# 单据维护插件示例

修改记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No** | **日期** | **编制\修订** | **校对** | **批准** | **修改的章节号** |
| V1.0 | 20150421 | 吴亮 | 张晋博 | 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

[liang\_wu@kingdee.com](mailto:liang_wu@kingdee.com)

目录

[单据维护插件示例 1](#_Toc417487241)

[单据维护插件概要说明 3](#_Toc417487242)

[生命周期中事件执行顺序 3](#_Toc417487243)

[1. 新增单据时，打开新增界面依次执行的插件事件如下： 3](#_Toc417487244)

[2. 查看（修改）单据时，打开单据界面依次执行的事件如下： 4](#_Toc417487245)

[公共属性 4](#_Toc417487246)

[View 4](#_Toc417487247)

[语法 4](#_Toc417487248)

[常用属性 5](#_Toc417487249)

[常用方法 5](#_Toc417487250)

[案例 –销售出库单修改单据类型 5](#_Toc417487251)

[事件 7](#_Toc417487252)

[OnBillInitialize 7](#_Toc417487253)

[案例 –批号调整单明细单据体不支持排序 7](#_Toc417487254)

[LoadData 8](#_Toc417487255)

[案例 –加载指定内码的科目表 8](#_Toc417487256)

[AfterLoadData 10](#_Toc417487257)

[案例 –费用项目设置字段可见性 10](#_Toc417487258)

[CopyData 11](#_Toc417487259)

[案例 –税务规则税率/税组合类型默认值 12](#_Toc417487260)

[AfterCopyData 13](#_Toc417487261)

[案例 –采购订单复制后计算数量 14](#_Toc417487262)

[BeforeSave 15](#_Toc417487263)

[案例 –销售订单保存前设置变更信息 15](#_Toc417487264)

[AfterSave 17](#_Toc417487265)

[案例 –采购订单保存后刷新委外订单 18](#_Toc417487266)

[BeforeSetStatus 19](#_Toc417487267)

[AfterSetStatus 19](#_Toc417487268)

[BeforeSubmit 19](#_Toc417487269)

[AfterSubmit 19](#_Toc417487270)

[OnGetConvertRule 20](#_Toc417487271)

[案例 –简单生产入库下推简单生产领料订单 20](#_Toc417487272)

[OnShowConvertOpForm 22](#_Toc417487273)

[案例 –采购收料单列表界面下推筛选掉资产卡片 23](#_Toc417487274)

[OnShowTrackResult 24](#_Toc417487275)

[案例 –生产领料单上查替换物料清单为生产订单 24](#_Toc417487276)

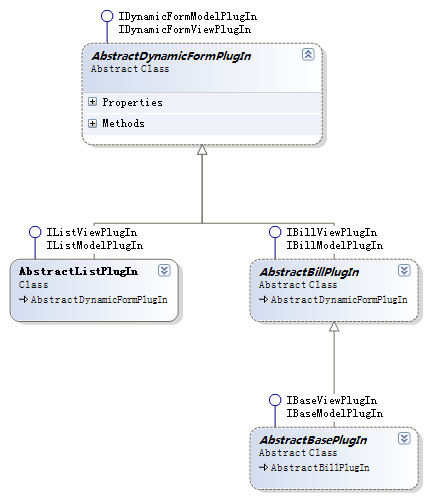
[SaveBillFailed 27](#_Toc417487277)

[VerifyImportData 27](#_Toc417487278)

[案例 27](#_Toc417487279)

# 单据维护插件概要说明

平台插件继承体系图如下



如上图所示，单据维护插件基类AbstractBillPlugIn继承自动态表单基类插件AbstractDynamicFormPlugIn，动态表单插件里面的事件在单据插件中都可以使用。并且实现了IBillViewPlugIn和IBillModelPlugIn两个插件接口，分别是视图层和模型层的插件接口。

# 生命周期中事件执行顺序

## 新增单据时，打开新增界面依次执行的插件事件如下：

1. PreOpenForm
2. OnInitializeService
3. OnSetBusinessInfo
4. OnSetLayoutInfo
5. OnCreateDataBinder
6. OnInitialize
7. OnBillInitialize
8. CreateNewData
9. AfterCreateNewData
10. AfterCreateModelData
11. OnLoad
12. BeforeBindData
13. AfterBindData

## 查看（修改）单据时，打开单据界面依次执行的事件如下：

1. PreOpenForm
2. OnInitializeService
3. OnSetBusinessInfo
4. OnSetLayoutInfo
5. OnCreateDataBinder
6. OnInitialize
7. OnBillInitialize
8. LoadData
9. AfterLoadData
10. OnLoad
11. BeforeBindData
12. AfterBindData

# 公共属性

## View

视图层访问接口，View属性强制转换为IBillView接口的实例对象

### 语法

**属性定义**：

|  |  |
| --- | --- |
| C# |  |
| public new IBillView View  {  get  {  return (IBillView)base.View;  }  } | |

**备注**：

### 常用属性

下表仅列出了IBillView接口中的常用属性，其他IDynamicFormView中的属性请查阅《IDynamicFormView接口使用示例代码.docx》文档

|  |  |
| --- | --- |
| **名称** | **描述** |
| OpenParameter | 单据页面加载时传入的参数 |
| Model | 单据Model层接口 |
| IdList | 当前导航的单据主键集合 |

### 常用方法

下表仅列出了IBillView接口中的常用方法，其他IDynamicFormView中的方法请查阅《IDynamicFormView接口使用示例代码.docx》文档

|  |  |
| --- | --- |
| **名称** | **描述** |
| void CreateNewModelData() | 创建新的模型数据包 |
| NetworkCtrlResult StartNetworkCtrl() | 开启网络控制 |
| bool CommitNetworkCtrl() | 释放网络控制 |

### 案例 –销售出库单修改单据类型

**代码来源**

销售出库单表单插件

Kingdee.K3.SCM.Sal.Business.PlugIn.OutStockEdit,

Kingdee.K3.SCM.Sal.Business.PlugIn

**需求背景**

手工新增的销售出库单,单据类型不允许选择寄售单据类型

**实现方案**

捕获BeforeUpdateValue事件，如果是在修改单据类型，则判断目标值，如果是“寄售单据类型”，则给出提示信息

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.DynamicForm.PlugIn.Args;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.ServiceHelper;  using Kingdee.BOS.Util;  using System;  namespace Kingdee.K3.SCM.Sal.Business.PlugIn  {  /// <summary>  /// 销售出库单表单插件  /// </summary>  public class OutStockEdit : AbstractBillPlugIn  {  public override void BeforeUpdateValue(BeforeUpdateValueEventArgs e)  {  switch (e.Key.ToUpperInvariant())  {  case "FBILLTYPEID":  string newbillTypeId = Convert.ToString(e.Value);  string businessType = "";  DynamicObject billTypePara = BusinessDataServiceHelper.LoadBillTypePara(this.Context, "SAL\_OSBillTypeParaSetting", newbillTypeId);  if (billTypePara != null)  {  businessType = Convert.ToString(billTypePara["BussinessType"]);  }  if (businessType.EqualsIgnoreCase("CONSIGNMENT")  && (this.View.OpenParameter.CreateFrom == CreateFrom.Default  || this.View.OpenParameter.CreateFrom == CreateFrom.Copy))  {  this.View.ShowMessage("手工新增的销售出库单不允许选择寄售单据类型！");  }  break;  //其他case分支略去  }  }  }  } | |

# 事件

## OnBillInitialize

**触发时机**

页面初始化时触发，在OnInitialize事件之后

**应用场景**

初始化时设置控件属性、获取参数

**关键字**

初始化

**备注**

### 案例 –批号调整单明细单据体不支持排序

**代码来源**

批号调整单编辑插件

Kingdee.K3.SCM.Stock.Business.PlugIn.LotAdjustEdit,

Kingdee.K3.SCM.Stock.Business.PlugIn

**需求背景**

批号调整单，明细单据体现实的批号调整的明细信息。由于批号调整是分组的，组内按调整前、调整后排序的，这样比较直观。而如果用户再在单据体上进行分组、排序，会打乱这个显示，显示混乱。所以需要控制用户在前端不允许进行分组排序

**实现方案**

插件在单据初始化的时候，将明细单据体强制所有列为简单列，不支持分组、排序

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  using Kingdee.BOS.Core.DynamicForm.PlugIn.ControlModel;  namespace Kingdee.K3.SCM.Stock.Business.PlugIn  {  /// <summary>  /// 批号调整单单据插件  /// </summary>  public class LotAdjustEdit : AbstractBillPlugIn  {  public override void OnBillInitialize(BillInitializeEventArgs e)  {  this.View.GetControl<EntryGrid>("FSTK\_LOTADJUSTENTRY").SimplizationAllColumns();  }  }  } | |

## LoadData

**触发时机**

开始加载单据数据时触发

**应用场景**

插件干预，加载指定内码的单据，调整加载的单据数据包

**关键字**

加载数据

**备注**

模型层插件事件

### 案例 –加载指定内码的科目表

**代码来源**

科目表编辑插件

Kingdee.K3.BD.Common.Business.PlugIn.AccountTableEdit,

Kingdee.K3.BD.Common.Business.PlugIn

**需求背景**

科目列表界面左侧有一个树形控件，节点层次是科目表，下面子节点是会计要素，再下面是科目类别。在点击树控件上面的【修改】菜单时，需要判断当前所在的节点类型，分别响应节点点击事件。如果是科目表，则对该科目表进行修改。

**实现方案**

科目的列表插件捕获树形控件菜单点击事件，如果当前节点是科目表节点，则打开科目表修改界面，并当前节点的内码值作为自定义参数传递过去；在科目表的表单插件中，捕获LoadData事件，获取自定义参数，加载指定内码的数据。

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Metadata;  using Kingdee.BOS.ServiceHelper;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  namespace Kingdee.K3.BD.Common.Business.PlugIn  {  /// <summary>  /// 科目表编辑插件  /// </summary>  public class AccountTableEdit : AbstractBillPlugIn  {  public override void LoadData(LoadDataEventArgs e)  {  object oId = this.View.OpenParameter.GetCustomParameter("Id");  OperationStatus operationStatus = OperationStatus.ADDNEW;  var openType = this.View.OpenParameter.GetCustomParameter("OperationStatus");  if (openType != null && openType.ToString() == "New")//自定义参数中存在【操作状态】参数  {  operationStatus = OperationStatus.ADDNEW;  }  else if (openType != null && openType.ToString() == "Edit")  {  operationStatus = OperationStatus.EDIT;  }  if (oId != null && !string.IsNullOrEmpty(oId.ToString()) && operationStatus == OperationStatus.EDIT)  {  //自定义参数中存在【单据ID】参数，且单据操作状态是【修改】  e.PrimaryKeyValue = oId;  e.BizDataObject = BusinessDataServiceHelper.LoadSingle(this.Context, oId, this.View.BusinessInfo.GetDynamicObjectType());  }  base.LoadData(e);  }  }  } | |

## AfterLoadData

**触发时机**

单据数据加载完成后触发，此时模型还没有完成初始化

**应用场景**

数据加载完成后，设置字段、菜单的可见性、可用性；少量的字段值调整

**关键字**

数据加载

**备注**

### 案例 –费用项目设置字段可见性

**代码来源**

费用项目编辑插件

Kingdee.K3.BD.Common.Business.PlugIn.ExpenseEdit,

Kingdee.K3.BD.Common.Business.PlugIn

**需求背景**

打开费用项目的时候，需要根据选择的可归属成本类别，设置字段成本项目、成本形态、是否默认的可见性

**实现方案**

表单插件捕获AfterLoadData事件，设置相关字段的可见性

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using System;  namespace Kingdee.K3.BD.Common.Business.PlugIn  {  /// <summary>  /// 费用项目编辑插件  /// </summary>  public class ExpenseEdit : AbstractBillPlugIn  {  public override void AfterLoadData(EventArgs e)  {  var value = this.Model.GetValue("FType");  if (null != value && !string.IsNullOrWhiteSpace(value.ToString()))  {  this.View.StyleManager.SetVisible("FCostPattern", "FCostPattern", true);  this.View.StyleManager.SetVisible("FIsDefault", "FIsDefault", true);  }  else  {  this.View.StyleManager.SetVisible("FCostPattern", "FCostPattern", false);  this.View.StyleManager.SetVisible("FIsDefault", "FIsDefault", false);  }  base.AfterLoadData(e);  }  }  } | |

## CopyData

**触发时机**

单据进行复制操作，在获取到复制源数据之后触发

**应用场景**

插件修改、调整复制的目标数据包

**关键字**

复制

**备注**

当插件中直接给事件参数对象的DataObject属性赋值，返回复制目标数据包且给参数的Cancel属性赋值True时，后续不会触发AfterCopyData事件。

### 案例 –税务规则税率/税组合类型默认值

**代码来源**

税务规则表单插件

Kingdee.K3.BD.Common.Business.PlugIn.TaxRuleEdit,

Kingdee.K3.BD.Common.Business.PlugIn

**需求背景**

在复制税务规则的时候，需要给【税率/税组合类型列表】字段赋缺省值——根据启用税组合确定

**实现方案**

插件捕获CopyData事件，根据启用的税组合确定税率/税组合类型的缺省值，填入到目标数据包

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  using System;  namespace Kingdee.K3.BD.Common.Business.PlugIn  {  /// <summary>  /// 税务规则新增插件  /// </summary>  public class TaxRuleEdit : AbstractBillPlugIn  {  public override void CopyData(CopyDataEventArgs e)  {  base.CopyData(e);  var defaultTaxMixField = this.View.BusinessInfo.GetField("FDEFAULTISTAXMIX");  if (defaultTaxMixField != null)  {  e.DataObject[defaultTaxMixField.PropertyName] = this.TaxMixOrTaxRate;  }  }  private string \_TaxMixOrTaxRate = string.Empty;  /// <summary>  /// 根据启用税组合确定税率/税组合  /// </summary>  private string TaxMixOrTaxRate  {  get  {  if (string.IsNullOrWhiteSpace(this.\_TaxMixOrTaxRate))  {  this.\_TaxMixOrTaxRate = this.IsStartTaxMix() ? "BD\_TAXMIX" : "BD\_TaxRate";  }  return this.\_TaxMixOrTaxRate;  }  }  /// <summary>  /// 税组合是否启用  /// </summary>  /// <returns></returns>  private bool IsStartTaxMix()  {  // 得到启用税组合的状态，组织使用-1  object objStartTaxMix = Kingdee.BOS.ServiceHelper.SystemParameterServiceHelper.GetParamter(this.Context, -1, -1, FormIdConst.TM\_SystemParameter, FormConst.ENABLEDTAXMIX);  return (objStartTaxMix == null) ? false : Convert.ToBoolean(objStartTaxMix);  }  }  } | |

## AfterCopyData

**触发时机**

单据进行复制操作，在复制目标数据包构建完成之后触发。此时单据头、单据体的数据已经复制完成，单据编号值已更新。

**应用场景**

对复制的目标数据包做最后的调整，修改

**关键字**

复制

**备注**

当插件事件CopyData中直接给事件参数对象的DataObject属性赋值，返回赋值目标数据包且给参数的Cancel属性赋值True时，后续不会触发本事件。

### 案例 –采购订单复制后计算数量

**代码来源**

采购订单表单插件

Kingdee.K3.SCM.Purchase.Business.PlugIn.PurchaseOrderEdit,

Kingdee.K3.SCM.Purchase.Business.PlugIn

**需求背景**

采购订单在复制之后，订单明细单据体，需要根据数量和累计收料数量，计算剩余收货数量。还有剩余入库数量和交货明细子单据体上面的几个剩余数量也要进行计算

**实现方案**

插件捕获AfterCopyData事件，计算剩余数量，赋值给目标数据包

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Orm.DataEntity;  using System;  namespace Kingdee.K3.SCM.Purchase.Business.PlugIn  {  public class PurchaseOrderEdit : AbstractBillPlugIn  {  public override void AfterCopyData(BOS.Core.Bill.PlugIn.Args.CopyDataEventArgs e)  {  base.AfterCopyData(e);  foreach (DynamicObject entry in (e.DataObject["POOrderEntry"] as DynamicObjectCollection))  {  //剩余收货数量  entry["RemainReceiveQty"] = Convert.ToDecimal(entry["Qty"]) - Convert.ToDecimal(entry["ReceiveQty"]) + Convert.ToDecimal(entry["MrbQty"]);  //剩余入库数量  entry["RemainStockInQty"] = Convert.ToDecimal(entry["Qty"]) - Convert.ToDecimal(entry["StockInQty"]);  foreach (DynamicObject plandetail in (DynamicObjectCollection)entry["POOrderEntryDeliPlan"])  {  //剩余数量  plandetail["DeliRemainQty"] = Convert.ToDecimal(plandetail["PlanQty"]) - Convert.ToDecimal(plandetail["DeliCommitQty"]);  plandetail["BaseDeliRemainQty"] = Convert.ToDecimal(plandetail["BasePlanQty"]) - Convert.ToDecimal(plandetail["BaseDeliCommitQty"]);  }  }  }  }  } | |

## BeforeSave

**触发时机**

保存操作，调用数据保存接口保存业务数据前触发

**应用场景**

保存前客户端进行数据校验、调整保存的数据包

**关键字**

保存前

**备注**

模型层事件

如果在插件在该事件中取消了保存操作，则保存不会继续进行，且AfterSave事件也不会触发

### 案例 –销售订单保存前设置变更信息

**代码来源**

销售订单表单插件

**需求背景**

销售订单在保存前，如果是变更的销售订单，需要设置变更的相关信息

**实现方案**

捕获BeforeSave事件，判断是否为变更单，如果是，则设置变更的相关信息

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Metadata.EntityElement;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Util;  using System;  namespace Kingdee.K3.SCM.Sal.Business.PlugIn  {  /// <summary>  /// 销售订单 单据维护界面插件  /// </summary>  public class SaleOrderEdit : AbstractBillPlugIn  {  //是否变更标志,在做订单变更的时候会更新为True  private bool isOrderChange = false;  public override void BeforeSave(BeforeSaveEventArgs e)  {  //Kingdee.K3.SCM.Sal.Business.PlugIn.Common.SetSettleBeforeSave(this, "FSettleId", "FCustId");  string reason = Convert.ToString(this.View.Model.GetValue("FChangeReason"));  if (this.isOrderChange)  {  //为了保证版本号的连贯性，把版本号相关赋值，放到客户端校验之前，避免服务端失败和客户端保存失败的标识不一致问题造成版本号出问题  //设置变更相关信息的处理赋值（版本号、变更人、变更日期）（//更新单据体的变更状态为“已变更【C】”）  //SetCurrentModelDataVersionInfo();  if (reason.IsNullOrEmptyOrWhiteSpace())  {  this.View.ShowMessage("变更原因必录！");  e.Cancel = true;  return;  }  decimal billAllAmount;  decimal recAmount;  if (IsBillAllAmountLessThanRecAmount(out recAmount, out billAllAmount))  {  this.View.ShowErrMessage(string.Format("变更后价税合计{0}小于实收金额{1}，不允许变更！", billAllAmount, recAmount));  e.Cancel = true;  return;  }  }  }  /// <summary>  /// 价税合计是否小于实收总额  /// </summary>  /// <returns></returns>  private bool IsBillAllAmountLessThanRecAmount(out decimal recAmount, out decimal billAllAmount)  {  EntryEntity planEntity = this.View.BusinessInfo.GetEntryEntity("FSaleOrderPlan");  DynamicObjectCollection plan = this.View.Model.GetEntityDataObject(planEntity);  recAmount = 0;  foreach (DynamicObject dy in plan)  {  recAmount += Convert.ToDecimal(dy["RecAmount"]);  }  billAllAmount = Convert.ToDecimal(this.View.Model.GetValue("FBillAllAmount"));  //价税合计<0（退货订单退货）时，不做这个校验  return (billAllAmount > 0 && billAllAmount < recAmount) ? true : false;  }  }  } | |

## AfterSave

**触发时机**

保存操作，数据保存到数据库之后触发

**应用场景**

保存完成之后更新界面元素的值、状态

**关键字**

保存后

**备注**

模型层事件

如果插件在BeforeSave事件中取消了保存操作，该事件不会被触发

### 案例 –采购订单保存后刷新委外订单

**代码来源**

采购订单表单插件

Kingdee.K3.SCM.Purchase.Business.PlugIn.PurchaseOrderEdit,

Kingdee.K3.SCM.Purchase.Business.PlugIn

**需求背景**

在委外订单界面打开采购订单，修改保存之后，需要刷新外面的委外订单

**实现方案**

插件捕获AfterSave事件，如果是从委外订单界面打开的采购订单，则刷新父表单视图

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.Bill.PlugIn.Args;  using Kingdee.BOS.Util;  namespace Kingdee.K3.SCM.Purchase.Business.PlugIn  {  /// <summary>  /// 采购订单 单据维护界面插件  /// </summary>  public class PurchaseOrderEdit : AbstractBillPlugIn  {  public override void AfterSave(AfterSaveEventArgs e)  {  base.AfterSave(e);  //应制造强烈要求，在这里加刷新委外订单界面的语句  if (this.View.ParentFormView != null  && this.View.ParentFormView is IBillView  && this.View.ParentFormView.BusinessInfo != null  && this.View.ParentFormView.Model != null  && this.View.ParentFormView.BillBusinessInfo.GetForm().Id.EqualsIgnoreCase("SUB\_SUBREQORDER"))  {  this.View.ParentFormView.Refresh();  }  }  }  } | |

## BeforeSetStatus

已废弃，不会触发

## AfterSetStatus

已废弃，不会触发

## BeforeSubmit

已废弃，不会触发

## AfterSubmit

已废弃，不会触发

## OnGetConvertRule

**触发时机**

选单/下推过程，获取单据转换规则时触发

**应用场景**

弃用操作框里指定的转换规则，通过插件来获取新的单据转换规则

**关键字**

选单、下推、单据转换

**备注**

视图层插件，由于下推操作在列表界面一般也会配置，所以为了保持一致，在列表插件中也同样需要处理

选单/下推过程中，确定单据转换规则

### 案例 –简单生产入库下推简单生产领料订单

**代码来源**

供应链简单生产入库表单插件

Kingdee.K3.SCM.Stock.Business.PlugIn.SP.SpInStockEdit,

Kingdee.K3.SCM.Stock.Business.PlugIn

**需求背景**

简单生产入库在下推简单生产领料单时，需要使用BOM正向展开下推简单生产领料单的转换规则进行下推。

**实现方案**

插件捕获OnGetConvertRule事件，判断下推目标单是简单生产领料，则获取BOM正向展开下推简单生产领料单的转换规则，调整转换规则的相关属性，赋值到事件参数中

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.List.PlugIn.Args;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using Kingdee.BOS.ServiceHelper;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  namespace Kingdee.K3.SCM.Stock.Business.PlugIn.SP  {  [Description("简单生产入库单-表单插件")]  public class SpInStockEdit : AbstractBillPlugIn  {  public override void OnGetConvertRule(GetConvertRuleEventArgs e)  {  base.OnGetConvertRule(e);  if (e.ConvertOperation == FormOperationEnum.Push  && (e.TargetFormId == "SP\_PickMtrl"))  {  ConvertRuleElement rule = ConvertServiceHelper.GetConvertRules(this.Context, "ENG\_BomExpandBill",  e.TargetFormId).FirstOrDefault();  if (rule != null)  {  ConvertFilterPolicyElement filterPolicy = rule.Policies.Where(p => p is ConvertFilterPolicyElement)  .Select(p => p as ConvertFilterPolicyElement).FirstOrDefault();  filterPolicy.AlertMessage = new LocaleValue("1.用户需有入库物料BOM上发料组织下简单生产领料单的新增权限。\r\n2.简单生产入库单领料标识应该为“否”。", this.Context.LogLocale.LCID);  List<DefaultConvertPolicyElement> ruleFieldsMap = rule.Policies.Where(w => w is DefaultConvertPolicyElement).  Select(w => w as DefaultConvertPolicyElement).ToList();  var fRuleMap = ruleFieldsMap.FirstOrDefault();  if (fRuleMap != null)  {  KeyValuePair<string, string>[] fieldMaps = new KeyValuePair<string, string>[]  {  new KeyValuePair<string, string>("FStockOrgId", "FSupplyOrgId"), // 物料清单分录的供应组织  new KeyValuePair<string, string>("FPrdOrgId", "fprdorgid\_reg"),  new KeyValuePair<string, string>("FWorkShopId", "fworkshopid\_reg"),  new KeyValuePair<string, string>("FOwnerTypeId0", "fownertypeid\_reg"),  new KeyValuePair<string, string>("FOwnerId0", "fownerid\_reg"),  new KeyValuePair<string, string>("FOwnerTypeId", "fownertypeid\_reg"),  new KeyValuePair<string, string>("FOwnerId", "fownerid\_reg"),  new KeyValuePair<string, string>("FKeeperId", "FSupplyOrgId"), // 物料清单分录的供应组织  new KeyValuePair<string, string>("FParentOwnerTypeId", "fownertypeid\_reg"),  new KeyValuePair<string, string>("FParentOwnerId", "fownerid\_reg"),  };  foreach (var keyValuePair in fieldMaps)  {  var fieldMapElement = fRuleMap.FieldMaps.FirstOrDefault(w => w.TargetFieldKey == keyValuePair.Key);  if (fieldMapElement != null)  {  fieldMapElement.SourceFieldKey = keyValuePair.Value;  }  }  }  e.Rule = rule;  }  }  }  }  } | |

## OnShowConvertOpForm

**触发时机**

下推过程中，展示可选目标单据

**应用场景**

捕获此事件，调整可选的单据类型

**关键字**

下推

**备注**

视图层插件，由于下推操作在列表界面一般也会配置，所以为了保持一致，在列表插件中也同样需要处理

### 案例 –采购收料单列表界面下推筛选掉资产卡片

**代码来源**

采购收料单列表插件

Kingdee.K3.SCM.Purchase.Business.PlugIn.PurchaseReceiveEdit,

Kingdee.K3.SCM.Purchase.Business.PlugIn

**需求背景**

采购收料单下推过程，可选的单据列表需要筛选掉资产卡片

**实现方案**

表单捕获OnShowConvertOpForm事件，修改事件参数的Bills属性，筛选掉资产卡片

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Core.List.PlugIn.Args;  using Kingdee.BOS.Core.Metadata.ConvertElement;  using System;  using System.Collections.Generic;  using System.Linq;  namespace Kingdee.K3.SCM.Purchase.Business.PlugIn  {  public class PurchaseReceiveEdit : AbstractBillPlugIn  {  public override void OnShowConvertOpForm(ShowConvertOpFormEventArgs e)  {  base.OnShowConvertOpForm(e);  //下推操作，用来筛选掉资产卡片  if (e.ConvertOperation == FormOperationEnum.Push)  {  var bills = e.Bills as List<ConvertBillElement>;  if (bills != null && bills.Count > 0)  {  e.Bills = bills.Where(c => !c.FormID.Equals("FA\_CARD", StringComparison.OrdinalIgnoreCase)).ToList();  }  }  }  }  } | |

## OnShowTrackResult

**触发时机**

上/下查过程，获得上/下查结果

**应用场景**

插件捕获此事件，调整上/下查结果，调整结果显示

**关键字**

上查、下查

**备注**

视图层插件，由于上下查操作在列表界面一般也会配置，所以为了保持一致，在列表插件中也同样需要处理

### 案例 –生产领料单上查替换物料清单为生产订单

**代码来源**

生产领料单列表插件

Kingdee.K3.MFG.PRD.Business.PlugIn.Bill. PickMtrlEdit,

Kingdee.K3.MFG.PRD.Business.PlugIn

**需求背景**

在生产领料单界面，上查的时候，需要将上查结果中的物料清单列表替换成生产订单列表

**实现方案**

生产领料单的表单插件捕获OnShowTrackResult事件，修改上查结果

**示例代码**

|  |  |
| --- | --- |
| C# |  |
| using Kingdee.BOS.BusinessEntity.BillTrack;  using Kingdee.BOS.Core.Bill.PlugIn;  using Kingdee.BOS.Core.DynamicForm;  using Kingdee.BOS.Orm.DataEntity;  using Kingdee.BOS.Util;  using Kingdee.K3.MFG.ServiceHelper.PRD;  using System;  using System.Collections.Generic;  using System.Linq;  namespace Kingdee.K3.MFG.PRD.Business.PlugIn.Bill  {  /// <summary>  /// 生产领料单表单插件  /// </summary>  public class PickMtrlEdit : AbstractBillPlugIn  {  /// <summary>  /// 处理上下查结果  /// </summary>  /// <param name="e"></param>  public override void OnShowTrackResult(BOS.Core.List.PlugIn.Args.ShowTrackResultEventArgs e)  {  base.OnShowTrackResult(e);  switch (e.TrackOperation)  {  case FormOperationEnum.TrackUp: //上查  #region 第三方上查  if (e.TrackResult != null)  {  BillNode trackResult = e.TrackResult as BillNode;  if (!trackResult.FormKey.Equals(e.TargetFormKey, StringComparison.OrdinalIgnoreCase))  {  e.TrackResult = e.TargetFormKey.EqualsIgnoreCase("PRD\_PREPAREMTRL") ?  GetReplacedTrackResultForMoveMtrl(trackResult, e.TargetFormKey) : GetReplacedTrackResult(trackResult, e.TargetFormKey);  }  }  #endregion  break;  }  }  /// <summary>  /// 替换上下查结果,把用料清单列表替换成生产订单列表  /// </summary>  /// <param name="trackResult"></param>  /// <param name="targetFormKey"></param>  /// <returns></returns>  protected BillNode GetReplacedTrackResult(BillNode trackResult, string targetFormKey)  {  if (trackResult.FormKey.Equals("PRD\_PPBOM", StringComparison.OrdinalIgnoreCase)  && targetFormKey.Equals("PRD\_MO", StringComparison.OrdinalIgnoreCase))  {  DynamicObjectCollection moDataEntitys = MOServiceHelper.GetMoEntityIdsByPPBomEntityId(this.View.Context, trackResult.LinkIds.Select(o => Convert.ToInt64(o)).ToList());  trackResult = BillNode.Create("PRD\_MO", "");  trackResult.AddLinkCopyData(moDataEntitys.Select(o => o["FMOENTRYID"].ToString()).ToList());  return trackResult;  }  else  {  return trackResult;  }  }  /// <summary>  /// 替换上下查结果,把用料清单列表替换成生产挪料单列表  /// </summary>  /// <param name="trackResult"></param>  /// <param name="targetFormKey"></param>  /// <returns></returns>  protected BillNode GetReplacedTrackResultForMoveMtrl(BillNode trackResult, string targetFormKey)  {  if (trackResult.FormKey.Equals("PRD\_PPBOM", StringComparison.OrdinalIgnoreCase)  && targetFormKey.Equals("PRD\_PREPAREMTRL", StringComparison.OrdinalIgnoreCase))  {  List<long> moveMtrlEntryIds = MoveMtrlServiceHelper.GetMoEntryIdsByPPBomEntryId(this.View.Context, trackResult.LinkIds.Select(o => Convert.ToInt64(o)).ToList());  trackResult = BillNode.Create("PRD\_PREPAREMTRL", "");  trackResult.AddLinkCopyData(moveMtrlEntryIds.Select(o => o.ToString()).ToList());  return trackResult;  }  else  {  return trackResult;  }  }  }  } | |

## SaveBillFailed

已废弃，不会触发

## VerifyImportData

**触发时机**

引入操作，在读取引入文件将数据组装成关系型数据行之后，开始填充单据数据包之前触发

**应用场景**

对引入数据做校验，如果校验不过，引入过程会终止，并抛出具体的校验信息

**关键字**

引入、校验

**备注**

### 案例

暂缺