IDynamicFormModel接口调用示例代码

修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No** | **日期** | **编制\修订** | **校对** | **批准** | **修改的章节号** |
| V1.0 | 20150327 | 丁振华 | 吴亮 | 赖碧云 | 初始版本 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

问题和意见

如果你对文档有任何意见、问题或想法，或者你的问题未在此文档中找到答案，[请通过电子邮件联系我们:](mailto:请通过电子邮件联系我们:%20liang_wu@kingdee.com)  [liang\_wu@kingdee.com](mailto:liang_wu@kingdee.com)

# IDynamicFormModel接口概要说明

IDynamicFormModel接口，对动态表单的数据模型进行管理，广泛应用于插件、操作、表单服务等需要存取表单数据的模块；

表单维护插件(AbstractDynamicFormPlugIn)，属性Model指向IDynamicFormModel的接口实例；

# 公共属性

## BillBusinessInfo

当前界面所管理的单据元数据对象。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| BusinessInfo BillBusinessInfo { get; } | |

**备注**

对于单据维护界面，此属性就是本单的元数据；而对于列表界面来说，此属性是列表界面展示的单据元数据。

### 常用属性

|  |  |
| --- | --- |
| 名称 | 描述 |
| Elements | 返回单据上的全部元素，包括全部字段、实体 |
| Entrys | 返回单据上的全部实体 |
| MainOrgField | 返回单据的主业务组织字段 |

### 常用方法

|  |  |
| --- | --- |
| 名称 | 描述 |
| GetBillNoField() | 获取单据编号字段，如果是基础资料，则返回基础资料编码 |
| GetBillTypeField() | 获取单据类型字段 |
| GetBosFields() | 获取单据全部字段 |
| GetDynamicObjectType([bool *forceRebuild* = false]) | 获取单据的ORM数据模型对象，  参数决定是否重新构建ORM数据对象模型。单据元数据改变后，需要强制要求重新构建ORM数据对象模型。 |
| GetEntity(string *key*) | 获取单据的实体元数据 |
| GetField(string *key*) | 获取单据字段元数据 |
| GetFieldList() | 获取单据全部字段元数据列表 |
| GetForm() | 获取单据整体属性元数据 |
| GetRealField(string *key*) | 获取代理字段的真实字段元数据 |
| GetRedBlueField() | 获取红蓝单标志字段 |
| GetSubBusinessInfo(System.Collections.Generic.List<string> *sectionKeys*) | 根据所选字段，产生节选单据元数据 |

### 案例 – 供应链单据列表，默认按照日期、单据编号排序

**代码来源**

Kingdee.K3.SCM.Business.PlugIn.SCMBillList

**需求背景**

供应链单据，显示列表时，如果用户未明确指定排序字段，则列表将按照创建日期、单据编号、分录序号排序；

**实现方案**

设计一个公用列表插件，捕获PrepareFilterParameter事件，如判断出用户未设置排序字段，则到单据元数据中，尝试取单据的创建日期、单据编号、分录序号等信息，以这些字段作为排序字段。

注意，因为这是一个公用插件，需要用到BillBusinessInfo属性，动态的取创建日期、单据编号、分录序号等字段，不能直接写死字段Key，否则可能会引发字段不存在的错误。

**关键字**

单据元数据 界面元数据 去指定类型字段 取创建日期字段 取单据编号字段 列表排序

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Core.List.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata.FormElement;  using Kingdee.BOS.Core.Metadata.BarElement;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.List.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.K3.Core;  using Kingdee.K3.Core.SCM;  using Kingdee.BOS.Core.Bill;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.CommonFilter;  using Kingdee.BOS.Core.Enums;  namespace Kingdee.K3.SCM.Business.PlugIn  {  /// <summary>  /// 供应链单据模板列表公共插件  /// </summary>  [Description("供应链单据列表公共插件")]  public class SCMBillList : AbstractListPlugIn  {  /// <summary>  /// 列表取数前，修改过滤条件、排序字段  /// </summary>  /// <param name="e"></param>  public override void PrepareFilterParameter(FilterArgs e)  {  base.PrepareFilterParameter(e);  if (string.IsNullOrWhiteSpace(e.SortString))  {  // 创建日期  Field crDateFld = this.Model.**BillBusinessInfo**.GetFieldList().FirstOrDefault(  p => p is CreateDateField);  // 单据编号  Field numberFld = this.Model.**BillBusinessInfo**.GetBillNoField();  // 分录序号  StringBuilder entrySort = new StringBuilder();  foreach (FilterEntity ety in e.SelectedEntities)  {  if (ety.Selected && ety.EntityType == BOSEnums.Enum\_EntityType.Entity ||  ety.Selected && ety.EntityType == BOSEnums.Enum\_EntityType.SubEntity)  {  EntryEntity entryEntity = this.Model.**BillBusinessInfo**.GetEntryEntity(ety.Key);  if (string.IsNullOrWhiteSpace(entryEntity.SeqFieldKey))  {  entrySort.AppendFormat(", {0}.{1} ASC ",  entryEntity.TableAlias, entryEntity.EntryPkFieldName);  }  else  {  entrySort.AppendFormat(", {0}.{1} ASC ",  entryEntity.TableAlias, entryEntity.SeqFieldKey);  }  }  }    // 设置排序：创建日期、日期、分录序号  if (crDateFld != null)  {  if (numberFld != null)  {  e.SortString = string.Format(" {0} DESC,{1} DESC {2} ",  crDateFld.Key, numberFld.Key, entrySort.ToString());  }  else  {  e.SortString = string.Format(" {0} DESC {1} ",  crDateFld.Key, entrySort.ToString());  }  }  }  }  }  } | |

## BusinessInfo

当前界面对应的元数据对象；

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| BusinessInfo BusinessInfo { get; } | |

**备注**

与BillBusinessInfo属性相反，本属性为当前界面的元数据对象，而BillBusinessInfo为当前界面所管理的单据的元数据对象；

如列表界面，BusinessInfo为列表本身的元数据对象，而BillBusinessInfo，则为列表界面所展示的单据元数据对象

常用属性、方法，同BillBusinessInfo。

### 案例 – 信用重算过滤界面，列出可选单据

**代码来源**

Kingdee.K3.SCM.Credit.Business.PlugIn.CreditRecalFilterEdit

**需求背景**

信用重算列表过滤条件界面，提供一个下拉列表，填入可选单据，供用户选择不同单据进行信用重算；

**实现方案**

界面加载完毕后，取得界面上的下拉列表控件，添加可选项；

**关键字**

下拉列表可选项 字段是否存在判断

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.Metadata;  namespace Kingdee.K3.SCM.Credit.Business.PlugIn  {  public class CreditRecalFilterEdit : AbstractDynamicFormPlugIn  {  private bool IsCustChecked;  private bool IsGroupCustChecked;  private bool IsOrgChecked;  private bool IsDeptChecked;  private bool IsSalerChecked;  private bool IsSaleGroupChecked;  public override void AfterBindData(EventArgs e)  {  SetObjTypeSelectedItem();  this.View.GetControl("FRefreshTrack").Enabled = false;  }  private void SetObjTypeSelectedItem()  {  ComboField comfield = this.Model.**BusinessInfo**.GetElement("FObjectType") as ComboField;  if (comfield == null)  {  return;  }  ComboFieldEditor combox = this.View.GetFieldEditor("FObjectType", 0) as ComboFieldEditor;  List<EnumItem> listSubItems = new List<EnumItem>();  EnumItem blankitem = new EnumItem();  blankitem.Caption = new LocaleValue("", Context.CurrentThread.CurrentCulture.LCID);  blankitem.EnumId = "";  blankitem.Seq = 0;  blankitem.Value = "";  listSubItems.Add(blankitem);  if (IsCustChecked == true)  {  EnumItem item = new EnumItem();  item.Caption = new LocaleValue("客户", Context.CurrentThread.CurrentCulture.LCID);  item.EnumId = "";  item.Seq = 0;  item.Value = "BD\_Customer";  listSubItems.Add(item);  }  if (IsGroupCustChecked == true)  {  EnumItem item = new EnumItem();  item.Caption = new LocaleValue("集团客户", Context.CurrentThread.CurrentCulture.LCID);  item.EnumId = "";  item.Seq = 1;  item.Value = "BD\_GroupCustomer";  listSubItems.Add(item);  }  if (IsSalerChecked == true)  {  EnumItem item = new EnumItem();  item.Caption = new LocaleValue("销售员", Context.CurrentThread.CurrentCulture.LCID);  item.EnumId = "";  item.Seq = 2;  item.Value = "BD\_Saler";  listSubItems.Add(item);  }  if (IsSaleGroupChecked == true)  {  EnumItem item = new EnumItem();  item.Caption = new LocaleValue("销售组", Context.CurrentThread.CurrentCulture.LCID);  item.EnumId = "";  item.Seq = 3;  item.Value = "BD\_SaleGroup";  listSubItems.Add(item);  }  if (IsDeptChecked == true)  {  EnumItem item = new EnumItem();  item.Caption = new LocaleValue("销售部门", Context.CurrentThread.CurrentCulture.LCID);  item.EnumId = "";  item.Seq = 4;  item.Value = "BD\_Department";  listSubItems.Add(item);  }  if (IsOrgChecked == true)  {  EnumItem item = new EnumItem();  item.Caption = new LocaleValue("销售组织", Context.CurrentThread.CurrentCulture.LCID);  item.EnumId = "";  item.Seq = 5;  item.Value = "ORG\_Organizations";  listSubItems.Add(item);  }  combox.SetComboItems(listSubItems);  }  }  } | |

## Context

上下文对象。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| Context Context { get; } | |

**备注**

本对象记录了与数据中心的连接，在调用各种K/3 Cloud提供的服务时，均需要传入本对象；

另外，还记录了当前用户、语言等信息。

常用作服务参数；

### 常用属性

|  |  |
| --- | --- |
| 名称 | 描述 |
| ClientType | 客户端类型，如SL客户端，WPF客户端 |
| ComputerName | 客户端机器名 |
| ContextId | 上下文Id，唯一标识 |
| CurrentOrganizationInfo | 当前组织信息 |
| DatabaseType | 数据库类型，如SQL Server、Oracle |
| DataCenterName | 数据中心名称 |
| IpAddress | 客户端机器IP地址 |
| IsMultiOrg | 是否启用了多组织 |
| LoginName | 登录用户名 |
| UserId | 登录用户内码 |
| UserName | 登录用户名 |

### 案例 – 销售员列表新增，验证用户权限

**代码来源**

Kingdee.K3.SCM.Business.PlugIn.OperatorF8List

**需求背景**

在”销售员”、”仓管员”列表上，用户点击新增时，需要验证用户对”操作员”的新增权限，如果无权，禁止用户新增；

此需求的难点，需要在一种单据(A)列表上，验证另外一种单据(B)的权限，无法通过BOS设计器的操作定义，配置权限项，必须使用插件实现；

**实现方案**

捕获销售员、仓管员列表的菜单单击时间，如果点击的菜单是”新增”菜单，则调用验权服务，进行验权，如果验权不通过，给出提示，禁止操作；否则，调出操作员新增界面；

对于默认的新增操作，则直接取消，不再显示销售员的新增界面；

**关键字**

调用验权服务 验权 显示单据新增界面 取消菜单操作

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Core.Bill;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.List.PlugIn;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Permission;  using Kingdee.BOS.ServiceHelper;  using Kingdee.K3.Core.SCM;  namespace Kingdee.K3.SCM.Business.PlugIn  {  [Description("销售员，仓管员F8列表上的特殊处理")]  public class OperatorF8List : AbstractListPlugIn  {  public override void BarItemClick(BarItemClickEventArgs e)  {  base.BarItemClick(e);  switch (e.BarItemKey.ToUpperInvariant())  {  case "TBNEW":  case "TBSPLITNEW":  e.Cancel = true;  PermissionAuthResult perAuthResult = PermissionServiceHelper.FuncPermissionAuth(  this.Model.**Context**,  new BusinessObject() { Id = SCMFormIdConst.BD\_Operator },  PermissionConst.New);  if (perAuthResult.Passed)  {  BillShowParameter showParameter = new BillShowParameter();  showParameter.FormId = SCMFormIdConst.BD\_Operator;  showParameter.ParentPageId = this.View.PageId;  showParameter.Status = OperationStatus.ADDNEW;  this.View.ShowForm(showParameter);  }  else  {  this.View.ShowMessage("您没有业务员的新增权限!");  }  break;  default:  break;  }  }  public override void BeforeDoOperation(BeforeDoOperationEventArgs e)  {  base.BeforeDoOperation(e);  switch (e.Operation.FormOperation.Operation.ToUpperInvariant())  {  case "NEW":  e.Cancel = true;  break;  default:  break;  }  }  }  } | |

## DataChanged

界面上的数据包，是否被改动。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| bool DataChanged { get; set; } | |

**备注**

动态表单加载完毕，数据绑定到界面上之后，此属性被重置为false；此后，如果用户修改了界面上字段值，此属性更改为true；用户保存单据后，此属性会被重置为false。

因此，插件可以通过此属性，了解数据在上次保存之后，是否有改动；

### 案例 – 界面关闭前，检查改动，提醒保存

**代码来源**

Kingdee.K3.SCM.DRP.Business.PlugIn.DRPParameterEdit

**需求背景**

DRP系统参数配置界面，关闭界面前，检查用户是否修改了参数配置，但是未保存；如果有改动，则提醒用户保存。

**实现方案**

捕捉界面的BeforeClosed事件，在界面关闭前，进行修改检查；

**关键字**

系统参数 界面关闭事件 数据改动检查

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.K3.SCM.ServiceHelper;  using Kingdee.K3.SCM.DRP.Core;  namespace Kingdee.K3.SCM.DRP.Business.PlugIn  {  [Description("DRP系统参数")]  public class DRPParameterEdit : AbstractDynamicFormPlugIn  {  // 无关逻辑，略去代码  public override void BeforeClosed(BeforeClosedEventArgs e)  {  if (this.Model.**DataChanged** && \_needNotifyBeforeClose)  {  e.Cancel = true;  string msg = "内容已经修改，是否保存？";    // 显示提示信息，本在用户确认后回调BeforeClosedCallBack函数  this.View.ShowMessage(  msg,  MessageBoxOptions.YesNoCancel,  BeforeClosedCallBack);  }  }  /// <summary>  /// 退出前，提示信息后的回调  /// </summary>  /// <param name="result"></param>  private void BeforeClosedCallBack(MessageBoxResult result)  {  if (result == MessageBoxResult.Cancel) return;  if (result == MessageBoxResult.No)  {  \_needNotifyBeforeClose = false;  this.View.Close();  return;  }  if (result == MessageBoxResult.Yes)  {  // 用户选择了需要保存  bool success = SaveParameter();  if (success)  {  \_needNotifyBeforeClose = false;  this.View.Close();  return;  }  }  }  /// <summary>  /// 保存参数  /// </summary>  /// <returns></returns>  private bool SaveParameter()  {  // 无关逻辑，略去代码  // 重置数据改动标志为false  this.Model.**DataChanged** = !saveResult && this.Model.DataChanged;  return saveResult;  }  }  } | |

## DataObject

界面的后台数据包对象。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicObject DataObject { get; set; } | |

**备注**

通常来说，插件可以通过Model对象提供的方法，来读取、更新字段值，并触发字段值改变事件，无需直接访问此属性。

但是某些特殊场景，需要直接访问此属性，获取、更新字段值。

特别注意：通过访问DataObject对象来更新字段值，不会触发字段的值改变事件；如通过DataObject修改数量，不会自动计算金额；

### 案例 – 把子界面上的内容填入到本界面

**代码来源**

Kingdee.K3.SCM.DRP.Business.PlugIn.DRPElementEdit

**需求背景**

要补货方案要素界面，数据保存前，需要把公式配置子界面上，定义的公式、公式描述，填写到本界面的数据包，保存入库。

本需求的难点，在于公式编辑界面，是一个独立子界面，用户在其上编辑的公式，需要通过插件来取值，转存到本界面公式字段上，而且在填写过程中，不希望触发界面的值改变事件。

**实现方案**

捕获菜单单击事件BarItemClick，在保存前，取公式编辑子界面上的公式，格式化后填写到数据包；

**关键字**

嵌套界面 显示子界面在面板中 取子界面的值 直接到数据包修改字段值

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Formula;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  namespace Kingdee.K3.SCM.DRP.Business.PlugIn  {  [Description("要补货方案要素表单插件")]  public class DRPElementEdit : AbstractBillPlugIn  {  //子页面ID  private string subPageId = "";  public override void AfterBindData(EventArgs e)  {  base.AfterBindData(e);  // 在面板中，嵌入显示公式编辑子页面  ShowExpressionEditor();  }  public override void BarItemClick(BarItemClickEventArgs e)  {  switch (e.BarItemKey.ToUpperInvariant())  {  case "TBSPLITSAVE":  case "TBSAVE":  case "TBSAVETEMP":  // 取子界面上字段值，填写到本界面数据包  IDynamicFormView expressionView = this.View.GetView(subPageId) as IDynamicFormView;  DynamicObject dataObject = this.View.Model.**DataObject**;  dataObject["EXPRESSION"] = GetFormulaData(expressionView);  dataObject["EXPRESSIONDESC"] = GetFormulaDisplayText();  break;  }  }  #region 私有方法  /// <summary>  /// 显示公式编辑器，嵌套在本界面，公式编辑面板上  /// </summary>  private void ShowExpressionEditor()  {  // 无关代码，略过  }  #endregion  }  } | |

## FunctionLib

支持实体规则运算的函数库；

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| FunctionManage FunctionLib { get; } | |

**备注**

只有函数库中注册过的函数，才能被实体服务规则的前置条件、计算公式等引用，并顺利解析。

如IsPush()函数、IsDraw()函数等，需要先到平台注册，申明函数元数据对象。运行时，会据此生成函数库，支持实体规则的运算；

示例代码暂缺；

## OpenParameter

显示当前界面时，传入的参数。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicFormOpenParameter OpenParameter { get; set; } | |

**备注**

可以从此属性中，获取本界面的打开模式，以及定制参数等；

在父窗口A，显示子窗口B时，可能会需要传入一些定制的参数，以便控制子窗口B的行为；子窗口B，就可以通过本属性，获取到父窗口A传入的定制参数，并据此作出控制；

### 案例 – 列表特定场景下隐藏菜单

**代码来源**

Kingdee.K3.SCM.Business.PlugIn.HideMainBarMenuList

**需求背景**

专用于下推的列表，不能显示新增、修改等菜单；

**实现方案**

在列表界面加载时，读取列表调用参数，判断当前场景是否专用于下推，如果是，则隐藏菜单；

**关键字**

界面调用 定制参数 列表场景 隐藏菜单 界面初始化事件

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.ComponentModel;  using Kingdee.BOS.Core.List.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.Core.Metadata.FormElement;  using Kingdee.BOS.Core.Metadata.BarElement;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.K3.Core;  namespace Kingdee.K3.SCM.Business.PlugIn  {  /// <summary>  /// 目前用于：应收应付调用时候，根据标示来隐藏列表上的菜单  /// 只显示如下菜单：下推、过滤、退出、刷新  /// 并把下推的名称改名  /// </summary>  [Description("隐藏列表上的菜单插件")]  public class HideMainBarMenuList : AbstractListPlugIn  {  public override void OnInitialize(InitializeEventArgs e)  {  base.OnInitialize(e);  this.IsNeed = GetCustomerPara(this.View);  }  public override void AfterBindData(EventArgs e)  {  HideMainBarItemForConvert(this.View, this.ListView, this.IsNeed);  base.AfterBindData(e);  }  /// <summary>  /// 判断是否应收应付选单  /// </summary>  /// <param name="view"></param>  /// <returns></returns>  protected bool GetCustomerPara(IDynamicFormView view)  {  object obj = view.Model.**OpenParameter**.GetCustomParameter(PublicConst.ForList\_Push);  bool isFromARAP = false;  if (obj != null)  {  isFromARAP = true;  this.ReNamePushMenu = obj.ToString();  }  return isFromARAP;  }  /// <summary>  /// 隐藏菜单，应收应付会调用到这边的单据，但是有些菜单又不想显示，需要隐藏.  /// 添加人：wgj 2012-04-12  /// </summary>  /// <param name="view"></param>  /// <param name="listView"></param>  protected void HideMainBarItemForConvert(IDynamicFormView view, IListView listView, Boolean isHide = false)  {  if (!isHide)  {  return;  }  BarDataManager barDataManager = listView.BillLayoutInfo.GetFormAppearance().ListMenu;  List<long> operationids = new List<long>();  operationids.Add(FormOperation.Operation\_Push);  operationids.Add(FormOperation.Operation\_Filter);  operationids.Add(FormOperation.Operation\_Close);  operationids.Add(FormOperation.Operation\_Refresh);  operationids.Add(FormOperation.Operation\_View);  operationids.Add(FormOperation.Operation\_Track); //上下查  operationids.Add(FormOperation.Operation\_TrackUp); //上下查  operationids.Add(FormOperation.Operation\_TrackDown); //下查  long lngOperationId;  BarDataManager barMenus = listView.BillLayoutInfo.GetFormAppearance().ListMenu;  foreach (BarItem item in barMenus.BarItems)  {  lngOperationId = FormOperation.GetOperationIdByMenuBar(view, item);  if (!operationids.Contains(lngOperationId))  {  view.GetMainBarItem(item.Key).Visible = false;  }  else if (lngOperationId.Equals(FormOperation.Operation\_Push))  {  view.GetMainBarItem(item.Key).Text = this.ReNamePushMenu;  }  }  }  /// <summary>  /// 把下推重新命名的名称  /// </summary>  public string ReNamePushMenu  {  get;  set;  }  /// <summary>  /// 是否需要隐藏菜单  /// </summary>  public Boolean IsNeed  {  get;  set;  }  }  } | |

## ParameterData

单据选项数据包。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicObject ParameterData { get; set; } | |

**备注**

可以从此数据包中，获取用户配置各个单据选项值；

示例代码暂缺；

## SubSytemId

子系统内码。

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| string SubSytemId { get; set; } | |

**备注**

为发布后单据所在的子系统，与BOS设计器中单据所在子系统有所差异；

示例代码暂缺；

## UserTypes

用户类型：普通用户、管理员；

### 语法

**属性定义**

|  |  |
| --- | --- |
| C# |  |
| List<Core.Permission.UserType> UserTypes | |

示例代码暂缺；

# 公共方法

## BeginIniti

标记数据模型进入初始化状态，避免触发字段值改变事件；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void BeginIniti(); | |

**备注**

正常情况下，插件修改字段，需要触发字段值改变事件，并刷新界面显示。

但是在某些特殊场景下，却希望所做的改动，不去触发值改变事件，不刷新界面，此时，就需要调用本方法，标记数据模型为初始化状态；

例如价格与含税单价，是互相影响的，录入了价格后，会自动计算含税单价；相反，如果用户录入含税单价，则自动计算价格，但不希望再次重算含税单价，形成一个闭环；此时，可以在插件捕获含税单价的改动DataChanged事件，计算单价，设置单价之前，调用this.Model.BeginIniti()方法，关闭值改变事件，设置完之后，再调用this.Model.EndIniti()方法，开启值改变事件；

此方法必须与EndIniti方法成对调用，否则界面将一直处于初始化状态，任何字段的改动，都不会触发值改变事件；

### 案例 – 根据供应商取价，填写到单据上

**代码来源**

Kingdee.K3.SCM.Stock.Business.PlugIn.InStockEdit

**需求背景**

入库单填写供应商之后，需要取供应商价格填写到单据字段上；

供应商价格，会有多行，需要根据复杂规则，判断应该取那一行的价格携带到界面上，难以通过实体规则实现，需要写插件代码；

**实现方案**

捕获供应商字段值DataChanged事件，取供应商价格列表，根据规则取价，然后填写到本单字段上；

修改字段值的过程，不触发值改变事件，避免重新进入DataChanged事件处理代码，引发价格变动的计算；

价格填写完毕后，主动调用UpdateView方法，要求刷新被修改的价格字段值；

**关键字**

根据条件取基础资料详情 设置初始化状态 结束初始化状态 不触发值改变事件 强制要求刷新界面字段

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Util;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.SqlBuilder;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Resource;  using Kingdee.BOS.ServiceHelper;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.SCM.ServiceHelper;  using Kingdee.K3.SCM.Business;  using Kingdee.K3.Core.SCM.Args;  using Kingdee.K3.Core.MFG.PRD.BackFlush;  namespace Kingdee.K3.SCM.Stock.Business.PlugIn  {  /// <summary>  /// 入库单 单据维护界面插件  /// </summary>  public class InStockEdit : AbstractBillPlugIn  {  /// <summary>  /// 字段值改变事件  /// </summary>  /// <param name="e"></param>  public override void DataChanged(DataChangedEventArgs e)  {  switch (e.Field.Key.ToUpper())  {  case "FSUPPLIERID"://供应商(如果修改了供应商，需要清除供货方，结算方和收款方等相关信息；)  case "FSUPPLYID"://供货方（如果修改了供货方，重新取价目表，折扣表）  // 此处略过无关代码600行...  //根据供应商携带供货方，收款方，结算方等信息。 跨表查询携带 BOS 暂不支持，在插件中实现。  DoSupplyChange(e);  SetSupplyAddress(e);  break;  case "FPURCHASERID"://采购员，默认携带采购员的部门和默认采购组  case "FSTOCKERID"://仓管员，默认携带仓管员的部门和默认仓管组  // 此处略过无关代码600行...  break;  default:  break;  }  }  private void SetSupplyAddress(DataChangedEventArgs e)  {  // 此处略过无关代码600行...  }  /// <summary>  /// 根据供应商携带供货方，收款方，结算方等信息。 跨表查询携带 BOS 暂不支持，在插件中实现  /// </summary>  /// <param name="e"></param>  private void DoSupplyChange(DataChangedEventArgs e)  {  this.Model.**BeginIniti**();  DynamicObject objSupplier = this.View.Model.GetValue("FSupplierId") as DynamicObject;  long supplierId = objSupplier == null ? 0 : Convert.ToInt64(objSupplier["Id"]);  if (supplierId > 0)  {  OQLFilter ofilter = new OQLFilter();  OQLFilterHeadEntityItem filteritem = new OQLFilterHeadEntityItem();  filteritem.FilterString = string.Format(" FSupplierId = {0} ", supplierId.ToString());  ofilter.Add(filteritem);  List<SelectorItemInfo> supplierselector = new List<SelectorItemInfo>();  supplierselector.Add(new SelectorItemInfo("FNumber"));  supplierselector.Add(new SelectorItemInfo("FName"));  supplierselector.Add(new SelectorItemInfo("FPRICELISTID"));  supplierselector.Add(new SelectorItemInfo("FPayCondition"));  supplierselector.Add(new SelectorItemInfo("FPayCurrencyId"));  supplierselector.Add(new SelectorItemInfo("FSettleTypeId"));  DynamicObject oSupplier = BusinessDataServiceHelper.Load(  this.Context,  BOS.Core.FormIdConst.BD\_Supplier,  supplierselector,  ofilter).FirstOrDefault();  object opricelist = null;  object opaycondition = null;  object osettletypeid = null;  object ocurrencyid = null;  if (((DynamicObjectCollection)oSupplier["SupplierBusiness"]).Count > 0)  {  opricelist = ((DynamicObjectCollection)oSupplier["SupplierBusiness"])[0]["PriceListID"]; //oSupplier["PriceListID"];  osettletypeid = ((DynamicObjectCollection)oSupplier["SupplierBusiness"])[0]["SettleTypeId"];  }  if (((DynamicObjectCollection)oSupplier["SupplierFinance"]).Count > 0)  {  ocurrencyid = ((DynamicObjectCollection)oSupplier["SupplierFinance"])[0]["PayCurrencyId"];  opaycondition = ((DynamicObjectCollection)oSupplier["SupplierFinance"])[0]["PayCondition"];  }  //默认结算币别  this.View.Model.SetValue("FSettleCurrId", ocurrencyid != null ? ((DynamicObject)ocurrencyid)["Id"] : 0);  //入库单的付款条件、结算方式 在表头财务页签  this.View.Model.SetValue("FPayConditionId", opaycondition != null ? ((DynamicObject)opaycondition)["Id"] : null);  this.View.Model.SetValue("FSettleTypeId", osettletypeid != null ? ((DynamicObject)osettletypeid)["Id"] : null);  }  else  {  //this.View.Model.SetValue("FDiscountListId", null);  //this.View.Model.SetValue("FPriceListId", null);  this.View.Model.SetValue("FSettleCurrId", null);  }  this.Model.**EndIniti**();  this.View.UpdateView();  }  }  } | |

## EndIniti

标记数据模型已经结束了初始化，可以触发值改变事件；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void EndIniti(); | |

**备注**

此方法需要与BeginIniti方法成对调用；

示例代码参阅[BeginIniti](#_案例_–_取供应商价格填写到单据上)。

## CreateNewData

新建空白数据包，并刷新界面；

或者传入自行构建的数据包，刷新界面；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void CreateNewData();  void CreateNewData(DynamicObject newObject); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| DynamicObject newObject | 动态表单数据包；如果传入了此参数，将把此数据包绑定到界面上 |

**备注**

注意，所谓的空白数据包，实际上并不完全空白，各个字段已经设置了默认值；单据体，也创建了默认行；

示例代码暂缺；

## CreateNewEntryRow

向实体中增加一行，自动填写字段的默认值，并刷新到界面单据体表格；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void CreateNewEntryRow(string key);  void CreateNewEntryRow(Entity entity, int rowIndex); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| Entity entity | 实体 |
| int rowIndex | 指定新行位置 |

**备注**

默认情况下，新建的行会放在最后，也可以直接指定新行的行号；

示例代码暂缺；

### 案例 – 结算组织改变后重置单据体数据

**代码来源**

Kingdee.K3.SCM.IOS.Business.PlugIn.IOSCreateBalanceList

**需求背景**

内部结算单，结算组织是一个非常关键的字段，单据体中的数据，需要依赖与结算组织值，如果结算组织发生了改变，则必须重置单据体；

**实现方案**

捕获字段值改变事件，如果结算组织发生了改变，则清除单据体数据，并新增一空行；

**关键字**

清除单据体数据 单据体新增一行

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.K3.Core.SCM.IOS;  using Kingdee.K3.SCM.ServiceHelper;  namespace Kingdee.K3.SCM.IOS.Business.PlugIn  {  public class IOSCreateBalanceList : AbstractDynamicFormPlugIn  {  /// <summary>  /// 核算组织  /// </summary>  private const string KEY\_ACCTORG = "FACCTORGID";  /// <summary>  /// 核算体系  /// </summary>  private const string KEY\_ACCTSYS = "FACCTSYSID";  /// <summary>  /// 业务组织  /// </summary>  private const string KEY\_SETTLEORG = "FSETTLEORGID";  #region 基类事件  /// <summary>  /// 界面加载事件  /// </summary>  /// <param name="e"></param>  public override void OnLoad(EventArgs e)  {  base.OnLoad(e);  if (this.CheckIsAcctOrg(this.Context))  {  // 取当前组织，填写到核算组织字段  this.View.Model.SetItemValueByID("FACCTOrgId", this.Context.CurrentOrganizationInfo.ID, 0);  }  }  /// <summary>  /// 菜单单击事件  /// </summary>  /// <param name="e"></param>  public override void BarItemClick(BarItemClickEventArgs e)  {  // 无关逻辑，略去代码600行  }  /// <summary>  /// 基础资料显示F8列表前事件，附加过滤条件  /// </summary>  /// <param name="e"></param>  public override void BeforeF7Select(BeforeF7SelectEventArgs e)  {  base.BeforeF7Select(e);  string ActionKey = e.FieldKey.ToUpperInvariant();  switch (ActionKey)  {  case KEY\_ACCTORG:  e.ListFilterParameter.Filter = GetACCTORGFilter();  break;  case KEY\_ACCTSYS:  // 无关逻辑，略去代码600行  break;  case KEY\_SETTLEORG:  // 无关逻辑，略去代码600行  break;  default:  break;  }  }  /// <summary>  /// 填写基础资料之前，对返回的基础资料进行校验  /// </summary>  /// <param name="e"></param>  public override void BeforeSetItemValueByNumber(BeforeSetItemValueByNumberArgs e)  {  base.BeforeSetItemValueByNumber(e);  string ActionKey = e.BaseDataFieldKey.ToUpperInvariant();  switch (ActionKey)  {  case KEY\_ACCTORG:  e.Filter = GetACCTORGFilter();  break;  case KEY\_ACCTSYS:  // 无关逻辑，略去代码600行  break;  case KEY\_SETTLEORG:  // 无关逻辑，略去代码600行  break;  default:  break;  }  }  /// <summary>  /// 字段值改变前事件，在此事件，可以确认是否接受新值  /// </summary>  /// <param name="e"></param>  public override void BeforeUpdateValue(BeforeUpdateValueEventArgs e)  {  base.BeforeUpdateValue(e);  if (e.Key.ToUpperInvariant() == KEY\_ACCTORG)  {  // 核算组织发生了改变，单据体中的数据需要重置 -> 删除全部行，然后增加一空行  this.View.Model.SetItemValueByNumber(KEY\_ACCTSYS, "0", -1);  this.View.Model.**DeleteEntryData**("FEntity");  this.View.Model.**CreateNewEntryRow**("FEntity");  this.View.UpdateView("FEntity");  }  }  public override void DataChanged(DataChangedEventArgs e)  {  // 无关逻辑，略去代码600行  }  #endregion  #region 功能  /// <summary>  /// 获取核算组织的过滤条件  /// </summary>  /// <returns></returns>  private string GetACCTORGFilter()  {  // 无关逻辑，略去代码600行  }  private bool CheckIsAcctOrg(Context ctx)  {  // 无关逻辑，略去代码600行  }  #endregion  }  } | |

## BatchCreateNewEntryRow

批量新建单据体行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void BatchCreateNewEntryRow(string key, int rowCount);  void BatchCreateNewEntryRow(string key, DynamicObject rowDataEntity, int rowCount); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int rowCount | 批量创建的行数 |
| DynamicObject rowDataEntity | 以此数据包为模板，批量创建其他行 |

**备注**

单据体行的创建，会自动填写默认值、触发值改变事件，自动计算其他字段值，这些工作都需要花费时间。

一次需创建很多行时，将要创建的所有新行，所处的上下文环境是一样的，除了第一行需要填写默认值、触发值改变事件之外，其他行，完全可以复制第一行的数据，从而节省计算的时间。

本方法，就是在创建第一行成功之后，把第一行作为模板，批量复制产生出其他新行；

### 案例 – 根据评估模型批量创建信用评估单据体全部行

**代码来源**

Kingdee.K3.SCM.Credit.Business.PlugIn.CreditIndexSelect

**需求背景**

新建供应商信用评估表时，需要加载评估模型的全部行，在单据体中一一创建评估行；

评估模型可能会有很多行，用批量创建可以节省很多时间；

**实现方案**

捕获单据的新单数据包创建完毕事件(AfterCreateNewData)事件，在此事件中，读取评估模型，批量创建相应单据体行数，逐行把评估模型数据填入单据体；

**关键字**

批量创建单据体行 设置单据体行背景色 数据包创建完毕事件

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Orm.Metadata.DataEntity;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.DynamicForm.PlugIn;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using Kingdee.K3.SCM.Core.CRE;  namespace Kingdee.K3.SCM.Credit.Business.PlugIn  {  public class CreditIndexSelect : AbstractDynamicFormPlugIn  {  #region 重载函数  /// <summary>  /// 构造动态表单表体数据  /// </summary>  public override void AfterCreateNewData(EventArgs e)  {  // 获取供评估模型选择的信用评估指标数据  DynamicObjectCollection doSelectIndex = SCM.ServiceHelper.CreditServiceHelper.GetCreditIndexListForSelect(this.Context);  // 定义颜色  List<KeyValuePair<int, string>> colors = new List<KeyValuePair<int, string>>();  // 批量创建数据行  this.View.Model.**BatchCreateNewEntryRow**("FTreeEntity", doSelectIndex.Count());  for (int i = 0; i < doSelectIndex.Count(); i++)  {  if (Convert.ToChar(doSelectIndex[i]["FLASTLEAF"]) == '1')  {  this.View.Model.SetValue("FKey", doSelectIndex[i]["FID"], i);  this.View.Model.SetValue("FCreditIndex", doSelectIndex[i]["FID"], i);  this.View.Model.SetValue("FIsSelect", false, i);  this.View.Model.SetValue("FIndexParentId", -1, i);  this.View.Model.SetValue("FRowType", 16, i);  colors.Add(new KeyValuePair<int, string>(i, "#FFE4E1"));  }  else  {  this.View.Model.SetValue("FKey", doSelectIndex[i]["FID"], i);  this.View.Model.SetValue("FCreditIndex", doSelectIndex[i]["FID"], i);  this.View.Model.SetValue("FIsSelect", false, i);  this.View.Model.SetValue("FIndexParentId", doSelectIndex[i]["FGROUP"], i);  this.View.Model.SetValue("FRowType", 16, i);  }  }  EntryGrid grid = this.View.GetControl<EntryGrid>("FTreeEntity");  grid.SetRowBackcolor(colors);  this.View.UpdateView("FTreeEntity");  }  public override void DataChanged(DataChangedEventArgs e)  {  // 无关逻辑，略去代码600行  }  public override void EntryBarItemClick(BarItemClickEventArgs e)  {  // 无关逻辑，略去代码600行  }  #endregion  }  } | |

## InsertEntryRow

向单据体中插入新行，并放在第row行上；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void InsertEntryRow(string key, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int row | 指定新行行号 |

**备注**

新行插入完毕，自动处理各字段默认值，并调整行序号；

示例代码暂缺；

## ImportEntry

把uploadFile参数指定的Excel文件，导入到key参数对应的单据体中；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void ImportEntry(string key, string uploadFile); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| string uploadFile | 上传的文件，在临时目录中的完整文件名 |

**备注**

特别注意：要求Excel中包含完整的单据体字段，如果缺少字段，会中断报错；另外，导入为追加模式；

示例代码暂缺；

## CopyEntryColumn

把指定行的某字段值复制到整列；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void CopyEntryColumn(string key, int row, string field); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int row | 来源行号 |
| string field | 字段 |

**备注**

把指定单据体行上的某个字段值，复制到其他行上，其他行字段原值，将会被覆盖；

示例代码暂缺；

## CopyEntryRow

复制行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void CopyEntryRow(string key, int row, int newRow, bool isCopyLinkEntry = false); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int row | 来源行号 |
| int newRow | 目标行号；  如果为-1，则自动创建新行，放在最后，作为目标行 |
| bool isCopyLinkEntry | 是否复制关联关系；  关联关系记录了本行与上游单据的关系，默认不复制 |

**备注**

单据体中的字段，如果控制选项中，勾选了不允许复制，则在复制行时，略过这些字段；

示例代码暂缺；

## CopyEntryRowFollowCurrent

在行号curFocuseRowIndex之后，插入新行，并把行row的字段值，复制过来；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void CopyEntryRowFollowCurrent(string key, int curFocusRowIndex, int row, bool isCopyLinkEntry = false); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int curFocusRowIndex | 在此位置，创建新行，并作为目标行 |
| int row | 来源行号 |
| bool isCopyLinkEntry | 是否复制关联关系；  关联关系记录了本行与上游单据的关系，默认不复制 |

**备注**

于本函数在字段值的复制处理上，同CopyEntryRow方法；

示例代码暂缺；

## MoveDownEntryRow

把单据体表格中的数据行，往下移动；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void MoveDownEntryRow(string key, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int row | 需下移的行号 |

**备注**

移动完毕后，行序号并没有重排，需要自行处理；功能实现不太完备，请谨慎使用；

示例代码暂缺；

## MoveUpEntryRow

把单据体表格中的数据行，往上移动；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void MoveUpEntryRow(string key, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int row | 需上移的行号 |

**备注**

移动完毕后，行序号并没有重排，需要自行处理；功能实现不太完备，请谨慎使用；

示例代码暂缺；

## DeleteEntryData

清空指定单据体的全部行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void DeleteEntryData(string key); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |

**备注**

示例代码，参阅[CreateNewEntryRow](#_案例_–_结算组织改变后重置单据体数据)；

## DeleteEntryRow

删除单据体的指定行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void DeleteEntryRow(string key, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int row | 需删除的行号 |

**备注**

示例代码暂缺；

## GetDirty

获取单据数据包的脏标志状况；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| bool GetDirty(); | |

**备注**

保存单据数据时，会根据字段的改动情况，生成更新、插入、删除SQL语句，单据数据包没有进行过任何更改，保存时，不会执行任何SQL语句；

字段如果发生了改动，则会记录此字段的脏标志，单据体行发生了变动，也会记录脏标志；

单据数据包的脏标志，就是判断数据包是否发生过变更的依据；

如果单据数据包有更改，本方法即返回true；

如果返回false，表示未新增、删除行，字段也没有修改过；

示例代码暂缺；

## ClearDirty

清除整单数据包，或者所指定单据体行数据包中的脏标记；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void ClearDirty();  void ClearDirty(string entityKey, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string entityKey | 实体Key |
| int row | 需清除脏标记的行号 |

**备注**

BOS平台在保存单据时，会根据数据包中各字段的脏标记，判断字段值是否发生了改动，如果未发生改动，保存时略过此字段；

清除了脏标记，保存操作会判断字段未更改过，略过不更新；请谨慎调用此方法；

示例代码略；

## ClearNoDataRow

清除空行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void ClearNoDataRow(); | |

**备注**

单据界面在加载数据包时，为方便用户编辑，会为单据体自动追加一空白行，保存前，这些空行需要调用本方法清除掉。

默认情况下，平台会自动调用本方法清除空白行，如果需要在插件中自行调用保存操作，则需要在调用保存操作前，调用本方法清除空行；

清除空行的主要依据，是单据体的关键字段是否有值，如果关键字段设置了默认值，则系统会判定此行非空白，不予删除；

示例代码暂缺；

## GetBaseDataFieldByKey

到单据元数据中，取指定基础资料字段的元数据；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| BaseDataField GetBaseDataFieldByKey(string key); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 字段key |

**备注**

如果给定的key不存在，返回null；

如果字段不是基础资料字段，返回null；

示例代码暂缺；

## IsFlexField

判断字段是否为弹性域维度字段；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| bool IsFlexField(string key); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 字段key |

**备注**

辅助资料、仓位、核算项目等弹性域字段，会包含很多维度，也要展示在表格中，供用户直接编辑。这些维度字段，并不是单据的原生字段，是把弹性域字段展开后，自动产生的；在某些功能上，可能需要做特别处理，因此，需要区分出这些维度字段；

示例代码暂缺；

## GetDecimal

获取指定数值字段的精度；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| int GetDecimal(string key); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 字段key |

**备注**

如果字段不存在、或者字段非数值类型，返回0；

示例代码暂缺；

## GetEntityDataObject

获取指定单据体的全部行数据包集合，或者取指定单据体的行row数据包；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicObjectCollection GetEntityDataObject(Entity entity);  DynamicObject GetEntityDataObject(Entity entity, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| Entity entity | 实体 |
| int row | 行号 |

**备注**

子单据体，仅返回归属于父单据体当前焦点行的子集合，其他与当前父单据体焦点行无关的子单据体行，不会返回；

### 案例 – 取单据体所选行的内码

**代码来源**

Kingdee.K3.SCM.Credit.Business.PlugIn.CreditBillEdit

**需求背景**

信用管理界面，特别定义了几个操作，针对单据体所选行数据进行处理；支持多选行，批量处理。

服务端处理时，需要有单据体行内码，但是界面上拿到的只是行号，需要到行数据包中取；

**实现方案**

在操作执行前事件，读取单据体所选行的行号，根据行号，到单据体数据集合中取行数据包，然后再取单据体行内码，作为操作附加参数，传入操作服务端组件；

**关键字**

根据单据体行号取行内码，操作定制参数参入

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS.Util;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.K3.Core.SCM;  using Kingdee.K3.SCM.ServiceHelper;  namespace Kingdee.K3.SCM.Credit.Business.PlugIn  {  /// <summary>  /// 信用管理专属插件  /// </summary>  public class CreditBillEdit : AbstractBillPlugIn  {  private string \_startCre = "False";  public override void OnBillInitialize(BillInitializeEventArgs e)  {  base.OnBillInitialize(e);  // 取系统参数  \_startCre = Convert.ToString(CommonServiceHelper.GetSystemProfile(this.View.Context,  SCMConst.SysParamDataCenterOrgID,  Kingdee.K3.Core.SCM.SCMFormIdConst.CRE\_SystemParameter,  Kingdee.K3.Core.SCM.SCMConst.PARAM\_StartCreditCtl,  "False"));  }  public override void AfterBindData(EventArgs e)  {  // 设置菜单可见性  SetCreditMenuVisible();  }  public override void AfterCreateNewData(EventArgs e)  {  // 设置菜单可见性  SetCreditMenuVisible();  }  /// <summary>  /// 用户点击菜单后，调用操作处理代码前  /// </summary>  /// <param name="e"></param>  public override void BeforeDoOperation(BeforeDoOperationEventArgs e)  {  var info = this.View.Model.BillBusinessInfo;  var form = info.GetForm();  if (form.Id.EqualsIgnoreCase(SCMFormIdConst.SAL\_ORDER))  {  //销售订单 行操作  switch (e.Operation.FormOperation.Operation.ToUpper())  {  case "YLTERMINATE":  case "YLUNTERMINATE":  case "YLMRPCLOSE":  case "YLUNMRPCLOSE":  //支持多选了！！  //定义一个集合，收集多选的单据体行内码  List<long> pkEntryIds = new List<long>();  //取得单据体表格所选行号  int[] rows = this.View.GetControl<EntryGrid>("FSaleOrderEntry").GetSelectedRows();  //根据行号，到单据体数据集合中找行数据包，取内码  Entity entryEntity = info.GetEntity("FSaleOrderEntry");  DynamicObjectCollection objCollection = this.View.Model.**GetEntityDataObject**(entryEntity);  if (rows != null && rows.Length > 0)  {  foreach (var row in rows)  {  pkEntryIds.Add(Convert.ToInt64(objCollection[row]["Id"]));  }  }  //设置操作定制参数  e.Option.SetVariableValue("OperationRows", pkEntryIds);  break;  }  }  base.BeforeDoOperation(e);  }  private void SetCreditMenuVisible()  {  if (\_startCre.EqualsIgnoreCase("False"))  {  //设置菜单可见性  BarItemControl menuItem = this.View.GetMainBarItem("tbQueryCredit");  if (!menuItem.IsNullOrEmpty())  {  menuItem.Visible = false;  }  }  }  }  } | |

## TryGetEntryCurrentRow

尝试返回单据体焦点行数据包、行号；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| bool TryGetEntryCurrentRow(string entryKey, out DynamicObject row, out int rowIndex); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string entryKey | 实体Key |
| DynamicObject row | 输出焦点行数据包 |
| int rowIndex | 输出焦点行号 |

**备注**

如果指定的单据体中无数据行，则本函数返回false，输出参数row为null，rowIndex为-1；

示例代码暂缺；

## GetEntryCurrentRowIndex

获取单据体当前焦点行号；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| int GetEntryCurrentRowIndex(string key); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |

**备注**

对单据体字段赋值SetValue时，需要传入行号，如果当时的上下文中，并没有行号参数，则可以调用此方法，获取单据体焦点行号；

### 案例 – 取当前物料的销售价格返回

**代码来源**

Kingdee.K3.SCM.Business.DynamicForm.Operation.QuerySalPrice

**需求背景**

供应链领域，封装了一个通用的操作：取销售价格返回；

此操作需要根据界面上录入的价目表、物料等信息，显示出对应的销售价格列表，供用户查询；

难点在于显示销售价格列表之前，需要取得界面上的相关字段值，如果字段在单据体，必须取焦点行的字段值；

**实现方案**

编写操作实现类(AbstractBillOperation)，重载操作执行方法。

依赖IDynamicFormModel接口，取得当前界面上，价目表、物料等字段值，并拼接出过滤条件；

并把过滤条件传入，据此显示、过滤销售价格列表；

**关键字**

自定义操作，操作实现类，显示列表界面，设置列表过滤条件，取单据体字段值，取单据体当前行的字段值；

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Kingdee.BOS;  using Kingdee.BOS.Util;  using Kingdee.BOS.Core;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Core.Metadata.FormElement;  using Kingdee.BOS.Core.Bill;  using Kingdee.BOS.Core.List;  using Kingdee.BOS.JSON;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.CommonFilter;  using Kingdee.BOS.Core.Metadata.FieldElement;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Core.Permission;  using Kingdee.BOS.ServiceHelper;  using Kingdee.BOS.Resource;  namespace Kingdee.K3.SCM.Business.DynamicForm.Operation  {  public class QuerySalPrice : AbstractBillOperation  {  /// <summary>  /// 获取新的实例  /// </summary>  /// <returns></returns>  public override IFormOperation GetNewInstance()  {  return new QuerySalPrice();  }  protected override bool ExecuteOperation()  {  ////是否有权限  //if (this.FuncPermissionAuth() == false)  //{  // return false;  //}  //验证是否有价目表的查看权限  PermissionAuthResult authResult = PermissionServiceHelper.FuncPermissionAuth(  this.View.Context,  new BusinessObject() { Id = K3.Core.SCM.SCMFormIdConst.SAL\_PRICECATEGORY },  PermissionConst.View);    if (!authResult.Passed)  {  this.View.ShowMessage("没有价目表的查看权限，不允许查询！");  return false;  }  if (!(this.FormOperation.Parmeter is OperationParmListParameter))  {  return false;  }  // 操作配置参数，其中记录了物料该取那个字段...  OperationParmListParameter operparm = (OperationParmListParameter)this.FormOperation.Parmeter;  if (this.View is IListViewService)  {  // 列表界面，不支持价格查询  return false;  }  else  {  StringBuilder sbFilter = new StringBuilder();  sbFilter.AppendLine(" FDocumentStatus='C' AND FForbidStatus='A' AND FEntryForbidStatus = 'A' AND FROWAUDITSTATUS='A'");  // 取物料字段Key  List<ParmValue> parms = operparm.InParamFieldList;  ParamInfo materialInfo = GetParamInfo(parms, "FMaterialId");  if (materialInfo.IsNullOrEmpty())  {  this.View.ShowWarnningMessage("请正确配置服务！");  return false;  }  string materialKey = materialInfo.BillFieldKey;  // 取界面上价格表字段值  // 这是通用服务啊，这些字段名怎么能都写死呢？ 要改造！！！！  DynamicObject pricelist = this.View.Model.GetValue("FPriceListId") as DynamicObject;  long pricelistId = pricelist == null?0:Convert.ToInt64(pricelist["Id"]);  if (pricelistId == 0)  {  this.View.ShowWarnningMessage("没有价目表，不能查询！");  return false;  }  // 条件中，拼接上价目表内码  sbFilter.AppendFormat(" AND FID={0} ", pricelistId.ToString());  // 有效日期范围  DateTime billdate = Convert.ToDateTime(this.View.Model.GetValue("FDate"));  sbFilter.AppendFormat(" AND FEntryEffectiveDate <={0}", billdate.ToKSQlFormat());  sbFilter.AppendFormat(" AND FEntryExpiryDate >{0}", billdate.ToKSQlFormat());  string priceobject = Convert.ToString(pricelist["PriceObject"]);    // 取物料字段所在单据体，以及其焦点行  string entityKey = this.View.GetControl("FMaterialId").ControlAppearance.EntityKey;  int rowIndex = this.View.Model.**GetEntryCurrentRowIndex**(entityKey);    // 取所选物料  DynamicObject material = this.View.Model.GetValue("FMaterialId",rowIndex) as DynamicObject;  // 价目表取价方法  if (priceobject == "A") //按物料  {  long materialId = material == null ? 0 : Convert.ToInt64(material["Id"]);  long materialmasterId = material == null ? 0 : Convert.ToInt64(material[FormConst.MASTER\_ID]);  //sbFilter.AppendFormat(" AND FMaterialId = {0} ",materialId.ToString());  sbFilter.AppendFormat(" AND FMaterialId in (SELECT FMaterialId FROM T\_BD\_MATERIAL WHERE FMasterID={0}) ",  materialmasterId.ToString());  }  else  {  // 按物料类别  DynamicObjectCollection materialBase = (DynamicObjectCollection)material["MaterialBase"];  if (materialBase.Count != 0)  {  DynamicObject materialType = (DynamicObject)(materialBase[0])["CategoryID"];  //long materialTypeId = Convert.ToInt64(((DynamicObject)materialBase[0])["CategoryID\_Id"]);  long materialTypemsterId = materialType == null ?0: Convert.ToInt64(materialType[FormConst.MASTER\_ID]);  //sbFilter.AppendFormat(" AND FMaterialTypeId = {0} ", materialTypeId.ToString());  sbFilter.AppendFormat(" AND FMaterialTypeId in (SELECT FCATEGORYID FROM T\_BD\_MATERIALCATEGORY WHERE FMasterID={0}) ",  materialTypemsterId.ToString());  }  }  // 计价单位  DynamicObject unitObj = this.View.Model.GetValue("FUnitId", rowIndex) as DynamicObject;  long unitId = unitObj == null ? 0 : Convert.ToInt64(unitObj["Id"]);  DynamicObject priceunitObj = this.View.Model.GetValue("FPriceUnitId", rowIndex) as DynamicObject;  long priceunitId = priceunitObj == null ? 0 : Convert.ToInt64(priceunitObj["Id"]);    //需要加单位过滤吗？  if (unitId > 0)  {  sbFilter.AppendFormat(" AND (FUnitId ={0} OR FUnitId ={1}) ", unitId.ToString(),priceunitId.ToString());  }  decimal qty = Convert.ToDecimal(this.View.Model.GetValue("FQty", rowIndex));  sbFilter.AppendFormat(" AND (FToQty >= {0} OR FToQty=0) ", qty.ToString());  // 显示销售价格查询列表，并设置过滤条件  ListShowParameter ShowPara = new ListShowParameter();  ListRegularFilterParameter filterparam = new ListRegularFilterParameter();  //DynamicFormShowParameter ShowPara = new ListShowParameter();  ShowPara.ParentPageId = this.View.PageId;  ShowPara.MultiSelect = false;  ShowPara.FormId = "SAL\_QueryPriceList";  ShowPara.Height = 600;  ShowPara.Width = 800;  ShowPara.Caption = "销售价格查询";  ShowPara.IsLookUp = true;  // 传入过滤条件  filterparam.Filter = sbFilter.ToString();  ShowPara.ListFilterParameter = filterparam;  ShowPara.IsLookUp = true;  this.View.ShowForm(ShowPara);  return true;  }  }  private class ParamInfo  {  public string Key;  public string BillFieldKey;  public bool NeedFilter;  }  private ParamInfo GetParamInfo(List<ParmValue> parms, string key)  {  ParamInfo info = new ParamInfo();  ParmValue value = parms.Find(o => o.FieldKey == key);  if (!value.IsNullOrEmpty())  {  info.Key = value.FieldKey;  info.BillFieldKey = value.BillFieldKey;  info.NeedFilter = value.NeedFilter;  }  return info;  }  }  } | |

## SetEntryCurrentRowIndex

设置单据体的焦点行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void SetEntryCurrentRowIndex(string key, int rowIndex); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |
| int rowIndex | 焦点行号 |

**备注**

子单据体数据集合与单据体焦点密切相关，如果需要对其他单据体行所属的子单据体行进行数据处理，需要切换焦点；

示例代码暂缺；

## GetRowIndex

返回单据体数据行的行号，行号从0开始；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| int GetRowIndex(Entity entity, DynamicObject activeRow); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| Entity entity | 实体 |
| DynamicObject activeRow | 实体行数据包 |

**备注**

本方法，能够把行数据包转换为行号，作为Model对象的其他方法的参数；

示例代码暂缺；

## GetEntryRowCount

获取单据体行数；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| int GetEntryRowCount(string key); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 实体Key |

**备注**

子单据体，返回父单据体当前焦点行所含子集合行数；其他父单据体行下的子单据体行，不包括在内；

示例代码暂缺；

## GetChildFieldRowIndex

根据父字段所在的单据体行，反查指定子字段所在的单据体的对应行号；常用于在刷新父字段后，同步刷新子字段之前，定位子字段焦点行；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| int GetChildFieldRowIndex(Field parentField, Field childField, int parentFieldRowIndex); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| Field parentField | 父字段 |
| Field childField | 子字段 |
| int parentFieldRowIndex | 父单据体行号 |

**备注**

示例代码暂缺；

## LoadFieldDataWithCoreFilter

附加基础资料限定条件，取其数据包；如果条件不满足，则返回null；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicObject LoadFieldDataWithCoreFilter(BaseDataField field, object pkValue, DynamicObject activeRow); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| BaseDataField field | 基础资料字段 |
| object pkValue | 内码 |
| DynamicObject activeRow | 基础资料所在单据体行数据包，需要据此取基础资料的控制字段值，以组成完整的过滤条件 |

**备注**

基础资料在单据上的使用时，可能需要一些额外的限制，如需要受数据授权、组织隔离等限制，也可能是需要受到其他字段所选值限制，比如供应商联系人，并不能选择任意联系人填写到单据上，必须选择当前供应商名下的联系人；

因此，插件给基础资料字段赋值时，随意的给一个pkValue，可能赋值不会成功；

本函数，会自动考虑各种限制条件，如果条件满足，则返回pkValue对应的基础资料数据包，如果条件不满足，则返回null；

示例代码暂缺；

## LoadReferenceData

到数据库中，加载dt模型，内码为pkValue的数据包；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicObject LoadReferenceData(DynamicObjectType dt, object pkValue); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| DynamicObjectType dt | 基础资料ORM数据模型：  BaseDataField.RefFormDynamicObjectType |
| object pkValue | 内码 |

**备注**

本方法可以如下代码替代：

BusinessDataServiceHelper.LoadSingle(this.Context, pkValue, dt, option);

差别在于如果数据库中不存在内码为pkValue的数据，本方法不会报中断，而BusinessDataServiceHelper.LoadSingle会报中断；

示例代码暂缺；

## LoadReferenceDataBatch

到数据库中，批量加载数据包；数据包模型为dt，数据内码为pkValues集合；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| DynamicObject[] LoadReferenceDataBatch(DynamicObjectType dt, object[] pkValues); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| DynamicObjectType dt | 基础资料ORM数据模型：  BaseDataField.RefFormDynamicObjectType |
| object[] pkValues | 内码数组 |

**备注**

本方法可以如下代码替代：

BusinessDataServiceHelper.Load(this.Context, pkValues, dt);

示例代码暂缺；

## GetValue

读取字段值；

### 语法

方法定义：

|  |  |
| --- | --- |
| C# |  |
| object GetValue(Field field);  object GetValue(string key);  object GetValue(Field field, int row);  object GetValue(string key, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| Field field | 字段 |
| string key | 字段Key |
| int row | 行号，如果不指定行号，取当前焦点行 |

**返回值**

字段类型不同，本函数的返回值类型也不同，可能的值类型如下：

|  |  |
| --- | --- |
| 值类型 | 字段类型 |
| string | 文本、多行文本、多语言文本、下拉列表、单据状态、单选按钮组等 |
| decimal | 小数、数量、基本单位数量等 |
| long | 整数等 |
| DateTime | 日期、长日期、时间 |
| bool | 复选框 |
| DynamicObject | 基础资料、辅助资料 |
| DynamicObjectCollection | 多选基础资料、单据体 |

**备注**

如果字段在单据体表格中，还需要指定单据体行号；如果不指定行号，则默认取单据体焦点行上的字段值；

示例代码暂缺；

## SetValue

为字段赋值。

### 语法

方法定义：

|  |  |
| --- | --- |
| C# |  |
| void SetValue(string key, object value);  void SetValue(Field field, DynamicObject activeRow, object value);  void SetValue(string key, object value, int row);  void SetValue(Field field, DynamicObject rowDataEntity, object value, int rowIndex); | |

**参数说明**

参数key为字段key；

参数row为单据体行号，如果为单据头字段赋值，可以略过此参数；

参数value为字段值，不同类型的字段可以接受的值不同，下表列出一些常用字段的值类型：

|  |  |
| --- | --- |
| 字段类型 | 值类型 |
| 文本 | String |
| 小数 | decimal |
| 整数 | Long |
| 日期 | DateTime |
| 长日期 | DateTime |
| 多行文本 | string |
| 组织 | Long |
| 复选框 | Bool |
| 下拉列表 | String |
| 单据编号 | String |
| 基础资料 | DynamicObjec或者基础资料内码(string或long) |
| 基础资料属性 | 不能赋值 |
| 多类别基础资料列表 | String |
| 多类别基础资料 | DynamicObjec或者基础资料内码long |
| 创建人 | 不能赋值 |
| 用户 | DynamicObjec或者用户内码long |
| 批次 | 根据录入模式确定值类型，可能为批号string或者是批号主档内码long |
| 单价 | decimal |
| 金额 | decimal |
| 数量 | decimal |
| 多选基础资料 | 单条基础资料数据包DynamicObject  单个基础资料内码long或者 string  多个基础资料内码IEnumerable<T> |
| 单选按钮 | 不能赋值 |
| 创建日期 | 不能赋值 |
| 修改人 | 不能赋值 |
| 修改日期 | 不能赋值 |
| 多选下拉列表 | String |
| 单选辅助资料列表 | DynamicObjec或者辅助资料内码string |
| 多选辅助资料列表 | 多条辅助资料数据集合DynamicObjectCollection  单条辅助资料数据包DynamicObject  多个辅助资料内码string或者用”,”分隔各内码  多个辅助资料内码object[] |
| 图片字段 | String |
| 金额大写控件 | 不能赋值 |
| 单选按钮组 | string |
| 分组 | DynamicObjec或者分组内码long |
| 多语言文本 | String |
| 基本单位数量 | decimal |
| 计量单位 | DynamicObjec或者单位内码long |
| 折扣率 | Decimal |
| 折扣额 | Decimal |
| 组合字段 | JSONObject其中包含了子项Key与Value |
| 颜色 | String |
| 打印次数 | 不能赋值 |
| 打印时间 | 不能赋值 |
| 单据体图片字段 | String |
| 富文本 | string或者byte[] |
| 上传文件字段 | string或者JSONArray |
| 富文本内容编辑 | String |
| 跨组织业务类型 | DynamicObjec或者内码long |
| 附件字段 | string或者JSONArray |
| 代理字段 | 不能赋值 |
| 单据状态 | String |
| 维度关联字段 | 不能赋值 |
| 时间 | DateTime |
| 单据类型 | 不能赋值 |
| 大文本 | String |
| XML字段 | String |
| 源单类型 | String |
| 源单编号 | String |
| 基本计量单位 | DynamicObjec或者内码long |
| 公式编辑器 | String |
| 动态文本 | String |
| 多语言XML字段 | String |
| 税组合 | DynamicObjec或者税组合内码long |
| 附件数 | 不能赋值 |
| 业务流程 | DynamicObjec或者流程内码string |
| 红蓝字 | String |
| 移动代理字段 | 不能赋值 |
| 进度文本 | Decimal |

**备注**

推荐使用void SetValue(string key, object value, int row)重载方法，参数更加简单、直接；其他重载方法效果一样，但是参数比较难以准备，或者难以理解；

示例代码暂缺；

## SetCombinedValue

设置组合字段值；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void SetCombinedValue(string combinedKey, string valueKey, object value, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string combinedKey | 组合字段Key |
| string valueKey | 组合字段子项Key |
| object value | 组合字段子项值 |
| int row | 行号 |

**备注**

组合字段，包含了多个子项，根据一定的条件，选择某个子项作为展示项；

赋值时，需要明确指定需要给那个子项赋值。相比其他字段，就复杂了一些，因此，封装了一个单独的方法；

示例代码暂缺；

## SetItemValueByID

设置基础资料内码，为其赋值；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void SetItemValueByID(string key, object id, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 基础资料字段Key |
| object id | 基础资料内码 |
| int row | 行号 |

**备注**

如果不存在此内码的基础资料，或者其他附加条件（数据授权、组织隔离、控制字段限定等）不符合，赋值会失败。

推荐统一使用SetValue方法为基础资料赋值，不推荐本方法；

示例代码暂缺；

## SetItemValueByIDFromClient

对基础资料字段赋值，设置基础资料内码；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void SetItemValueByIDFromClient(string key, object id, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 基础资料字段Key |
| object id | 基础资料内码 |
| int row | 行号 |

**备注**

本方法会略过条件检查，直接取基础资料数据包填写到字段上；

本方法是BOS平台内部为提升性能，专门封装的方法，不推荐插件使用；

示例代码暂缺；

## SetItemValueByNumber

根据基础资料的编码，检索基础资料数据，检索成功后，把成功取得的基础资料数据包，填写到单据上；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void SetItemValueByNumber(string key, string number, int row); | |

**参数说明**

|  |  |
| --- | --- |
| 参数 | 说明 |
| string key | 基础资料字段Key |
| string number | 基础资料编码 |
| int row | 行号 |

**备注**

如果不存在此编码的基础资料，或者其他附加条件（数据授权、组织隔离、控制字段限定等）不符合，赋值会失败。

在仅知道基础资料编码的情况下，推荐使用本方法为基础资料赋值；

示例代码暂缺；

## SummaryDataAndFill

把单据体字段，汇总到单据头字段上；

### 语法

**方法定义**

|  |  |
| --- | --- |
| C# |  |
| void SummaryDataAndFill(Field SummaryField, string TargetFieldName); | |

参数SummaryField为单据体字段；其他实体类型，包括子单据体的字段，未支持；

参数TargetFieldName为单据头字段Key，子单据头字段也支持；

如果SummaryField非单据体字段，本函数直接略过，不计算，也不对目标字段做改动；

示例代码暂缺；

## GetService

方法定义：

|  |  |
| --- | --- |
| C# |  |
| T GetService<T>(); | |

获取注入到表单模型的特殊服务，如填写默认值的服务（IdefaultValueCalculator），数据库访问服务（IDBModelService），函数实现服务（IExprFuncService）等；

这些服务只有在特殊场景下才会用到，插件中，直接使用Kingdee.BOS.ServerHelper中封装的静态类即可；

示例代码暂缺；

## FuncPermissionAuth

方法定义：

|  |  |
| --- | --- |
| C# |  |
| List<Core.Permission.PermissionAuthResult> FuncPermissionAuth(IEnumerable<DynamicObject> datas, string strPermissionItemId, string strFuncName = null, bool bUseFilterScope = true);  List<Core.Permission.PermissionAuthResult> FuncPermissionAuth(string[] ids, string strPermissionItemId, string strFuncName = null, bool bUseFilterScope = true); | |

验证功能权限；

默认情况下，权限项绑定在操作上，菜单点击时，调用操作，自动验证操作绑定的权限项；

某些特殊场景，插件需要调用此方法，自行判断用户是否具有某个权限项；

示例代码，参阅[Context](#_案例_–_验证用户新增权限)章节；

## GetPermissionOrg

方法定义：

|  |  |
| --- | --- |
| C# |  |
| List<long> GetPermissionOrg(string perItem); | |

返回一个组织列表：当前用户，在这些组织下，分配了指定的权限项；

例如，可以通过此方法，查询当前用户可以在那些组织下新增本单据；

示例代码暂缺；

## WriteLog

方法定义：

|  |  |
| --- | --- |
| C# |  |
| void WriteLog(LogObject logObject); | |

写上机日志；

参数logObject为日志详情，关键属性如下：

|  |  |
| --- | --- |
| 属性 | 描述 |
| string pkValue | 单据内码 |
| OperatingEnvironment Environment | 操作场景，枚举值，包括登录、退出、操作等； |
| string SubSystemId | 子系统 |
| string Description | 日志详情 |
| string ObjectTypeId | 业务对象内码，如单据的FormId |
| string OperateName | 操作名称，如”保存”、”删除”、”提交”、”审核”、”反审核” |

### 案例 – 操作执行完毕写日志

**代码来源**

Kingdee.BOS.Core.DynamicForm.AbstractFormOperation

**需求背景**

K/3 Cloud BOS封装的标准操作，在执行完毕后，需要写操作日志，以便追溯；

**实现方案**

封装好日志对象，填好日志内容，调用接口IDynamicFormModel的实例WriteLog方法，写日志；

操作的其他处理代码比较复杂，与本案例无关，为降低阅读难度，本案例仅列出了封装日志对象的部分代码；

**关键字**

日志对象，日志对象属性填写

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| /// <summary>  /// 写上机操作日志  /// </summary>  /// <param name="strDescription">描述</param>  /// <param name="pkValue">业务对象实例内码</param>  public void WriteLog(string strDescription, string pkValue = "")  {  if (this.FormOperation.WriteOperateLog == false) return;  this.View.Model.WriteLog(new Kingdee.BOS.Core.Log.LogObject()  {  pkValue = pkValue,  Description = strDescription,  OperateName = this.FormOperation.OperationName,  ObjectTypeId = this.View.Model.BillBusinessInfo.GetForm().Id,  SubSystemId = this.View.OpenParameter.SubSystemId,  Environment = Log.OperatingEnvironment.BizOperate  });  } | |

## BatchWriteLog

方法定义：

|  |  |
| --- | --- |
| C# |  |
| void BatchWriteLog(List<LogObject> logObjects); | |

批量写用户上机操作日志，需要预先准备好待写的日志对象(LogObject)；

示例代码暂缺；