个重要的数据，需要实现证券代码的按照不同的市场进行补齐和判断证券代码是否有效，并根据证券代码和市场获得证券的其他信息，如证券简称，证券类别等。

### 示例界面

见简单委托界面就需要实现这个功能。

### 解决方法

可以有两种实现方法：

1. 调公用函数。
2. 证券代码Ex域。

调公用函数的方法是具体调用GetSecInfo函数。该函数的原型为：

procedure TComCmdDefinition.GetSecInfo(const SecCode, Market: string; SecInfos: TObjectList);

由于这种方式要自己编写代码去处理，不太方便，已经不流行了。但它有一个优点，对时效性要求强的必须采用调此公用函数的方法。

另外一种就是使用证券代码Ex域。这种方法是只需设置一些属性就即可，除非要使用证券代码中一些不太常用的标志。

下面来讲一下证券代码Ex域的使用。

和一般的域的使用一样，你须在工作字段的DomainName里设置域，在这里设“证券代码Ex”。和其它普通域不同的是，要在工作字段的Features属性里附加一些信息。这些信息主要包括允许的证券类别、是否必输，证券名称显示到那个工作字段对应的控件里等等，每一行表示一种控制。如：

允许类别=

必输=1

证券对象=证券对象

证券名称=输入证券名称

其中，每一行“=”号前是关键字，系统定义的，必须严格写，“=”号后面的是你自己设置的值。

如果设置了“允许类别”，如设成“允许类别=HH”，那么表示只承认证券类别为“HH”的证券为此时合法的证券，如果为空表示允许所有证券。如果设置成只允许部分证券，一般还要设一行信息，一般类似于“不允许的类别错误=不是可转换债券类型”，表示对输入不是该合法类型证券时报的错误。

“必输”：等于“1”表示必输，不等于“1”表示不必输。

证券对象：后面设置的是工作视图里的某个工作字段，必须是Object类型。存放该证券的各种信息。如需要取一些证券里的标志或者某些指标等，这时需要用到。

证券名称：后面设置的是工作字段的某个工作字段，该工作字段用来显示证券名称。因此，该证券名称一般应该对应一个文本标签控件(label)。

对于使用了“证券对象”的，一般需要编写部分代码来获取需要的信息。一般需要调用TKSAbstractSecInfo的GetParamValue方法来获取信息。具体的实例参见[证券业务]|[简单委托]里的相关代码。

## 单机双柜员复核。

对资金股票或客户业务控制的许多界面需要进行单机双柜员复核。

### 示例界面

现金存取

### 解决方法

对于那些需要单机双柜员的程序，一般需加少量代码，主要是调用DoSingleStationCheck函数。一般需要在界面中重载DoIt方法（普通的设置界面），对于参数设置界面，需要重载ComfirmEdit方法

procedure DoIt;

var

CheckerNo, CheckerPassword,CustNo: string;

…

//检验是否单机双柜员

CustNo:=WorkView.fieldbyname('客户.客户号').Data.Asstring;

if not ComCmdDefinition.DoSingleStationCheck('客户类别修改',CheckerNo,

CheckerPassword,FuncNoModifyCustType,CustNo) then exit;

end

### 进一步的说明

公共函数DoSingleStationCheck定义在单元ComCmdDefs单元中

function TComCmdDefinition.DoSingleStationCheck(

const AOperation : string;

var CheckerNo, CheckerPassword : string;

FuncNo : Integer=0;

const CustomerNo : string = '';

const CurrentType : string = '';

const ACaption : string = ''

) : Boolean;

对各参数做一个简单说明。

Aoperation：指在KsClient.ini配置文件里[doulecheck]部分相应的操作名，一般与菜单同名。

CheckerNo, CheckerPassword：复核职工代码和密码。不是传入参数，而是传出参数。接受这两个参数后可以将它们对应到交易编码的输入参数中去。

FuncNo：该操作对应的功能号。一个界面里操作可能对应多个功能，这时需分别处理。

CustomerNo：客户号。可传空。

CurrentType：货币类型。可传空。

Acaption：复核界面的标题，默认为“复核”，一般不传。

下面是在[客户管理]|[客户类别修改]里的关于单机双柜员的代码。

## 行情显示。

在委托界面上需要显示行情。

### 示例界面

简单委托

### 解决方法

已经做好模板。如果需要在界面上显示行情，你可以直接把简单委托里的行情panel拷贝过去。然后拷贝SetPriceColor和SetSituationColor函数，不需改具体代码即可。

### 进一步的说明

简单介绍一下实现原理，便于理解和应用。

行情显示是调用查询证券行情的交易编码实现的。行情可以自动刷新和手动刷新。自动刷新是在一个定时器(timer)里的调用刷新行情的函数ReFreshSituation。手动刷新自然是在相应对方也调用ReFreshSituation函数。

理解ReFreshSituation函数。该函数有一个参数ShowMsg。主要是用于控制如果行情取不到或者出错了的时候是否报出警示信息。如手动刷新时需要报，而自动刷新时就不需要报，因为如果自动刷新每次都报势必造成有很多的弹出对话框。在ReFreshSituation里，首先判断该页面是否可见，调用CtrlVisible，如果不可见，就不刷新，这主要是用于控制自动刷新时切换到其他界面时不刷新。然后就调用交易编码，将行情赋值到界面控件标签上。最后还要调用SetSituationColor函数，该函数是给行情各数据赋颜色。该函数的参数是比较的基准数，即数据大于该数时表示涨，否则表示跌。

ReFreshSituation函数的实例如下：（摘自[证券业务]|[简单委托]）

//取证券行情

procedure TfaEntrust.ReFreshSituation(ShowMsg:Boolean=false);

begin

if not CtrlVisible(Self) then exit;

WorkView.SynchronizeCtrlsToFields ;

WorkView.Reset(8);

if (WorkView.FieldByName('市场').Data.AsString='') or (not WorkView.FieldByName('市场').Valid) then exit;

if (trim(WorkView.FieldByName('证券代码').Data.AsString )='') or (not WorkView.FieldByName('证券代码').Valid) then exit;

//取行情

try

RqGetSecSituation.SendCommand ;

SetSituationColor(WorkView.FieldByName('昨收盘').Data.AsFloat);

except

if ShowMsg then

Context.GetDialogSystem.ShowMessage('无此证券的行情或者行情应用服务器未启动！');

end;

end;

理解SetSituationColor。该函数就是在ReFreshSituation函数里调用的用于给行情赋颜色。

在这个函数里，首先设置价格数据项的颜色，因为只有价格可以与基准价（昨收盘）比较得出涨或者跌，买卖手数无法与基准价比，但仍需设颜色。然后再设买卖手数的颜色，分别于买卖价颜色相同。然后设置涨跌值的颜色，与0比。在这个函数里，比较并赋值是调用另一个函数SetPriceColor。

下面是SetSituationColor和SetPriceColor的具体代码实现。

procedure TfaEntrust.SetSituationColor(CompPrice:Real);

var

i:integer;

begin

//设置价格数据项的颜色

for i:=0 to Self.ComponentCount-1 do

begin

if (Components[i].Tag =PriceLabelTag) then

SetPriceColor(TWVLabel(Components[i]),CompPrice);

end;

//设置买卖盘数据项的颜色

lbSellVol4.Font.Color :=lbSellPrice4.Font.Color;

lbSellVol3.Font.Color :=lbSellPrice3.Font.Color;

lbSellVol2.Font.Color :=lbSellPrice2.Font.Color;

lbSellVol1.Font.Color :=lbSellPrice1.Font.Color;

lbBuyVol1.Font.Color :=lbBuyPrice1.Font.Color;

lbBuyVol2.Font.Color :=lbBuyPrice2.Font.Color;

lbBuyVol3.Font.Color :=lbBuyPrice3.Font.Color;

lbBuyVol4.Font.Color :=lbBuyPrice4.Font.Color;

//设置涨跌值的颜色

SetPriceColor(lbRiseFallFlag,0);

end;

procedure TfaEntrust.SetPriceColor(WVLabel: TWVlabel; CompPrice: Real);

var

Price:real;

begin

Price :=WorkView.FieldByName(WVLabel.FieldName).Data.AsFloat;

if (Price>CompPrice) then

WVLabel.Font.Color :=RiseColor else

if (Price<CompPrice) then

WVLabel.Font.Color :=FallColor

end;

## 根据存取方式控制客户密码。

一般只有客户取钱时才需要输入密码，用于存取钱和用一个界面，所以需要进行区分

### 示例界面

现金存取

### 解决方法

实现方法主要是当币种发生改变的时候（在币种工作字段的onValueChanged里编程），设置客户号控件的NeedPassWord属性。

下面是实例：（[资金业务]|[现金存取]）

procedure TfaAccessCash.DoIsNeedPasswordByCurrentType(

WorkField: TWVField);

begin

inherited;

{判断是存还是取

如果是存，设置客户号控件属性，不需密码

如果是取，设置客户号控件属性，需要密码

}

if WorkField.Data.IsEmpty then exit;

edCustNo.Reset ;

if WorkField.Data.AsString='1' then

edCustNo.NeedPassword :=true

else

if WorkField.Data.AsString='2' then

edCustNo.NeedPassword :=false;

end;

如果需要实现按存取方式决定是否需要密码，完全可以仿照上面代码，几乎不用修改。

### 进一步的说明

## 根据查询方式控制结果集表格的显示

### 示例界面

综合查询的挂失冻结查询

### 解决方法

{

<Procedure> Doit

<What> 根据操作选择的变化,设定WorkView字段的值并根据查询内容设置DBGrid列名的显示

<Params>

<Exception>

}

procedure TfaQryLoseFrozen.DoIt;

var

QueryContentField:TWVField;

begin

QueryContentField:=WorkView.FieldByName('查询内容');

if WorkView.FieldByName('状态').Data.AsString='0' then WorkView.FieldByName('状态').Data.AsString:='';

if QueryContentField.Data.AsInteger=1 then

begin

Grid.Columns[3].Title.Caption :='市场';

Grid.Columns[3].Visible :=False;

Grid.Columns[4].Visible :=False;

TWVDBBinding(WVDBBinder.Bindings.Items[0]).DataPresentParam := '1301';

Grid.Columns[5].Title.Caption :='帐号';

Grid.Columns[5].Visible :=False;

end;

if QueryContentField.Data.AsInteger=2 then

begin

Grid.Columns[3].Title.Caption :='市场';

Grid.Columns[3].Visible :=True;

Grid.Columns[4].Visible :=False;

Grid.Columns[5].Visible :=True;

TWVDBBinding(WVDBBinder.Bindings.Items[0]).DataPresentParam := '1301';

Grid.Columns[5].Title.Caption :='股东帐号';

end;

if QueryContentField.Data.AsInteger=3 then

begin

Grid.Columns[3].Title.Caption :='银行';

Grid.Columns[3].Visible :=True;

Grid.Columns[4].Visible :=True;

Grid.Columns[5].Visible :=True;

TWVDBBinding(WVDBBinder.Bindings.Items[0]).DataPresentParam := '1601';

Grid.Columns[5].Title.Caption :='银行帐号';

end;

if QueryContentField.Data.AsInteger=4 then

begin

Grid.Columns[3].Title.Caption :='市场';

Grid.Columns[3].Visible :=False;

Grid.Columns[4].Visible :=False;

Grid.Columns[5].Visible :=True;

TWVDBBinding(WVDBBinder.Bindings.Items[0]).DataPresentParam := '1301';

Grid.Columns[5].Title.Caption :='磁卡号';

end;

inherited;

end;

上面这一段代码是在DoIt过程中根据查询内容在控制DBGrid的显示，如果查询的是客户号，DBGrid中第四列，第六列显示市场和帐号,而第四列隐藏；如果查询股东代码的话， DBGrid中第四列，第六列显示市场和股东帐号，而第四列隐藏；如果查询银行的话DBGrid中第四列，第六列显示银行和银行帐号，而第四列显示；

值得注意的是在种显示和隐藏的转换中要注意对字段的绑定。

另外至于要将转换放在DoIt中是因为在查询过程中客户可能要频繁切换分页来看查询结果，这是如果用的是即时转换时，即一旦改变查询内容就改变查询结果集的显示，这时候就可能有问题，因为涉及到一个绑定的问题，会出现绑定错误的情况。而放在DoIt事件的好处就是DoIt只有按下查询键才触发（这是在模板里面写的，具体可察看查询模板），所以可以避免上述情况的发生。

### 进一步的说明

　　如综合查询的开户销户查询。

## 有关单个客户的资料查询

### 示例界面

　综合查询的股东代码查询。

### 解决方法

因为客户资料查询是一种嵌入式的页面,他要和其他的嵌入式界面一起在客户综合信息查询界面中显示,所以别忘了在IntQueryUI50包里的UcustIntQuery.pas单元里要对本界面进行注册,修改PageCount在实现的use中加入界面所在的单元，另外要修改CustQueryPageClasses，UICodes，Captionｓ，另外还要修改UcustQueryFrame的TCustQueryFactory.CreateInstance函数。

值得注意的是

{

<Procedure> DoCheckQueryBtn

<What> 将框架UCustQueryTemp中的客户号输入控件的有效性

传入界面中并控制'查询'的有效性

<Params>

<Exception>

}

procedure TfaQueryCustAccount.DoCheckQueryBtn(Sender: TWVFieldMonitor; Valid: Boolean);

begin

inherited;

InputValidChanged(Valid);

end;

鉴于这种界面的特殊性，他的客户号输入是在模板上的，我们要把客户号的有效性传到本界面中所以必须要在Monitor事件中加入InputValidChanged(Valid);这样才能把模板中对客户号的有效性传入。

在单元的“Initialization”部分编写代码，创建界面工厂，注册该界面的时候，写下如下代码。

TCustQueryFactory.Create('客户股东代码查询',TfaQueryCustAccount,'股东代码');

值得注意的是最后一个参数是和嵌入式页面相关，就是系统显示的Caption。

### 进一步的说明

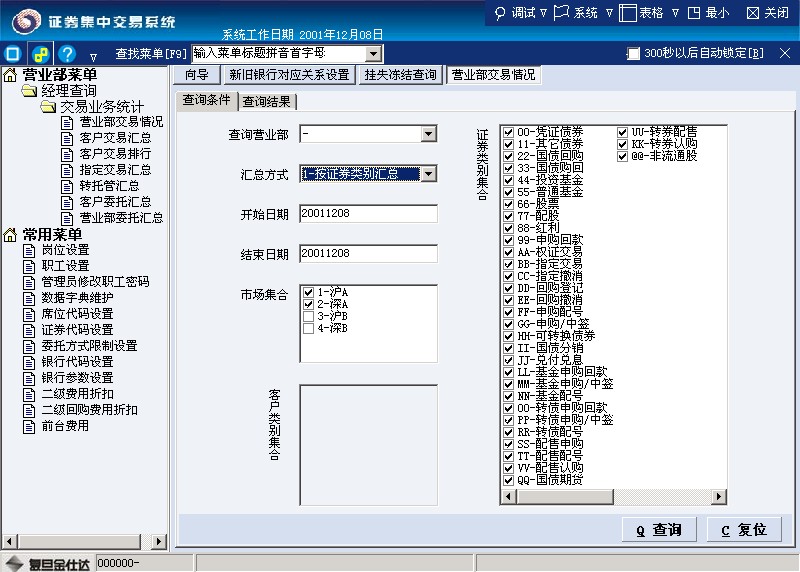
　综合查询的委托明细，资金明细，交易汇总。

## 根据数据字典填充列表

### 示例界面

　经理查询的营业部交易情况汇总

### 解决方法



显示情况如证券类别集合就是根据数据字典显示的。

这个设置是比较简单的只要设置几个属性就可以了，具体如下：

因为这个例子是checklistbox，我们可以在控件的ItemsDataEntry设为1206，另外FieldName属性设为WorkView中的相关字段，本例是“证券类别集合”，还要设置SeperateChar为“，”SeperatedStr为“-”这样就可以了。

如果是下拉框的话不用设置SeperateChar属性，但要设置DataPresentType属性为IDOBJECTS。

另外程序中可能要控制显示的默认值，则可以通过对应的WorkView中的字段的DefaultValue来设置，如默认显示人民币的话，只要DefaultValue设为1就可以了。

### 进一步的说明

经理查询的委托汇总查询，资金业务的资金临时冻结等。

## 日期的缺省值和日期的校验

### 示例界面

　　经理查询的营业部交易情况汇总，报表的特殊客户返佣表。

### 解决方法

日期的默认值一般为当前的系统日期，而报表的日期为当前系统日期的前一个交易日，其他还有一些特殊的情况。

首先是当前的系统日期的设置，一般在界面的Init中写两行代码就可以了，如：

WorkView.FieldByName('开始日期').DefaultValue := GetToday;

WorkView.FieldByName('结束日期').DefaultValue := GetToday;

源代码如下：

{

<Procedure> Init

<What> 初始化营业部名称和日期,重置WorkView

<Params>

<Exception>

}

procedure TfaBranchTxSituation.Init;

begin

InitOnlyDepartmentItems(cbBranchCode1.Items,WorkView.FieldByName('查询营业部代码'),True); // 初始化营业部下拉框

cbBranchCode1.Enabled := IsCenterDepartment;

inherited;

//WorkView.FieldByName('查询营业部代码').DefaultValue := Context.ParamData(svDepartNoName).AsString;

WorkView.FieldByName('开始日期').DefaultValue := GetToday;

WorkView.FieldByName('结束日期').DefaultValue := GetToday;

WorkView.FieldByName('市场代码集合').DefaultValue:='12';

if IsCenterDepartment then Grid.Columns[0].Visible:=True

else Grid.Columns[0].Visible:=False;

WorkView.Reset;

SelectAll(cbCustType,True);

end;

如果是报表日期的话，参考报表的特殊客户返佣表，把那两句话改为

Workview.FieldByName('开始日期').DefaultValue := ComCmdDefinition.GetReportDay;

Workview.FieldByName('结束日期').DefaultValue := ComCmdDefinition.GetReportDay;

源代码

源代码如下：

procedure TfaQryCustCms.Init;

begin

// 初始化营业部下拉框

InitOnlyDepartmentItems(cbBranchCode1.Items,WorkView.FieldByName('开户营业部'),False);

cbBranchCode1.Enabled := IsCenterDepartment;

inherited;

Workview.FieldByName('开始日期').DefaultValue := ComCmdDefinition.GetReportDay;

Workview.FieldByName('结束日期').DefaultValue := ComCmdDefinition.GetReportDay;

Workview.Reset;

SelectAll(cbMoneyType,True);

SendCommandWhenDoIt:=False;

end;

至于校验的话，就要设置控件对应WorkView字段的DomainName;开始日期设为日期校验，结束日期设为日期大小校验。并在结束日期的MonitorValueChange的属性选上开始日期，这样就可以监控开始日期，确保结束日期大于等于开始日期。对于一些要求日期可以不输但是一旦输入就要如此校验的情况，那要选择日期校验\*和日期大小校验\*这两个域，同样结束日期要监控开始日期，另外将两个字段的Constrained设为True。

还有一种特殊的情况，参看客户返佣里面的返佣金额计算。在这个界面中结束日期是系统日期，开始日期是根据结束日期变化的，即是结束日期的前一个月，如：结束日期是20001102，那么开始日期是20001002，另外开始日期是不能输入的，他的ReadOnly属性设为True(这样设置的原因参看设计)，至于校验如上所述。

### 进一步的说明

综合查询的回购交易查询等

## 重输字段的校验

### 示例界面

　资金业务的现金存取。

### 解决方法

　我们只要在重输控件对应的WorkView字段的DomainName属性设为重输校验，并在其MonitorValueChange属性设为要监控的WorkView字段，在本例中输入金额对应的WorkView字段是“资金发生”，所以设为它就可以了，另外将两个字段的Constrained设为True。

### 进一步的说明

　资金业务的支票存取等。

## 查询出来的营业部和设置营业部不相同

### 示例界面

　系统管理的费用设置的二级费用

### 解决方法

　将数据从数据源同步到工作视图（再到各种控件）。要注意的是查询得到的营业部和设置界面的营业部不相同时，需要重载DisplayData，看源代码如下：

procedure TfaSet2CustFeeRate.DisplayData;

begin

inherited;

if status=ssAdd then

WorkView.FieldByName('设置.营业部代码').Reset;

end;

这样切换到增加页面的时候，营业部代码不是查询同步得到的而是，系统设置的默认值。

### 进一步的说明

## 根据具体数据,改变表格里面数据的显示方式

### 示例界面

　　客户返佣的返佣统计计算。

### 解决方法

这是一个根据后台返回的数据中数字来进行翻译成中文（主要针对没有对应数据字典的情况），先看看源代码：

首先在WVDBBinder中选择要进行翻译但是没有数据字典的字段，在本例中选择“status2”，在其OnBinding事件中编写如下代码

procedure TfaCommisionRtnCount.WVDBBinderBindings11Binding(

Binding: TWVDBBinding; ADBField: TField; AWVField: TWVField);

begin

inherited;

ADBField.OnGetText := GetText;

end;

然后另外声明一个GetText过程，如下编写代码。

procedure TfaCommisionRtnCount.GetText(Sender: TField; var Text: string;

DisplayText: Boolean);

begin

if DisplayText then

begin

if Sender=DBGrid1.DataSource.DataSet.FieldByName('sstatus2') then

begin

if Sender.AsString='0' then Text:='现金';

if Sender.AsString='1' then Text:='转帐';

end;

end;

end;

如果后台返回为0，则翻译为现金，如果后台返回1的话就翻译为转帐。

### 进一步的说明

## 根据具体数据,改变报表里面数据的显示方式

### 示例界面

报表的通存通兑资金明细。

### 解决方法

　根据后台返回如果是本代他的话设置报表格式文件中的营业部标题为“开户营业部”，如果是他代本的话显示为“发生营业部”。

procedure TfaEBs.DBReportInfoPrint(Processor: TReportProcessor;

Ctrl: TRPPrintCtrl; ReportInfo: TRPPrintingInfo; var IsPrint: Boolean);

begin

inherited;

if TObject(Ctrl) is TRPPrintItemCtrl then

begin

if TRPPrintItemCtrl(Ctrl).ID='1' then

begin

if DataSource.DataSet.FieldByName('lvol0').AsString='1' then

DBReportInfo.SetVariantValue('报表.营业部标题','开户营业部')

else

DBReportInfo.SetVariantValue('报表.营业部标题','发生营业部');

end;

end;

end;

注意红笔勾勒的代码，其中的ID中的数字是在设计报表格式文件的时候，将“报表.营业部标题”的ID设为1。在程序要判断的原因是这样每次后台返回数据，系统都可以进行转换，否则的话，只有第一次分组的时候显示，后面的分组就不会转换了。

另外uses中要加入RPProcessors,RPPrintObjects,RPBands单元的引用。

相关引用

## 打印回单

许多界面完成了一定的业务后需要打印回单

### 示例界面

添加银行帐号

### 解决方法

在implement部分的uses里面增加：DataUsageUtils.pas单元。

procedure TfaAddBankAcc.DoIt;

begin

…………

…………

[**DoPrintReceipt**](#DoPrintReceipt) ( '添加银行帐号' , '银行开销户回单.rpt' , WorkView ) ;

…………

…………

end;

### 进一步的说明

**函数原型：**

function DoPrintReceipt (const AOperation : string; const BasedFileName : string; WorkView : TWorkView; AddDefaultSystemParams : Boolean=True):Boolean;

**函数参数说明：**

Aoperation：即ksclient.ini配置文件中[PrintReceipt]段下定义的名称

BasedFileName：即报表文件的文件名（一定要有扩展名**.rpt**）

WorkView：即Frame界面上的工作视图的Name

AddDefaultSystemParams：是否增加默认的系统参数，例如日期，时间等

**函数功能说明：**

根据配置确定是否打印回单，如果要打印，出现对话框，询问操作员，是否打印。

返回是否打印

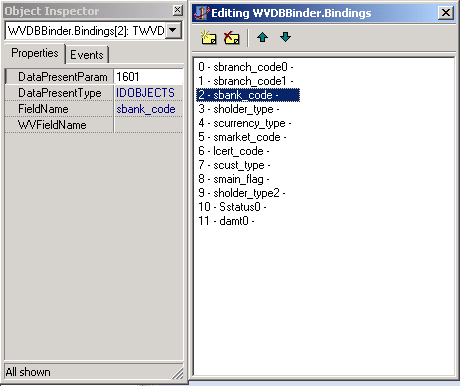
## 在表格里面根据数据字典翻译

数据字典是本系统的一个重要的功能，除了在列表框和选择下拉框中使用外，在表格中根据数据字典的条目进行翻译也是很重要的

### 示例界面

设置营业部对应银行。

### 解决方法



在  WVDBBinder控件里增加项目后设置参数即可完成数据字典的翻译

**相关参数设置说明：**(见上图中<Delphi>的Object Inspactor)

DataPrasentParam: 即数据字典条目

DataPresentType: 固定填写“**IDOBJECTS**”

FieldName: 对应的参数名称

### 进一步的说明

关于数据字典和**IDOBJECTS**可以参见前文的数据字典的部分。

## 工作字段的缺省值。

对于工作字段，有时需要赋上缺省值。特别是数字型的工作字段

### 示例界面

简单委托

### 解决方法

一般地，对下列类型的工作字段缺省值进行说明

1. 字符串型（String）
2. 整数型和浮点型（Integer & Double）
3. 字符串型（String）
   1. 字段是一数据字典条目

如允许为空“-”，则字段的DefaultValue属性值不设置任何值，但Type属性应设置为“**String**”；如果不允许为空，应在“Init”过程中编写代码进行设置。

例：WorkView.FieldByName('货币类型').DefaultValue := GetFirstItemID ( deCurrency );

**GetFirstItemID函数**：取一字符串列表的第一项

**DeCurrency:** 全局变量，对应于“币种”的数据字典条目

* 1. 字段是一日期

应在“Init”过程中编写代码进行设置。

例：WorkView.FieldByName('起始日期').DefaultValue := GetToday;

WorkView.FieldByName('结束日期').DefaultValue := GetToday;

**GetToday:** 取工作日期

* 1. 特别注意：“营业部”字段的设置

应在“Init”过程中编写代码进行设置。

例：InitOnlyDepartmentItems(cbBranchCode.Items,WorkView.FieldByName('发生营业部'),True);

1. 整数型和浮点型（Integer & Double）

**必须**对字段的DefaultValue属性设置一合适的值（一般地，设置值为“0”），否则在界面“确定”操作时会引起“字段数据类型”错误

## 批量打印

有许多的报表需要实现批量打印的功能。

### 示例界面

客户成交公告

### 解决方法

在implement部分的uses里面增加：ProgressShowing和KSClientConsts单元。

1、 **在DoPrintOneReport界面里面先查询数据，然后调用“DefaultPrint”完成实际的打印**。

procedure MyFrame.**DoPrintOneReport** (Currency, Bank: TWVField);

begin

SendCommand; // 获得数据

……………………

//Currency, Bank报表表头参数值的设置

……………………

**DefaultPrint; //** 完成实际的打印。

end;

2、 **在RoundOperation过程中完成一个操作使用了DoPrintOneReport**

procedure TfaBankTrfAccDetail.**RoundOperation**(Currency, Bank: TWVField);

var

I: Integer;

Canceled : Boolean;

begin

Canceled := False;

ShowProgress('银行转帐明细', True, SCancelPrompt);

// 银行转帐明细---操作名称（一般设置为Frame界面的标题）

// ScancelPrompt---一全局常量，= '按ESC键或者鼠标双击这里终止等待';”

try

for I := 0 to clbBankCode.Items.Count - 1 do

begin

if clbBankCode.Checked[I] then

begin

Currency.Data.SetString(TKSAbstractBankInfo(FBankCodeList[I]).Currency);

Bank.Data.SetString(TKSAbstractBankInfo(FBankCodeList[I]).BankNo);

UpdateProgress('正在对'+clbBankCode.Items[I]+'进行银行转帐明细......');

// UpdateProgress(当前操作);

// 当前操作---含有运行值的运行状态描述

if IsProgressCanceled then

begin

// 如果在进度窗口取消，那么退出循环

Canceled := True;

Break;

end;

try

// 完成一个操作

[**DoPrintOneReport**](#DoPrintOneReport)(Currency, Bank);

except

// 如果在完成一个操作的时候产生一个意外，询问用户是否继续处理下一个

// 如果不处理，退出循环

on E : Exception do

if IsProgressCanceled or (Context.GetDialogSystem.ShowError2(

Format(SConfirmIgnoreError, [E.Message]))<> uaYes) then

begin

Canceled := True;

Break;

end;

end;

end;

end;

finally

CloseProgress; // 保证关闭进度窗口

end;

if Canceled then

Context.GetDialogSystem.ShowMessage('银行转帐明细操作被用户终止');

end;

**3、 重载DoPrintReport过程，使用RoundOperation**

procedure TfaBankTrfAccDetail.**DoPrintReport**;

begin

………………………

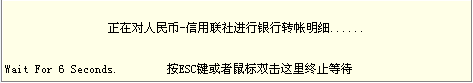
**报表名称和表头的字段值的设置**

………………………

[**RoundOperation**](#RoundOperation)(Workview.FieldByName('货币类别'), Workview.FieldByName('银行代码'));

end;

代码的功能在操作时的体现



### 进一步的说明

对于报表模板的使用请参见前文关于报表模板的使用

## 根据查询方式（按照营业部查询、按照客户号查询），控制

在一些查询界面，需要根据查询方式（按照营业部查询、按照客户号查询），来控制是否需要输入客户号。

### 示例界面

客户资金业务汇总

### 解决方法

在implement部分的uses里面增加：KSComCmdDefs.pas单元。

procedure TfaQryBankTransfer.DoGetQueryControlOnQueryMethodChanged(

WorkField: TWVField);

begin

inherited;

ComCmdDefinition.[**QueryControl**](#QueryControl)(WorkField.Data.AsString,

WorkView.FieldByName('客户号'),

WorkView.FieldByName('发生营业部'),

edCustNo, cbBranchCode);

end;

### 进一步的说明

**函数原型：QueryControl**定义在KSComCmdDefs.pas单元

procedure TComCmdDefinition.**QueryControl**(const QueryMethod: String;

CustField, DeptField: TWVField; CustNo: TKSAccountEdit;

BranchCode: TWVComboBox);

begin

CustNo.Reset;

if QueryMethod = '1' then //按营业部查询

begin

CustField.Constrained := False;

DeptField.Reset;

CustNo.Enabled := False;

CustNo.Required := False;

BranchCode.Enabled := IsCenterDepartment;

end else //按客户号查询

begin

CustField.Constrained := True;

DeptField.Data.Clear;

CustNo.Enabled := True;

CustNo.Required := True;

BranchCode.Enabled := False;

end;

end;

**函数参数说明：**

QueryMethod：查询方式 （1-按营业部查询；2-按客户号查询）

CustField: “客户号”字段

DeptField: “营业部”字段

CustNo: “客户号”编辑控件

BranchCode: “营业部”下拉框控件

**函数功能说明：**

在特定查询方式下，相关项目的Enabled属性和显示控制（用于查询界面）

## 营业部的数据和总部的数据用不同的颜色显示。

当同时查询出的数据既有营业部的数据又有总部的数据时，为了区分，往往需要用不同的颜色显示。

### 示例界面

二级清算费用

### 解决方法

调用以下的方法。

**DBGrid的OnDrawColumnCell事件引用了GridDrawColumnCell过程**

procedure TfaSetParamsTemp.**GridDrawColumnCell**(Sender: TObject;

const Rect: TRect; DataCol: Integer; Column: TColumn;

State: TGridDrawState);

begin

if FDepartmentWVField.Data.AsString = **CenterDepartNo** then

// 全局常量，在KSConstsClint.pas单元中；CenterDepartNo = '000' 总部营业部代码

[**DrawCellForSysParams**](#DrawCellForSysParams)(TDBGrid(Sender),Rect,DataCol,Column,State,'')

else

DrawCellForSysParams(TDBGrid(Sender),Rect,DataCol,Column,State,

**FDepartmentFieldName**);

//FDepartmentFieldName参数设置模板的一个私有变量，为DBGrid中“营业部”的// FieldName, **在Init过程中会有赋值**

end;

### 进一步的说明

**使营业部的数据和总部的数据用不同的颜色显示的真正过程**

procedure**DrawCellForSysParams**(

Grid : TDBGrid;

const Rect: TRect;

**DataCol**: Integer; // 列数组中列的Index

**Column**: TColumn; // 描述Grid中Cell的显示属性和字段绑定的Tcolumn对象

**State**: TGridDrawState; // 关于Grid中Cell在被重新着色的状态信息，

// 说明Grid中Cell是否具有“输入激活”

const **DepartFieldName** : string); //“营业部”栏位名称

var

Dataset : TDataset;

begin

If (**State=[]**) then

begin

if Grid.DataSource <> nil then

Dataset := Grid.DataSource.Dataset

else

Dataset := nil;

if (Dataset <> nil) and (**DepartFieldName <> ''**) and

(Dataset.FieldByName (DepartFieldName).AsString = **CenterDepartNo**) then

begin

**Grid.Canvas.Brush.Color := clBtnFace;**

end;

end;

Grid.**DefaultDrawColumnCell** (Rect,DataCol,Column,State);

// OnDefaultDrawColumnCell是TCustomDBGrid的事件

end;

## 其他批量处理：批量指定交易

基本同“批量打印”

## 根据客户号取得银行帐号

根据客户号取得银行帐号来填充

### 示例界面

银行账号挂失

### 解决方法

在implement部分的uses里面增加：KSComCmdDefs.pas和KSCustomers.pas单元。

通过‘客户号’，‘银行代码’，‘货币类型’获得银行帐号信息

procedure TfaXXXXXX.WorkViewFieldsMonitors1ValueChanged(

Sender: TWVFieldMonitor; Valid: Boolean);

var

BankAccs: TObjectList;

begin

Inherited;

clbBankAccs.Items.Clear;

if not Valid then exit;

BankAccs:=TObjectList.Create;

try

ComCmdDefinition.GetBankAccounts(

WorkView.FieldByName('客户号').Data.AsString,

WorkView.FieldByName('银行代码').Data.AsString,

WorkView.FieldByName('货币类型').Data.AsString,BankAccs);

GetObjectsItems(clbBankAccs.Items,BankAccs);

finally

BankAccs.Free;

end;

end;

这个程序中Monitors的工作字段包括'**客户号**'，'**银行代码**'和'**货币类型**'，并且已经做了优化。用这个方法可以获得该客户的所有相应的银行帐号的信息并填充到clbBankAccs(银行帐号组合框)中。

### 进一步的说明

公共函数GetBankAccounts定义在单元KSComcmdDefs中。他调用了交易编码100307

**调用交易编码100307--取客户银行帐号**

输入参数：

客户号 scust\_no

银行代码 sbank\_code

货币类型 scurrency\_type

查询方式 lvol0 0取所有 1取连通

营业部代码 sbranch\_code0

职工代码 semp

网卡号 sstation0

输出参数：

银行代码 sbank\_code

货币类型 scurrency\_type

银行帐号 sbank\_acc

存折姓名 sname

帐户状态 sstatus0

自助转款标志 sstatus1

市场代码 smarket\_code

股东代码 sholder\_ac\_no

证件类别 lvol0

证件号码 scert\_no

开户连通标志 sstatus2

开户连通日期 sdate0

在上面的调用方法中有**相关的类TKSAbstractBankAccount**，这个类提供的属性包括Bank（银行代码），Account（银行帐号号码），AccountName（银行帐号姓名）。这些属性足以完成通常的功能，如果需要取出其他的字段，可以使用GetParamValue方法。如 sstatus2 ：= TKSAbstractBankAccount(BankAccs [0]).GetParamValue(‘sstatus2’)可以取出开户连通标志。

## 更新市场的多选，控制证券代码的输入

在本系统中，有很多的地方需要输入证券代码，在程序框架中定义了证券代码，证券代码\*和证券代码Ex三个域，用来统一控制证券代码的输入校验，补齐和获得证券信息。由于证券代码是跟市场密切相关的，所以使用这三个域，都需要将‘证券代码’这个工作字段的MonitorValueChangedFields属性设置为‘|市场|’（‘证券代码’和‘|市场|’需要根据实际使用的工作字段进行调整）

但是在有的时候需要对市场进行多选，比如使用选择列表框，这个时候就需要对是否可以输入证券代码进行控制，因为当不选择任何一个市场或者选择了不止一个市场时，对证券代码的输入校验，补齐和获得证券信息就会出现问题。对此，我们的解决方案就是当不选择任何一个市场或者选择了不止一个市场时使得证券代码的输入框无效。

### 示例界面

国债抵押查询

### 解决方法

以下的代码具体完成了这个工作：

procedure TfaQryPubDebtMort.DoMakeStockEnabledByMarket(

WorkField: TWVField);

begin

inherited;

WorkView.FieldByName('证券代码').Data.Clear;

if Length(WorkField.Data.AsString) = 1 then

edStockNo.Enabled := True

else

edStockNo.Enabled := False;

end;

这个过程是工作字段‘市场代码集合’的ValueChanged事件的处理程序。

首先将工作字段‘证券代码’清空，然后判断字段的长度，当长度为1时，使得证券代码的输入框edStockNo有效，当长度不为1时，使得证券代码的输入框edStockNo无效。

## 在表格中根据营业部控制客户类别和客户组别。

在所有的数据字典的条目中客户类别和客户组别是比较特殊的，因为对于不同的营业部客户类别和客户组别可以不同，所以在表格中不能简单的通过设置相应的数据字典条目的方法来进行修改。必须进行额外的处理。

### 示例界面

二级费用折扣

### 解决方法

以下的两部分代码具体完成了这个工作：

procedure TfaSet2FeeDiscount.DoGetCustomerTypeText(Binding: TWVDBBinding;

ADBField: TField; AWVField: TWVField);

begin

ADBField.OnGetText := GetCustTypeText;

end;

procedure TfaSet2FeeDiscount.GetCustTypeText(Sender: TField;

var Text: string; DisplayText: Boolean);

begin

if DisplayText then

KSDataDictionary.GetCustTypeText(Sender, 'sbranch\_code0', Text);

end;

DoGetCustomerTypeText是数据集绑定对象（TWVDBBinder）的客户类别这个数据绑定对象的OnBinding事件的处理程序，数据集绑定对象负责将工作视图中的查询结果集对象（实际上是一个Dataset对象）和数据源对象（DataSource）绑定到一起，并可以转换Dataset对象中的数据。

转换Dataset对象中的数据实际上是通过改变Dataset对象所包含的TField对象的Text来实现的。

DoGetCustomerTypeText实现了将GetCustTypeText这个函数指定为ADBField这个TField对象的OnGetText事件的处理函数。

GetCustTypeText则调用了KSDataDictionary单元的公共函数GetCustTypeText进行处理。

### 进一步的说明

公共函数GetCustTypeText定义在KSDataDictionary单元

procedure GetCustTypeText(Sender: TField; const DepartFieldName : string;

var Text: string);

var

IDObjects : TIdObjectList;

DepartField : TField;

begin

DepartField := Sender.Dataset.FindField(DepartFieldName);

if (DepartField<>nil) and (DepartField.AsString<>'') then

begin

IDObjects := NeedIdObjectList(deDepartCustomerType,DepartField.AsString);

IDObjects.GetText(Sender, Text, True);

end;

end;

这个函数首先通过参数DepartFieldName（Dataset中的字段名）来获得数据集中的字段对象，这个字段对象包含的内容事实上应该是营业部的代码，然后就可以通过调用NeedIdObjectList来获得数据字典中的内容。

## 用一个程序界面实现多个类似的操作功能

有一些界面的功能和内容是相当类似的，为了减少代码量和方便维护，可以用一个程序界面实现多个类似的操作功能。

### 示例界面

股东代码挂失，股东代码解挂，股东代码冻结，股东代码解冻

### 解决方法

以下就是一个具体的例子，用同一个程序界面实现了四个界面的功能。

TUpdateHolderStatusClassFactory = class(TUIStdClassFactory)

private

FOperator: Integer;

protected

function CreateInstance : TObject; override;

public

property Operator : Integer read FOperator write FOperator;

end;

const

TitleLoseHolderAcc='股东代码挂失';

TitleUnLoseHolderAcc='股东代码解挂';

TitleFrozenHolderAcc='股东代码冻结';

TitleUnFrozenHolderAcc='股东代码解冻';

function TUpdateHolderStatusClassFactory.CreateInstance: TObject;

begin

Result := inherited CreateInstance;

faUpdateHolderStatus:= TfaUpdateHolderStatus(Result);

faUpdateHolderStatus.Operator:= self.Operator;

with faUpdateHolderStatus do begin

case Operator of

1 :

begin

Caption := TitleLoseHolderAcc;

HelpID:=TitleLoseHolderAcc;

WorkView.FieldByName('显示操作类别').DefaultValue:=TitleLoseHolderAcc;

KSAccountEdit.NeedPassword := True;

KSAccountEdit.ProhibitCrossDepart := False;

KSAccountEdit.NeedPasswordOnCrossDepart := True;

end;

2 :

begin

Caption:=TitleUnLoseHolderAcc;

HelpID:=TitleUnLoseHolderAcc;

WorkView.FieldByName('显示操作类别').DefaultValue:=TitleUnLoseHolderAcc;

KSAccountEdit.NeedPassword := True;

KSAccountEdit.ProhibitCrossDepart := False;

KSAccountEdit.NeedPasswordOnCrossDepart := True;

end;

3 :

begin

Caption:=TitleFrozenHolderAcc;

HelpID:=TitleFrozenHolderAcc;

WorkView.FieldByName('显示操作类别').DefaultValue:=TitleFrozenHolderAcc;

KSAccountEdit.NeedPassword := False;

KSAccountEdit.ProhibitCrossDepart := True;

end;

4 :

begin

Caption:=TitleUnFrozenHolderAcc;

HelpID:=TitleUnFrozenHolderAcc;

WorkView.FieldByName('显示操作类别').DefaultValue:=TitleUnFrozenHolderAcc;

KSAccountEdit.NeedPassword := False;

KSAccountEdit.ProhibitCrossDepart := True;

end;

end;

WorkView.FieldByName('操作类别').DefaultValue :=Operator;

end;

end;

var

LoseHolderClassFactory : TUpdateHolderStatusClassFactory;

UnLoseHolderClassFactory : TUpdateHolderStatusClassFactory;

FrozenHolderClassFactory : TUpdateHolderStatusClassFactory;

UnFrozenHolderClassFactory : TUpdateHolderStatusClassFactory;

initialization

LoseHolderClassFactory := TUpdateHolderStatusClassFactory.Create(TitleLoseHolderAcc,TfaUpdateHolderStatus);

LoseHolderClassFactory.Operator := 1;

UnLoseHolderClassFactory := TUpdateHolderStatusClassFactory.Create(TitleUnLoseHolderAcc,TfaUpdateHolderStatus);

UnLoseHolderClassFactory.Operator := 2;

FrozenHolderClassFactory := TUpdateHolderStatusClassFactory.Create(TitleFrozenHolderAcc,TfaUpdateHolderStatus);

FrozenHolderClassFactory.Operator := 3;

UnFrozenHolderClassFactory := TUpdateHolderStatusClassFactory.Create(TitleUnFrozenHolderAcc,TfaUpdateHolderStatus);

UnFrozenHolderClassFactory.Operator := 4;

end.

## 报表控件的使用

报表控件的使用参见报表模板的使用

## 将全选转换成全不选

在一些查询界面上放置了列表框用来输入查询条件，如果不选择任何一项，表示不作为查询条件进行控制，如果选择了列表框中所有的项目，如果其效果同不选择任何一个项目完全相同，那么为了实现优化，就可以考虑将全选转换为全不选。为什么要这样做呢，这同后台的具体实现有关，如果不选择任何一项，后台在查询处理时就会完全忽略这个条件，如果选择了所有的项目，后台处理时就不得不加上这个条件，而从结果上来说这两种情况是完全相同的，所以为了优化和提高查询的效率，我们就可以这样去做。

### 示例界面

国债抵押查询

### 解决方法

以下的代码具体完成了这个工作：

procedure TfaQryPubDebtMort.DoChangeMarketSelect(FieldData,

ParamData: TKSDataObject);

begin

MakeSelectAllToNoSelected(FieldData,ParamData,clbMarket);

end;

以上的代码是命令请求对象(TWVRequest)的‘市场代码集合’这个工作字段参数绑定对象（TWVFieldBinding）的OnBindInput事件的处理程序。

命令请求对象(TWVRequest)负责在工作视图对象（TWorkView）和命令对象（TWVCommand）之间传送数据。命令请求对象的SendCommand方法的实际作用就是创建一个命令对象，将命令请求对象所包含的所有的工作字段的数据传送给命令对象的相应参数，命令对象向命令处理对象请求处理并获得返回数据，然后再传送回工作视图中的相应字段，最后释放命令对象。

对于工作字段参数绑定对象，将数据从工作字段传送到命令对象所采用的机制是：如果OnBindInput事件有相应的处理函数，就使用该函数进行处理，该函数有两个参数FieldData工作字段和ParamData参数字段，分别表示对应的工作字段和参数字段。如果OnBindInput事件没有处理函数，就使用默认的方式ParamData.Assign(FieldData)将工作字段的值简单的原封不动的传送给参数字段。

在这个函数中，调用了公共函数MakeSelectAllToNoSelected将全选转换为全不选，这个公共函数除了FieldData和ParamData这两个参数以外，还需要另外一个参数clbMarket，这是一个TWVCheckListBox对象，并且同FieldData相关联，在本例中也就是市场代码选择列表框。

### 进一步的说明

公共函数MakeSelectAllToNoSelected定义在单元KSFrameWorkUtils中，完成转换的工作。

## 将工作视图中的一个字段分拆成几个字段发送给后台

在某些情况下，输入的参数可能相当长，比如输入业务代码作为查询条件，由于在系统中使用的cpack包中的容量最大的参数也只能存放200个字符。所以就必须将输入参数拆分成好几个字段进行传输，到了后台再将这几个字段拼接起来。

### 示例界面

未交收交易查询

### 解决方法

以下的代码具体完成了这个工作：

procedure TfaQryUndeliverTx.DoDistribute;

var

s: string;

begin

s:= WorkView.FieldByName('业务类别集合').Data.AsString

if Length(s) > 0 then

begin

WorkView.fieldByName('业务类别集合1').Data.SetString(Copy(s,1, 200));

WorkView.fieldByName('业务类别集合2').Data.SetString(Copy(s,201,200));

WorkView.fieldByName('业务类别集合3').Data.SetString(Copy(s,401,100));

WorkView.fieldByName('业务类别集合4').Data.SetString(Copy(s,501,100));

WorkView.fieldByName('业务类别集合5').Data.SetString(Copy(s,601,100));

end;

end;

procedure TfaQryUndeliverTx.DoIt;

begin

DoDistribute;

inherited;

end;

模板的DoIt方法实现了查询的工作，所以必须在界面程序中重载DoIt方法，首先调用DoDistribute函数将业务类别集合这个字段的值拆分到五个工作字段中去，然后再调用父类的DoIt方法。

# 第五部分 程序清单

# 十八、程序清单

## 1、工程文件一览

|  |  |  |
| --- | --- | --- |
| **目录** | **工程文件** | **说明** |
| $SOURCE | 国通50.bpg | 包含所以的工程文件的工程组(Project Group)。 |
| $SOURCE\Framework | KSFramework50.dpk | 构造集中交易程序框架的程序包的工程文件。 |
| $SOURCE\Framework | DKSFramework50.dpk | 设计时候需要的程序包 |
| $SOURCE\Main | KSClient50.dpr | 构造主程序的工程文件。 |
|  |  |  |

以下对各个工程文件进行说明。

注意：对具体的界面模块只需要列出几个界面单元示例即可。

## 2、KSFramework50.dpk

构造集中交易程序框架的程序包的工程文件。

包含文件一览

|  |  |  |
| --- | --- | --- |
| **目录** | **文件** | **说明** |
| $SOURCE\Framework | KSFrameWorks.pas | 定义程序框架 |
| KSClientConsts.pas | 前台框架程序中使用到的常量 |
| KSFrameWorkUtils.pas | 辅助的函数 |
| KSFrames.pas | 操作界面的基类 |
| $SOURCE\Templates | UInputTemp.pas  UInputTemp.dfm | 输入界面模版 |
| UQueryTemp.pas  UQueryTemp.dfm | 查询界面模版 |

## 3、KSClient.dpr

构造主程序的工程文件。

|  |  |  |
| --- | --- | --- |
| **目录** | **文件** | **说明** |
| $SOURCE\Main | UMain.pas  UMain.dfm | 主程序界面 |
| UDatabase.pas  UDatabase.dfm | 数据库连接 |
| UContext.pas  UContext.dfm | 1、实现了上下文对象，提供运行环境  2、负责启动和关闭系统 |
| KSMenus.pas | 菜单处理 |
| UFieldDomains.pas  UFieldDomains.dfm | 统一的字段域 |
| UStyles.pas  UStyles.dfm | 统一的界面风格定义：字体、背景颜色、按键。 |
| UBrowser.pas  UBrowser.dfm | 实现向导 |
| USearch.pas  USearch.dfm | 实现界面搜索 |
| UWait.pas  UWait.dfm | 实现等待界面 |

# 十九、公共的域

所有的公共域在KSComCmdDefs文件内，一般的只需在WorkView的相应的工作字段的DomainName属性中使用相应的域名即可，少数需要和其他工作字段的值进行比较，需要在MonitorValueChangedFields属性中进行指定。(如重输校验)

## 1、正整数和正整数\*

正整数（fdInteger），大于等于零的整数，范围从0—10000000000（100亿），只接受数字输入。

正整数\*（fdInteger2），大于零的整数，范围从0—10000000000（100亿），只接受数字输入。

## 2、正实数和正实数\*

正实数（fdDouble），大于等于零的实数，范围从0—10000000000（100亿），接受数字和符号输入，不接受字母输入。

正实数\*（fdDouble2），大于零的实数，范围从0—10000000000（100亿），接受数字和符号输入，不接受字母输入。

## 3、比例和比例\*

比例（fdRate），0到1的实数，接受数字和符号输入，不接受字母输入。

比例\*（fdRate2），严格大于0小于1的实数，接受数字和符号输入，不接受字母输入。

## 4、选择项目

选择项目（fdChoice），针对组合框，必须选择一个值。

## 5、日期校验和日期校验\*

日期校验（fdDateCheck），日期：YYYYMMDD (四位年2位月2位日)。只接受数字输入，必须是8位并是真正的日期。

日期校验\*（fdDateCheck2）日期：YYYYMMDD (四位年2位月2位日)。只接受数字输入，必须是8位并是真正的日期。但是也可以不输入。

## 6、日期大小校验

日期大小校验（fdDateReCheck），日期：YYYYMMDD (四位年2位月2位日)并且需要大于等于开始日期。其他规则和日期校验一样，并且必须大于开始日期。一定要在MonitorValueChangedFields属性中指定要进行比较的开始日期的工作字段

## 7、重输校验

重输校验（fdReinputCheck），同进行比较的工作字段的值必须相同，输入方式不限制。一定要在MonitorValueChangedFields属性中指定要进行比较的工作字段

## 8、证券代码和证券代码\*

证券代码（fdStockNo），6位整数，只接受数字输入，并实现自动补齐的功能。如果在工作视图中有工作字段‘市场’，尽量在工作字段‘证券代码’的 MonitorValueChangedFields属性中加上工作字段‘市场’，可以实现按市场进行补齐。为空时Valid为False, 证券代码为必输时使用

证券代码\*（fdStockNo2），基本同证券代码，但为空时Valid为True,证券代码可输可不输入时使用

## 9、操作员帐号

操作员帐号（fdOperator），操作员帐号:6位的数字，只接受数字输入并根据营业部实现自动补齐功能

## 10、操作员密码

操作员密码（fdOperatorPassword），密码：0到8位的数字和字母。如果想实现一开始就有效的功能，可以将字段的DefaultValue设置为空串，type设置为string，注意不是Unassigned。

## 11、客户密码

客户密码（fdPassword），密码：1到8位的数字，只接受数字输入。

## 12、时间校验

时间校验（fdCheckTime），时间:hh:nn:ss，判断是否为时间输入。如果输入的时间为六位hhnnss自动补齐为8位

## 13、时间大小校验

时间大小校验（fdTimeReCheck），必须大于开始时间;时间:hh:nn:ss。其他同时间校验

## 14、数据值大小校验

数据值大小校验（fdCompData），必须大于起始数据,大于等于零的数。必须大于进行比较的工作字段的值，输入方式不限制。一定要在MonitorValueChangedFields属性中指定要进行比较的工作字段。

## 15、各种号码

包括‘客户号’，‘沪A账号’，‘深A账号’，‘沪B账号’，‘深B账号’，‘银行帐号’，‘牛卡账号’，‘客户号\*’，‘沪A账号\*’，‘深A账号\*’，‘沪B账号\*’，‘深B账号\*’，‘银行帐号\*’，‘牛卡账号\*’，‘经纪人号’，‘经纪人号\*’表示各类号码的输入规则。加‘\*’表示也可为空，不加‘\*’表示必输。

# 二十、工作字段表现形式

所有的工作字段表现形式在KSComCmdDefs文件内，可以在数据显示控件的DataPresentType属性中指定工作字段表现形式，也可以在WVDBBinder的WVDBBinding的DataPresentType属性中指定工作字段表现形式。相应的在DataPresentParam里面设置补充的参数。1

## 1、IDOBJECTS

IDOBJECTS提供了数据字典条目的转换，例如将01转换为人民币，可以通过使用IDOBJECTS和参数1101（1101是货币的数据字典条目）来完成。

IDOBJECTS通常可以用在如下三种情况

控件是WVLabel(标签)，可以将其DataPresentType设置为IDOBJECTS, DataPresentParam设置为相应的数据字典条目

控件是WVCombeBox (组合框)，可以将其DataPresentType设置为IDOBJECTS, ItemsDataEntry设置为相应的数据字典条目,记住一定要将Style设置为csDropDownList。如此可以实现将该条目的数据字典的所有内容填满组合框，如果要包含‘所有’，要将相应的数据字典条目的数字前加上‘\*’号。

控件是WVDBBinder（数据绑定控件），可以将其DataPresentType设置为IDOBJECTS, DataPresentParam设置为相应的数据字典条目，同时不要忘了在FieldName中加上DBGrid中的字段的名字。

在报表和其他地方要完成这样的功能可以使用公共函数来减少开销，公共函数见本文的公共函数部分

DataPresentParam应该设置为数据字典条目。

## 2、IDOBJECTS2

IDOBJECTS2同IDOBJECTS的功能基本相同，但是 IDOBJECTS2处理多个由‘，’逗号分隔开的数据字典项目的转换。如’01，02’在IDOBJECTS2和参数1101下转换为‘人民币，美元’。

IDOBJECTS2通常可以用在如下两种情况

控件是WVLabel(标签)，可以将其DataPresentType设置为IDOBJECTS2, DataPresentParam设置为相应的数据字典条目

控件是WVDBBinder（数据绑定控件），可以将其DataPresentType设置为IDOBJECTS2, DataPresentParam设置为相应的数据字典条目，同时不要忘了在FieldName中加上DBGrid中的字段的名字。

DataPresentParam应该设置为数据字典条目。

## 3、货币

货币可以实现将一个普通的浮点数或整数转换为一定格式的数字的字符串，可以提供DataPresentParam，使用方法同Delphi中的FormatFloat函数，请自行参看Delphi中的帮助了解如何设置，如果不提供DataPresentParam，转换的格式默认为'#,##0.00'，即每三位一个逗号并保留两位小数的格式

货币格式一般可以用于WVLabel(标签)和WVDBBinder（数据绑定控件）。

在报表和其他地方要完成这样的功能可以使用公共函数来减少开销，公共函数见本文的公共函数部分

DataPresentParam如果为空，使用缺省的格式；否则按照DataPresentParam指定的格式调用FormatFloat。

## 4、大写货币

大写货币可以实现将一个普通的浮点数或整数转换为一定格式的中文数字的字符串，一般只适用于使用WVLabel(标签)

在报表和其他地方要完成这样的功能可以使用公共函数来减少开销，公共函数见本文的公共函数部分

## 5、费率

费率的功能基本同货币相同，转换的格式默认为'#0.00000000''，8位小数。

在报表和其他地方要完成这样的功能可以使用公共函数来减少开销，公共函数见本文的公共函数部分

## 6、代理期限

代理期限的功能是为了当客户的代理人的代理有效日期小于当前日期是将代理有效日期用红色显示。

# 二十一、公共函数

## 1清单

### 1.1说明

**级别**

说明使用的频率。有些函数虽然在框架里面占据重要地位，但是只在框架列表调用，那么还是属于不常用的部分。

**常用** 在操作界面程序里面常常使用的调用。

**次常用** 在操作界面程序里面不常使用的调用。

**不常用** 在操作界面程序里面几乎不使用的调用。这些调用被内部框架使用。

**分类**

按照功能对函数进行基本分类。分类名称尽量简短。

**基本** 框架级别的“常用”公共函数。

**辅助** 和证券业务没有关系，是程序编码部分使用的公共函数。

**业务相关** 和证券业务相关的公共函数。这些函数不和操作员交互，或者产生明显输出（例如打印机）。

**系统信息** 获得当前执行系统的信息。

**数据字典** 和数据字典相关的所有操作。

**业务操作** 和特殊的证券业务相关的操作，例如双柜员复核、打印回单、确认等等。这些操作一般弹出对话框或者使用打印机。

**其他** 几乎不直接调用的其他所有未归类的函数。

### 1.2列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **级别** | **分类** | **函数** | **说明** | 所属单元 |
| 常用 | 基本 | procedure RegisterUIClassFactory(Factory : TUIClassFactory) | 注册界面类工厂 | KSFrameWorks |
| 不常用 | 其他 | procedure UnRegisterUIClassFactory(Factory : TUIClassFactory) | 注销界面类工厂 | KSFrameWorks |
| 不常用 | 其他 | function FindUIClassFactory(const AUICode : TUICode): TUIClassFactory | 查找界面类工厂 | KSFrameWorks |
| 不常用 | 其他 | function CreateUserInterface(const AUICode : TUICode) : TUIInstanceRecord; | 创建界面实例记录 | KSFrameWorks |
|  |  |  |  |  |
| 常用 | 基本 | function Context : TKSContext; | 返回当前的上下文对象（代表了运行环境） | KSFrameWorks |
| 不常用 | 其他 | procedure SetContext(AContext : TKSContext); | 设置上下文对象(只能设置一次) | KSFrameWorks |
|  |  |  |  |  |
| 不常用 | 基本 | procedure InstallSubSystem(SubSystem : IDebugSubSystem); | 安装子系统 | KSFrameWorks |
| 不常用 | 基本 | procedure UnInstallSubSystem(SubSystem : IDebugSubSystem); | 注销子系统 | KSFrameWorks |
|  |  |  |  |  |
| 常用 | 系统信息.营业部 | function IsCenterDepartment : Boolean; | 判断当前营业部是否是总部 | KSFrameWorkUtils |
| 常用 | 系统信息.营业部 | function CurrentDepartment : TIdObject; | 返回当前营业部（作为TIdObject对象） | KSFrameWorkUtils |
| 常用 | 系统信息.营业部 | function CurrentDepartmentNo : string; | 返回当前营业部代码 | KSFrameWorkUtils |
|  |  |  |  |  |
| 常用 | 数据字典.字符串列表 | procedure InitIdObjectItems(IdObjectItems : TStrings; const DataEntry : TDataEntry;AddEmptyIdObject : Boolean = False; Filtered : Boolean = True; const DepartmentNo : string=''); | 将数据字典增加到字符串列表里面。字符串列表的Objects包含对应的对象 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure InitDepartmentItems(DepartmentList : TStrings; AddAllDepart : Boolean = False); | 将营业部增加到字符串列表里面 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure InitOnlyDepartmentItems(DepartmentList : TStrings; WorkField : TWVField = nil ;AddAllDepart : Boolean = False); | 将营业部增加到字符串列表里面,不添加总部,同时支持字段的默认值 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure InitDepartCustomerTypes(const DepartmentNo : string; CustomerTypes : TStrings; AddAll : Boolean = False); | 将营业部客户类别填充到字符串列表里面 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure InitCustomerGroups(const DepartmentNo : string; CustomerGroups : TStrings; AddAll : Boolean = False); | 将营业部客户组别填充到字符串列表里面 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure AddOnlyEmptyItem(IdObjectItems : TStrings); | 将代表所有或者空项目的IDObject加入到字符串列表里面，使得列表仅仅包含一个项目 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure InitBusinessFlagItems(BusinessFlagItems : TStrings; const AFrom,ATo : string); overload; | 根据业务代码初始化字符串列表 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure InitBusinessFlagItems(BusinessFlagItems : TStrings; const ACategories : string=''); overload; | 根据业务代码初始化字符串列表 | KSFrameWorkUtils |
| 常用 | 数据字典.字符串列表 | procedure GetObjectsItems(Items : TStrings; Objects : TObjectList); | 据对象列表生成字符串列表。对象列表包含的对象是TKSAbstractObject对象 | KSCustomers |
|  |  |  |  |  |
| 次常用 | 辅助.文件名 | function GetFileName(DirKey : Integer; const BasedFileName : string) : string; | 根据目录类型和文件名获得文件名全路径 | KSFrameWorkUtils |
| 次常用 | 辅助.文件名 | function GetReportFileName(const BasedFileName : string) : string; | 获得报表文件全路径名 | KSFrameWorkUtils |
|  |  |  |  |  |
| 常用 | 系统信息.日期 | function GetToday : string; | 获得当前系统日期(yyyymmdd) | KSFrameWorkUtils |
| 常用 | 系统信息.日期 | function GetTodayStr : string; | 获得当前系统日期的字符串表达，格式由FullDateFormat确定 | KSFrameWorkUtils |
| 次常用 | 辅助.日期 | function DBStrDateToDateTime(const DateStr : string) : TDateTime; | 将数据库的字符串日期格式(yyyymmdd)转换为日期类型 | KSFrameWorkUtils |
| 次常用 | 辅助.日期 | function FormatDBStrDate(const DateFormat,DateStr : string) : string; | 将数据库的字符串日期格式(yyyymmdd)格式化为指定的字符串格式 | KSFrameWorkUtils |
|  |  |  |  |  |
| 次常用 | 其他 | procedure DoLongOperation(const WaitMsg : string; DoSomeThing : TDoSomeThing); | 进行一个长时间的操作。将会显示一个等待处理的对话框。处理完毕以后关闭该对话框 | KSFrameWorkUtils |
| 次常用 | 其他 | procedure DoLongDatasetOperation(const WaitMsg : string; Dataset : TDataset; DoSomeThing : TDoSomeThingForDataset); | 进行一个长时间的遍历数据集的操作。将会显示一个等待处理的对话框 | KSFrameWorkUtils |
|  |  |  |  |  |
| 常用 | 数据字典 | function FindIdObject(const DataEntry : TDataEntry; const ID : string; const Department : string='') : TIdObject; | 通过DataEntry（数据字典条目）和ID (TidObject的ID标示符)来获得相应的TidObject对象如果TidObject对象不存在，返回nil | KSDataDictionary |
| 常用 | 数据字典 | function GetIdObjectName(const DataEntry : TDataEntry; const ID : string; const Department : string='') : string; | 通过DataEntry（数据字典条目）和ID (TidObject的ID标示符)来获得相应的TidObject对象的Name如果TidObject对象不存在，返回空串 | KSDataDictionary |
| 不常用 | 数据字典 | function GetIdObjectList(const DataEntry : TDataEntry; const Department : string='') : TIdObjectList; | 通过DataEntry（数据字典条目）来获得相应的TIdObjectList列表对象。如果不存在，返回为nil。 | KSDataDictionary |
| 常用 | 数据字典 | function NeedIdObjectList(const DataEntry : TDataEntry; const Department : string='') : TIdObjectList; | 通过DataEntry（数据字典条目）来获得相应的TIdObjectList列表对象。如果不存在，会使用系统定义的方法来生成TIdObjectList列表对象。 | KSDataDictionary |
| 不常用 | 数据字典 | function DataEntryToCategory(const DataEntry : TDataEntry; const Department : string) : string; | 将DataEntry和Department组合获得Category的名字 | KSDataDictionary |
| 常用 | 数据字典 | function GetFirstItemID(const DataEntry : TDataEntry; const Department : string=''; Filtered : Boolean = True) : string; | 通过DataEntry（数据字典条目）来获得相应的TIdObjectList列表对象中的第一个TidObject对象的ID，如果不存在返回空串 | KSDataDictionary |
| 不常用 | 数据字典 | procedure FreeIdObjects(const DataEntry : TDataEntry; const Department : string=''); | 释放指定的TidObjectList对象 | KSDataDictionary |
| 不常用 | 数据字典 | procedure GetCustTypeText(Sender: TField; const DepartFieldName : string; var Text: string); | 将数据集中营业部客户类别字段转换为文字 | KSDataDictionary |
| 不常用 | 数据字典 | procedure GetCustGroupText(Sender: TField; const DepartFieldName : string; var Text: string); | 将数据集中营业部客户组别字段转换为文字 | KSDataDictionary |
|  |  |  |  |  |
| 不常用 | 使用数据集 | function SetupGrid(Grid: TDBGrid; ColColor: TColor):Boolean; | 弹出调整Grid对话框进行调整 | DataUsageUtils |
| 不常用 | 使用数据集 | procedure ReadGridInfo(Grid: TDBGrid; ColColor: TColor; const FileName : string); | 读取保存在文件中的Grid中的信息 | DataUsageUtils |
| 不常用 | 使用数据集 | procedure SaveGridInfo(Grid: TDBGrid; const FileName : string); | 将Grid中的信息保存到文件中 | DataUsageUtils |
| 不常用 | 使用数据集 | function IsSearchDataset(DataSet: TDataSet; var SearchOptions : TDataSearchOptions):Boolean; overload; | 弹出从Grid中搜索数据的对话框 | DataUsageUtils |
| 不常用 | 使用数据集 | function IsSearchDataset(Grid : TDBGrid; var SearchOptions : TDataSearchOptions):Boolean; overload; | 弹出从Grid中搜索数据的对话框 | DataUsageUtils |
| 不常用 | 使用数据集 | function SearchDataset(DataSet: TDataSet; var SearchOptions : TDataSearchOptions):Boolean; | 从数据集中搜索数据 | DataUsageUtils |
| 不常用 | 使用数据集 | procedure DbGridPrint(DBGrid : TDBGrid; const RptTitle : string); | 打印DbGrid中的数据 | DataUsageUtils |
| 不常用 | 使用数据集 | procedure DrawCellForSysParams(Grid : TDBGrid;const Rect: TRect;DataCol: Integer;Column: TColumn;State: TGridDrawState;const DepartFieldName : string); | 提供对多个营业部的数据进行查询，将总部数据显示为灰色的功能 | DataUsageUtils |
| 不常用 | 使用数据集 | procedure ShowDetailForGrid(Grid : TDBGrid); | 显示Grid中数据的详细信息 | KSFrameWorkUtils |
|  |  |  |  |  |
| 常用 | 辅助 | procedure ReinputCheck(WorkField: TWVField); | 对WorkField进行重入校验，注意WorkField.MonitorValueChangedFields里面包含第一次输入的字段的名称 | KSFrameWorkUtils |
| 常用 | 辅助 | procedure AddRecToListview(listview:Tlistview; const Rec:Tstrings;Const MaxRecCount:integer=3); | 存放当前输入记录到listview | KSFrameWorkUtils |
| 常用 | 辅助 | function GetPrefix(const Ftext:String;const SeperateChar:char='-'):string; | 获取前缀.如获取'01-人民币'的前缀为'01' | KSFrameWorkUtils |
| 常用 | 辅助 | function DelPrefix(const Ftext:String;const SeperateChar:char='-'):string; | 滤去前缀。如将'01-人民币'滤去前缀后为'人民币' | KSFrameWorkUtils |
| 常用 | 辅助 | procedure SelectAll(CheckListBox:TCheckListBox;SelectAll:Boolean); | 控制TCheckListBox对象的全部选中或全部不选中 | KSFrameWorkUtils |
| 不常用 | 辅助 | procedure MakeSelectAllToNoSelected(FieldData, ParamData: TKSDataObject; CheckListBox : TWVCheckListBox); | 将全选转变为全不选，进行优化处理 | KSFrameWorkUtils |
| 次常用 | 业务操作 | function InputText(const ACaption, APrompt: string; HideText,CustomerPassword : Boolean; var Value: string): Boolean; | 输入内容，支持密码键盘用 | KSFrameWorkUtils |
| 次常用 | 业务操作 | function InputCustomerPasswordForField(Field : TWVField) : Boolean; | 输入客户密码，支持密码键盘用 | KSFrameWorkUtils |
| 常用 | 业务操作 | procedure PrintReceipt(const BasedFileName : string; WorkView : TWorkView; AddDefaultSystemParams : Boolean=True); | 打印回单 | DataUsageUtils |
| 常用 | 业务操作 | function DoPrintReceipt(const AOperation : string; const BasedFileName : string; WorkView : TWorkView; AddDefaultSystemParams : Boolean=True):Boolean; | 弹出对话框询问是否需要打印回单 | DataUsageUtils |
| 常用 | 业务操作 | function DoPrompt(const AOperation, APrompt : string) : Boolean; | 弹出确认对话框进行确认 | DataUsageUtils |
| 常用 | 业务相关 | function FormatAndValidAccount(MethodChar : Char; var Account : string; CanEmpty : Boolean=False): Boolean; | 实现各种帐好的补齐和校验功能 | KSEdits |
|  |  |  |  |  |
| 次常用 | 基本 | procedure RegisterModule(Module : TComponent); | 注册Module包含的特殊组件到框架中 | KSFrameWorkUtils |
| 常用 | 基本 | procedure CheckCommandReturnValue(Command: TWVCommand; ShowWarning : Boolean=True); | 检查命令对象返回的数据是否有效。如果无效，抛出意外 | KSFrameWorkUtils |
| 不常用 | 其他 | procedure FreeDataSetForCommand(Command: TWVCommand); | 释放命令对象里面包含的结果集对象 | KSFrameWorkUtils |
| 次常用 | 其他 | procedure ResetOtherInputBox(Owner : TComponent); | 将特殊的输入控件复位（清空）。目前主要处理客户号输入控件 | KSFrameWorkUtils |
| 不常用 | 其他 | function FindAssociatedWorkView(Control : TControl) : TWorkView; | 查找与指定控件相关的工作视图 | KSFrameWorkUtils |
| 常用 | 系统信息 | procedure AddSystemParams(Params : TStrings); | 将系统变量（营业部代码、名称；职工代码、名称）增加到字符串列表里面 | KSFrameWorkUtils |
| 不常用 | 其他 | procedure UpdateControlStyle(Control : TControl); | 根据系统配置，修改控件的外观（字体、颜色等等） | KSFrameWorkUtils |
| 不常用 | 其他 | procedure HookControl(Control : TControl); | 设置控件的事件处理句柄 | KSFrameWorkUtils |
| 不常用 | 其他 | procedure MakeControlReadOnly(Control : TControl; AReadOnly : Boolean); | 将控件设置为只读 | KSFrameWorkUtils |
| 不常用 | 其他 | procedure MakeChildrenReadOnly(AParent : TWinControl; AReadOnly : Boolean); | 将子控件设置为只读 | KSFrameWorkUtils |
| 不常用 | 其他 | function GetPrinterInfoToLabel(ALabel : TLabel) : Boolean; | 显示打印机信息 | DataUsageUtils |
|  |  |  |  |  |
| 常用 | 业务操作 | function DoSingleStationCheck(const AOperation : string;var CheckerNo, CheckerPassword : string;FuncNo : Integer=0;const CustomerNo : string = '';const CurrentType : string = '') : Boolean; | 进行单机双柜员复核,通过参数返回复核操作员的工号和密码，通过返回值返回操作员的工号和密码是否合法 | KSComCmdDefs |
| 次常用 | 辅助 | procedure CompDataCheck(WorkField: TWVField; IsZeroEqueltoAll: Boolean; IsBig : Boolean=True; RetouchNum: Double=0); | 比较数据值的大小 | KSComCmdDefs |
| 次常用 | 辅助 | function CheckDateValid(const DateString : string; Needed: Boolean) : Boolean; | 检验日期的有效性 | KSComCmdDefs |
| 次常用 | 辅助 | function CheckTimeValid(const TimeString: string): Boolean; | 检验时间的有效性 | KSComCmdDefs |
| 次常用 | 业务相关 | function CheckExists(AccountType : Char; CertType : Integer; const BankCode : string; const Account : string): Boolean; | 检查账号/客户/证件是否存在。返回true存在,false不存在 | KSComCmdDefs |
| 常用 | 业务相关 | function GetCustomer(const BranchCode, CustomerNo: string): TKSAbstractCustomerInfo; | 根据营业部、客户号获取客户信息 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetCustomerInfo(const BranchCode, CustomerNo: string; CustomerInfos: TObjectList); | 根据营业部、客户号获取客户信息 | KSComCmdDefs |
| 常用 | 业务相关 | function GetBroker(const BranchCode, BrokerNo: string): TKSAbstractBrokerInfo; | 根据营业部、经纪人号获取经纪人信息 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetBrokerInfo(const BranchCode, BrokerNo: string; BrokerInfos: TObjectList); | 根据营业部、经纪人号获取经纪人信息 | KSComCmdDefs |
| 次常用 | 业务相关 | function GetSecAccount(const Market, Account : string) : TKSAbstractSecAccount; | 根据市场、股东代码获取股东信息 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetSecAccounts(const CustomerNo : string; const Market : string; OnlyMainAccounts : Boolean; SecAccounts : TObjectList); | 根据客户号获取股东代码 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetMoneyAccounts(Customer : TKSAbstractCustomer; const CurrencyType : string; MoneyAccounts : TObjectList); | 根据客户号获取资金账号信息 | KSComCmdDefs |
| 常用 | 业务相关 | procedure GetBankAccounts(const CustomerNo, Bank , CurrencyType : string; BankAccounts: TObjectList; QryType :Integer=0); | 根据客户号获取银行账号 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetSecInfo(const SecCode, Market: string; SecInfos: TObjectList); | 根据证券代码和市场代码获取证券信息 | KSComCmdDefs |
| 常用 | 业务相关 | procedure GetSeatInfo(const Market,BranchCode,SeatNo : string;SeatInfos: TObjectList; UniqueFlag:Integer=1 ); | 根据市场代码,查询营业部和席位代码获取席位信息 | KSComCmdDefs |
| 常用 | 业务相关 | function GetSeat(const Market: string;const BranchCode : string='';UniqueFlag:Integer=1): TKSAbstractSeatInfo; | 根据市场代码,查询营业部和席位代码获取席位信息 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetEmpInfo(const BranchCode, EmpNo: string; EmpInfos: TObjectList); | 根据查询营业部和职工代码获取职工信息 | KSComCmdDefs |
| 常用 | 业务相关 | function GetEmp(const EmpNo: string; const BranchCode: string =''): TKSAbstractEmpInfo; | 根据查询营业部和职工代码获取职工信息 | KSComCmdDefs |
| 常用 | 业务相关 | function GetReportDay : string; | 获得可以打印报表的最后一个工作日 | KSComCmdDefs |
| 常用 | 业务相关 | procedure GetBranch2BankInfo(const BranchCode, CurrencyType, BankCode: string; BankInfos: TObjectList); | 根据查询营业部,货币代码和银行代码获取营业部对应的银行信息 | KSComCmdDefs |
| 次常用 | 业务相关 | function GetStationAuth(const BranchCode: string ='';const Station: string =''): TKSAbstractStationAuthInfo; | 根据查询营业部和岗位代码获取岗位权限信息 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure GetStationAuthInfo(const BranchCode,Station: string; StationInfos: TObjectList); | 根据查询营业部和岗位代码获取岗位权限信息 | KSComCmdDefs |
| 次常用 | 业务相关 | procedure CheckAccountFieldValid(WorkField : TWVField; MethodChar : Char; CanEmpty : Boolean=False); | 检查帐号的有效性 | KSComCmdDefs |
| 次常用 | 业务相关 | function NeedMarkets : TObjectList; | 获得所有的市场信息 | KSComCmdDefs |
| 常用 | 业务相关 | function GetMarket(const Market : string) : TKSAbstractMarket; | 获得单一市场信息 | KSComCmdDefs |
| 次常用 | 业务相关 | function FindStock(const SecCode,Market : string) : TKSAbstractSecInfo; | 获得证券信息 | KSComCmdDefs |
| 次常用 | 业务相关 | function FormatAndValidSecurityCode(const Market : string; var SecurityCode : string) : Boolean; | 根据市场,检验证券代码的有效性并补齐 | KSComCmdDefs |
| 次常用 | 业务相关 | function FormatAndValidBrokerNo(var BrokerNo : string) : Boolean; | 根据营业部代码,检验经纪人号的有效性并补齐 | KSComCmdDefs |
| 常用 | 业务相关 | procedure QueryControl(const QueryMethod: String; CustField, DeptField: TWVField;CustNo: TKSAccountEdit; BranchCode: TWVComboBox); | 在特定查询方式下，相关项目的显示控制（用于查询界面） | KSComCmdDefs |
| 常用 | 业务相关 | function GetBranchParam(ParamCode: string;const BranchCode:string=''): string; | 获取营业部参数的值 | KSComCmdDefs |

## 2、获得客户、股东等信息的函数说明

## 3、其他常用函数说明

# 二十二、其他

## 1、对密码小键盘的支持

|  |  |  |
| --- | --- | --- |
| **属于** | **单元** | **增加的支持** |
| 框架 | KSFrameWorks | 在IDialogSystem.InputText提供对字符串（包括密码）输入对话框的支持。 |
| 框架 | KSFrameWorkUtils | 函数InputText和InputPasswordForField对IDialogSystem.InputText的使用进行了简化。 |
| 框架 | KSClientConsts | 定义了密码的最大长度和配置文件与密码小键盘修改的键值。 |
| 框架 | KSEdits | 对象类TPasswordInputMethod和全局对象PasswordInputMethod提供了处理密码小键盘的接口。 |
| 主程序 | UInputTextDlg | 如果要求输入客户密码，同时设置了全局对象PasswordInputMethod，那么使用全局对象PasswordInputMethod提供的密码输入方法。 |
| 主程序 | UContext | 实现IDialogSystem |
| 主程序 | UCommData | 支持客户号输入控件。同时在输入客户密码的时候调用UInputTextDlg里面的对话框。 |
| 驱动小键盘的程序包 | UPwdKeyboard | 驱动密码小键盘的接口函数 |
| 驱动小键盘的程序包 | UPwdInp | TPasswordInputMethod的实例，调用密码小键盘接口函数，完成密码输入。 |

# 附录、参考文档

编码规范：“Delphi编码规范草案.doc”。

公共程序：“公共程序清单.xls”和“公共程序的使用方法.doc”。

其他：

“工作检查表.xls”、

“集中交易前台程序通用功能列表.doc”。

“集中交易柜台程序使用手册.doc”。

“集中交易柜台程序维护手册.doc”。