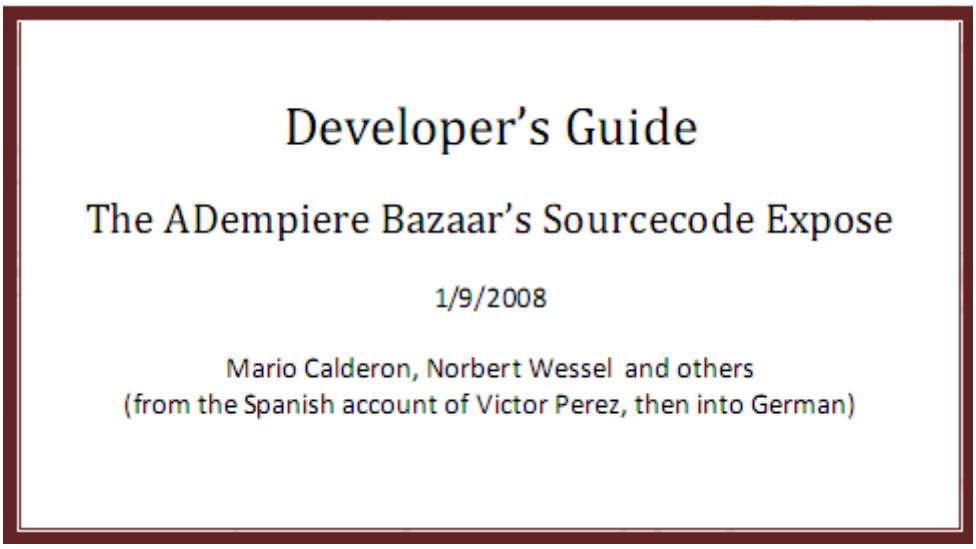
开发者手册

The ADempiere Bazaar's Sourcecode Expose

2/5/2009

Kevin D.J White, Mario Calderon, Norbert Wessel and others
(from the Spanish account of Victor Perez, then into Genman)



内容目录

If you tell the truth you don't have to remember anything.

- Mark Twain

It is not enough just to learn and know, but you ought to possess and own it

- Aristotle

I hope to free my followers from clinging to styles, patterns, or molds

- Bruce Lee, commenting on his Kung Fu

A question may be in the future tense, and so it cannot be answered just yet

- Red1, when asked why things aren't done

ADEMPIERE 开发者文档

前言

本文主要内容是 ADempiere 类是如何组织的，这些类之间又是如何交互从而实现设计功能的。文章安排比较随意，只是尽力做到每一个章节为一个完整的内容，理解上不需要太多其他章节内容的支持。

本文不是完整的单据，只是 ADempiere 的基础单据，我们希望社区里广泛使用本文，欢迎修正与加强。

本文目的是使一个具备 JAVA 基础知识并理解 ADempiere 基础功能的开发人员在读后可以开发、定制自己的系统，并为 ADempiere 的改进做出贡献。

本文写于 2007 年 8 月，在我游览厄瓜多尔后。在那里，Victor Perez 在一次 7 天 ADempiere 核心研究会上告诉我。

在本文中增加了我写的 ADempiere 其他方面的内容，最后成文德语 80 余页，感谢 Metas 协会的 Norbert Wessel 和 Moritz Weber 将本文原稿翻译为英文版，感谢 Karsten Thiemann 的备注，也感谢制作 Doc Wiki Book 的人。

真诚希望本文成为适用于开发人员指导手册的根据。

Mario Calderon

圣萨尔瓦多，2007 年 11 月 26 日

译者序

说到 ADempiere，不得不先说 Compiere，Compiere ERP&CRM 为全球范围内的中小型企业提供综合型解决方案，覆盖从客户管理、供应链到财务管理的全部领域，支持多组织、多币种、多会计模式、多成本计算、多语种、多税制等国际化特性。易于安装、易于实施、易于使用。只需要短短几个小时，您就可以使用申购-采购-发票-付款、报价-订单-发票-收款、产品与定价、资产管理、客户关系、供应商关系、员工关系、经营业绩分析等强大功能了。

Compiere 目前已经不完全开源，官方更新比较慢，ADempiere 从 Compiere 发展而来，目前是 Sourceforge 开源项目组织排名第一的开源项目，设计规范，更新快，是中小型企业 ERP 系统用户的非常好的选择，相对 Compiere 来说，Adempiere 开放了更多的文档，使得开发者更容易上手，相信这也是 ADempiere 为什么受到越来越多人关注的关键之一。

笔者有多年 OA 系统的开发经验，主要是基于 Lotus ND 系统，几年前转入 J2EE 做中小型专业搜索引擎，对 ERP 的研究并不多，此次翻译 ADempiere 系统的部分单据，主要是为了学习。本文档从德文翻译成英文，笔者又从英文译为中文，加之笔者经验不足，译文中不足之处请大家批评指正，

Kevin D.J White

Class org.compiere.util.GenerateModel

包 package org.adempiere.util

public class GenerateModel

Adempiere 持久层主要类是 PO.java，在这个抽象类中定义了 load()，save() ，delete()，get_ID()等方法。

每个持久对象（数据库中的一个表）都是 PO.java 的扩展，这样做是为了让每个持久对象包含全部持久化功能（虽然在有些情况下不需要这样做，但是这样做并没有坏的影响）。针对扩展 PO.java 的 X_*类（定义在应用程序字典 AD 中的所有表）ADempiere 提供一个类来生成这些类的代码，这样做一方面是便于扩展，另一方面 X_*类的大多数方法只是 setter 与 getter，在 ADempiere 3.3.0 前由 org.compiere.util.GenerateModel()实现，在 3.3.0 版本后由 org.adempiere.util.GenerateModel 实现，现在 ADempiere 不只是生成 X_*类的代码，还可以生成 X_*类实现的 I_*接口的代码。要实现全功能（仅持久化方面）的持久 x 对象，ADempiere 提供了这些方法。

在最新的 Adempiere wiki 教程中的有这样一个实例，创建一个表存储原材料信息，如果只需要 store/load 数据到一个 ADempiere 窗口，我们不需要创建一个 M_*类，只是生成一个 X_XX_Material.java 就足够了。给定的实例中，MMater -ial.java 增加了通过给定的 materialno 和 colorno 获取一个 MMaterial x 对象的一些功能，给定一组属性值从数据库中读取 x 对象或所有 x 对象在 M_*类中仅仅是一个最普通的功能。这只是一个例子，我们不需要为我们的 x 对象实现这一类的功能。

在 GenerateModel.java 中

如果不输入或输入错误路径会报错，所以最好是改写 GenerateModel.java

1）缺省路径改为我们用的路径

2）可选-改写包名字

3）可选-在 97 行增加实体类型如“D”（Dictionary 字典）

无参数运行 GenerateModel

- main()方法生成 java beans（X_xxxxxx.java 如 e.g.e.g.X_C_Invoice.Java）。main()方法从 AD_Table 表中读取数据表数据，并为每个表的每个列生成 get-与 set-代码。
- 也可以生成单一 beans
- Adempiere 源代码中这些文件保存在 org.compiere.model 中
- 编译过的 GenerateModel.class 文件保存在../adempiere_trunk/base/build/org/compiere/util
- 可以使用参数配置输出
  - 在 eclipse 中选中 main()方法，点击鼠标右键，点击上下文菜单中的“run”
  - 在弹出的窗口中输入参数
    - 参数 0: 文件夹（Folder）

      生成的文件要拷贝到的位置，缺省值为“C:\\Adempiere\\adempiere-all\\extend\\src\\adempiere\\model\\”

      svn trunk 中为/adempiere_trunk/extend/src/compiere/model//
    - 参数 1: Package 包名字，生成的 java bean 文件的包名字，缺省为“compiere.model”
    - 参数 2: 实体类型（用户自定义，Adempiere，Dictionary 等等），GenerateModel 中只定义了 User 与 Application，象 Adempiere 需要在这里输入，注意: 用户自定义（User-defined）在今后的版本上将保持。缺省值为“'U', 'A'”
    - 参数 3: tabelle，要生成的 java bean 相关的表的名字，缺省为%，表示所有数据库表
- 也可以改写代码来改变参数缺省值
- 当应用程序字典中的数据库表变化时 X_files 一定要重新生成

从版本 3.3.0 开始，/Adempiere/adempiere_trunk/base/src/org/adempiere/util/GenerateModelJPA.java 类来完成上述任务，这个类生成的 JAVA 文件没有 X_

前缀，在部署前必须重命名。

Java-Beans

Bean 特点:

- 都是 POJOs
- 每个业务对象对应一个 java bean
- bean 连接源代码与数据库数据，帮助开发者避免直接访问数据，或者说保证 Table Ids 或列名称信息写死在系统里，如: bp.getM_PriceList_ID()，BusinessPartner 对象返回价格清单的 id
- 都放在 org.compiere.model 包内，这个包放置所有 javabean 类和业务逻辑类
- javabean 类声明

  如:

  public class X_C_Invoice extends PO

  一些方法在 PO（持久化对象定义 org.compiere.model.PO）类中实现如所有继承的构造器调用 javabean 类 PO 的构造器（super(ctx, rs,trxName)或 super(ctx,C_Invoice_ID,trxName)
- *static final* 属性 Table_ID 保存数据库表的 ID，如:

  public static final int Table_ID=MTable.getTable_ID(Table_Name);
- *protected static* 属性 *Model*，如:

  protected static KeyNamePair Model=new KeyNamePair(Table_ID,Table_Name);
- *static final* 属性 accessLevel，

  在数据库表创建时定义的值（Client，Organisation，Client+Organisation,等等）如:

  protected BigDecimal accessLevel=BigDecimal.ValueOf(1);
- AD_ORGTRX_ID_AD_Reference_ID，字典中的 ID
- 方法:
  - initPO

    由调用链上一个 PO 构造器调用（PO(Properties ctx,int ID,

String trxName,ResultSet rs))

- toString（返回类字符串如C_Invoice: X_C_Invoice）

  compare()或log.info(toString())调用

- 数据库表所有列的get-与set-方法

  例外: *processing，processed，Created，CreatedBy*等列在PO类的load方法中创建，load方法调用*loadDefaults()/ setStandradDefaults()*。

  - *getDocStatus()*
  - *setDocStatus()*

- Misc
  - 所有基类和bean都是同一个包的一部分，这样可以通过调用构造器创建一个对象
  - Date日期由Timestamp控制
  - Boolean值是长度为1的字符串（Y=TRUE，N=FALSE），查询布尔值时，没有get方法，请使用象isApproved()的方法
  - 不同的包内还有更多的bean，本文将逐步介绍如MWFActivi -ty，MWFNode，MWFprocess，MDocType，MWorkflow。也有部分javabean并没有与业务对象相关联，如:
    public class X_AD_WF_Activity extends PO
    此类为MWFActivity的帮助类
    public class MWFActivity extends X_AD_WF_Activity implements Runnable
  - Adempiere中Beans和PO一起实现持久化
    - Beans保存数据
    - PO提供持久化机制支持

持久化引擎（Persistency-Engine）

org.compiere.model.PO

Adempiere.PO持久化类提供DB迁移方法同时调用触发器和模型有效性验证（Triggers与model validators）

public abstract class PO implements Serializable, Comparator, Evaluatee

- Serializable

  对象流输出

- Comparator

  PO 实现了 compare()与 equals()方法

- Evaluate

  实现 get_ValueAsString()方法，此方法被 org.compiere.util.Evaluator.evaluateLogicTuple()调用，tupels @xxxx@定义在应用程序字典（AD）中，可评估。

类继承: Adempiere 类如 MInvoice 继承 X_C_Invoice，X_C_Invoice 类继承 PO

- *public class Minvoice extends X_C_Invoice implements DocAction*
- *public class X_C_Invoice extends PO*

重要的构造器:

- 根构造器最后调用

  public PO(Properties ctx,int ID, String trxName,ResultSet rs)

  - 若参数 ID 值为 0，创建一个新的实例。
  - 若事务（transaction）为 null，创建一个新的

    org.compiere.util.Trx 类中的 static 方法控制事务

    - 新事务创建时 Trx 设置一个随机名称
      Trx.createTrxName()
    - 创建一个新的事务
      Trx.createTrxName(“Cost”)
    - 获取当前事务名称
      Trx.get(trxName,True)
    - 若名称为 trxName 的事务不存在且第二个参数为 True，创建一个事务
      Trx.get(trxName,True)
      在 trx.get 中创建一个新的事务
      {
      :
      retValue=new Trx(trxName)
      :
      }

- 设置事务名称保证所有DB保存的同样名称的事务在提交和rollback操作时作为同一个事务来控制

根构造器调用:

- {
  :
  load(int ID, String trxName)
  :
  }
  若ID和事务已知
  - ID>0
    通过SQL语句检索所有列值并写入实例
  - ID<=0
    新对象创建
    私有属性m_createNew获取值True，对象知道这是创建一个新对象
    列processing, processed, Created, CreatedBy等通过调用loadDefaults()创建

  若需要子类中可以定义loadComplete(boolean success)，目前没用
- load(ResultSet rs)
  load(int ID, String trxName)的最后调用load(ResultSet rs)
  这种情况使用一个结果集创建对象，取结果集的当前位置，无导航指针

- 构造器PO(Properties ctx, int ID, String trxName)
  MBPartner bp=new MBPartner(line.getCtx(), line.getC_BPartner_ID(), line.get_TrxName());
- 构造器PO(Properties ctx, ResultSet rs, String trxName)

Po中的其他方法

- public final Object get_Value(String columnName)
  - 检查列是否激活
  - 若列激活，调用get_Value(int index)，该方法返回m_newValues[index]。
    参考Propertiex
- set_ValueNoCheck(String ColumnName, Object Value)
  不经过检查获取列ColumnName值

Value 可以截取，在这种情况下，用户输入到程序中的值会被缩短保存，长度定义在 Adempiere 的 AD（应用程序字典）中

下面的方法只在本章介绍，在继承 PO 的子类中不再说明

- delete()
  删除一个对象，特定情况下，使用 ModelValidationEngine 确认 events
- delete_Tree()
  删除 ID-Trees
- save()
  子类实例保存
  检查是否是新对象，组织
  在 save()中还调用另外的一些方法:
  - beforeSave()
    多数情况下由业务类象 MInvoice, Mproduct 等等实现
  - saveNew()
    保存新对象
  - saveUpdate()
    保存已有的对象

    变化跟踪
    - 创建一个 session 变量
      Msession session=MSession.get(p_ctx,false)
    - 若值变更，数据库表 MChangeLog 保存变更前后的状态:
      MchangeLog cLog=session.changeLog
      Adempiere 中必须做一些配置: 定义一个数据库表保存日志，
      Adempiere 可以在数据库表显示的窗口显示变更日志，双击窗口右下方（右键点击显示数据库的窗口的右下角）
      系统显示如下内容:
      - 数据库表名称
      - 创建表的时间与人员
      - 修改表的时间与人员
      - 变更日志
  - saveNew()与 saveUpdate()最后调用 saveFinish()，saveFinish()调用 afterSave()。

beforeSave()与 afterSave()方法通常在业务类如 MInvoice, Mproduce 等中实现

Po 的属性

- POInfo p_info ( )
  列信息由 p_info 提供
  p_info 值由根构造器提取
  p_info=initPO(ctx)
  子类实现 initPO(), 查看 POInfo 类
  列信息: 调用 get_Value(“Columnname”), 该方法调用 get_Value(int index); 该方法返回 m_newValues[index]。
  （若 m_newValues==null, 则返回 m_oldValues[index] ）
- protected transient Clogger log=CLogger.getCLogger(getClass())
  显示子类在系统控制台输出使用
  private static Clogger s_log=CLogger. getCLogger(PO.class);
  显示 PO 类在系统控制台输出
- private Doc m_doc
- private Object[] m_IDs=new Object[]{I_ZERO}//数据集的 ID
- private Object[] m_oldValues=null;//旧值
- private Object[] m_newValues=null;//新值
- private Mattachment m_attachment=null;
- 等等

类 POInfo

Package org.compiere.model

public class POInfo implements Serializable

POInfo 类提供访问 POInfoColumn 实例的方法

- 包含业务对象列信息
- 可序列化（导出功能用）
- 属性（不完整列表）
  - m_AD_Table_ID
  - m_TableName

- m_AccessLevel
- POInfoColumn[] m_columns
- private Properties m_ctx=null

- 方法（不完整列表）
  使用索引处理m_column属性
  - 构造器
    POInfo(Propertiex ctx, int AD_Table_ID, boolean baseLanguage-Only)调用 loadInfo()，该方法做一些转换:
    - 使用构成数据库表的SQL语句
      - AD_Table t
      - AD_Column c
      - AD_Val_rule vr
      - AD_Element e

      SQL:m_AD_Table_ID的参数
    - 为每个列创建POInfoColumn实例，并用SQL数据填充
      - t.TableName
      - c.ColumnName
      - c.AD_Reference_ID
      - c.IsMandatory
      - c.IsYpdateable
      - c.DefaultValue
      - e.Name, e.Description
      - c.AD_Column_ID
      - c.IsKey
      - c.IsParent
      - c.AD_Reference_Value_ID
      - vr.Code
      - c.FildLength
      - c.ValueMin

        - c.ValueMax
        - c.IsTranslated
        - t.AccessLevel
        - c.ColumnSQL
        - c.IsEncrypted
      - 所有列信息拷贝到m_columns
      - 数据库表AD_Element_TRL用于事务和选择语言
  - public static POInfo getPOInfo(Properties ctx, int AD_Table_ID)调用构造器，getPOInfo()本身在initPO()（在JAVABEAN内部）中被调用，PO构造器调用initPO()。
  - getColumnName(int index)
  - isColumnMandatory(int index)
  - String getColumnDescription(int index)
  - getColumnCount()
  - Class getColumnClass(int index)

类POInfoColumn

package org.compiere.model

public class POInfoColumn implements Serializable

- 包含业务对象一个列的信息
- 属性（公共）
  - AD_Column_ID
  - ColumnName
  - ColumnSQL
  - DisplayType
  - ColumnClass
  - IsMandatory
  - DefaultLogic
  - IsUpdateable
  - ColumnLabel

    - ColumnDescription
    - AD_Reference_Value_ID
    - ValidationCode
    - FieldLength
    - ValueMin
    - ValueMax
    - ValueMin_BD
    - ValueMax_BD
- 方法
    - IsKey
    - IsParent
    - IsTranslated
    - isUpdateable
    - toString
    - IsEncrypted
- POInfoColumn, Po与beans（X_*类）的关系
    - POInfoColumn实例包含业务对象属性信息（列）
    - BEANS（业务对象类）包含数据
    - PO管理数据获取与存储

业务对象类

Package org.compiere.model

Adempiere包含两类业务对象类

- 工作流业务对象类
  业务对象类在工作流（workflow）中调用，因此这些类一定要满足工作流的需求，同时还要满足业务需求
    - 如: MInvoice, Morder
    - 类定义
      public class Minvoice extends X_C_Invoice implements DocAction
      DocAction接口使得类实现realize（识别）一个工作流的方法，这些

方法从按钮触发，按钮定义在窗口里
数据库表与列信息定义在 Adempiere 的 AD（应用程序字典）中，根据 DocAction 的当前状态和下一个状态来调用相关的方法。
更多信息参考工作流章节
下列方法查询状态设置 actions

- prepareIt()
  ModelValidationEngine 进行 Events 的有效性验证
  类 ModelValidationEngine 用于识别常规业务（bussiness）逻辑这里同时捕捉并执行监听 events，定义相关的单据状态和单据类型，并调用其他方法
  常规类定义在客户端窗口（client window），Field 验证类有效性，Libero 中可以找到一个自验证有效性的类的例子。
- CompleteIt()
  通过 ModelValidationEngine，Events 可以被其他 Events 验证
- approveIt()
- 等等

- 公文工作流类定义了属性 private String m_processMsg，该属性在相关窗口的左下方显示信息如 “PeriodClosed”
- 实现实际业务逻辑交互的方法，如在 MInvoice 中:
  - validatePaySchedule()
  - testAllocation()
  - getOpenAmt()
  - 等等

- 业务对象主要（Master）数据，如在 MProduct 中
  - public class Mproduct extends X_M_Product
  - 无实现部分
  - 识别实际业务逻辑的方法，如在 MProduct 中
    - isProductStocked()
    - isOneAssetPerUOM()
    - getAttributeSet()
    - 等
  - 根据类用途来定义类的属性

- 所有业务对象实现触发方法
  - beforeSave()
    数据存储前的动作
  - afterSave()
    数据存储后的动作
  - beforeDelete()
    数据删除前的动作
  - afterDelete()
    数据删除后的动作
- 根据用途不同业务类实现不同的构造器
  如:
  - 标准构造器
    MInvoiceLine (Properties ctx, int C_InvoiceLine_ID, String trxName)
    MInvoice to = new MInvoice (from.getCtx(), 0, null);
  - public MProduct (X_I_Product impP)
    Mproduct 类支持产品导入时使用的构造器，对象 impP 的值用于设置产品的属性。
    类 importProduct 与 X_I_Product 或 ValidateModel 类影响导入功能
- 业务类通常实现静态方法 get()，返回其类对象数组
  如: public static MProduct[] get(Properties ctx, String whereClause, String trxName)
- 业务类中实现一个属性记录日志，输出到系统控制台
  private static Clogger s_log = CLogger.getCLogger (Mproduct.class);
- 业务各实现一个对象 cache
  - private static CCache<Integer,MInvoice> s_cache= new CCache<Integer,MInvoice>("C_Invoice", 20, 2); // 2 minutes
  - cache 中对象数据（20）与生存周期（2 分钟）对每个对象来说都不同
  - org.compiere.util.CacheMgt 通过应用服务器管理 Cache

- 业务逻辑
  下面介绍基于MInvoice.prepareIt()的功能
  步骤:
  - Model验证（如果有）
    参考ModelValidation章节
  - 基于“C_DocTypeTarget_ID”获取DocumentType的一个实例
  - 基于DocBaseType检查是否打开
  - 行验证
  - 检查cashbooks
  - 检查或设置DocType
  - BOM扩展
  - Tax（税务）计算
  - 每行Landed Costs
    - 首先添加所有成本，分布到所有行
    - 在一行中所有成本分配给产品
    - MinvoiceLine allocateLandedCosts()调用MInvoiceLine getBase()，这里没有实现计算LANDEDCOSTDISTRIBUTION_Costs
  - 验证
  - set DocAction

DocAction接口

Package org.compiere.process

public interface DocAction

- 工作流业务对象实现方法定义在DocAction，如: public class MInvoice extends X_C_Invoice implements DocAction. 实现DocAction接口的BO（业务对象）称为Document（单据），最终情况依赖于对象
  - approveIt()

    isApproved属性设置为 true
  - closeIt()
    此操作后没有其他活动

    - invalidateIt()
    - prepareIt()
      执行前准备
    - processIt()
      执行业务逻辑
    - reActivateIt()
      调用 closeIt()后的单据可以通过 reActivateIt()重新激活
      这依赖于对象类型：一个订单可以重新激活，而一份发货清单不可以
    - rejectIt()
    - reverseAccrulIt()
    - reverseCorrectIt()
      在会计模块中生成条目（entries），要实现此功能，需要一份原始单据的拷贝
    - unlockIt()
      没有 lockIt()方法，通常是单据在工作流中出现故障，由系统管理员通过按钮或菜单解锁，将单据的状态退回前一个状态或重新启动单据工作流
    - voidIt()
- 定义静态属性（String 常量）活动与状态（actions 与 states）
    - ACTION_Complete
    - ACTION_Close
    - STATUS_Drafted
    - STATUS_Completed
    - 等
- 各种各样的方法
    - getApprovalamt()
    - getCtx()
      记录的 ID
    - getDocAction()
    - getDocStatus()
      返回定义在 static final Property 中的状态

- getDocumentInfo()
  单据类型名称与单据数量
- getDocumentNo()
- getDoc_User_ID()
  控制序列
  如: Selling- Purchase- Payment-No.
- getprocessMsg ()
  返回属性 m_processMsg 的值
- get_TrxName()
- save()
- setDocStatus ()
  设置单据状态，单据状态定义在 static final Property
- 更多

MDocType 类

Package org.compiere.model

public class MdocType extends X_C_DocType

- 代表一个单据类型
  由 BOs 根据单据类型初始化（在方法 prepare()或 getDocumentInfo()中），这样的 BO 有:
  - MCash
  - MInOut
  - MInventory
  - MInvoice
  - MJournal
  - MMovement
  - MOrder
  - MPayment
  - MPeriod
  - MRequisition
  - 等

- 只有几个方法，没有公共属性
- 方法
  - getOfDocBaseType()
    返回基单据
  - static public MdocType get(Properties ctx,int C_DocType_ID)
    从 cache 中返回一个 MDocType 实例并 instanciates（初始化）一个对象
  - isOffer()
    根据 X_C_DOcType 的属性判断 BO 是否是 Offer.Abstract 的一个子类型
    DOCSUBTYPESO_Proposal.equals(getDOcSubTypeSO())
  - isProposal() isQuotation()与 isOffer()类似

X_C_DocType 类

好象也是由 GenerateModel 生成

package org.compiere.model

public class X_C_DocType extends PO

- 属性
  - public static final int C_DOCTYPEINVOICE_ID_AD_Reference_ID=170; ID_AD_Reference_ID
  - public static final String COLUMNNAME_AD_PrintFormat_ID = "AD_PrintFormat_ID"
  - public static final String COLUMNNAME_C_DocType_ID = "C_DocType_ID"
  - public static final String COLUMNNAME_C_DocTypeInvoice_ID = "C_DocTypeInvoice_ID"
  - public static final String COLUMNNAME_C_DocTypeShipment_ID = "C_DocTypeShipment_ID"
  - public static final String COLUMNNAME_IsDefault = "IsDefault"
  - public static final String DOCSUBTYPESO_Proposal = "ON"
  - public static final int DOCBASETYPE_AD_Reference_ID=183; /** AP Credit Memo = APC */

    - public static final String DOCBASETYPE_APCreditMemo = "APC"; /** AP Invoice = API */
    - public static final String DOCBASETYPE_APInvoice = "API"; /** AP Payment = APP */
    - usw. für Bank Statement (CMB), GL Document(GLD), Material Movement (MMM), GL Journal (GLJ) etc.
- 方法
    - 很多 getter 方法调用 PO 中的方法如:
        - getDocSubTypeSO()调用 PO 中 get_Value(“DocSubTypeSO”)
        - getName 调用 PO 中的 get_Value(“Name”)
    - seeter 方法

单据工作流（公文流转）Documents Workflow

DocumentEngine 类

Package: org.compiere.process

public class DocumentEngine implements DocAction

注意: 业务对象 m_document 称为单据（document）因为它实现了接口 DocAction

DocumentEngine 识别下一个活动，检查（使用 BO 状态）活动是否有效，如果活动有效则执行此业务对象的这个活动。

m_Document 对象在执行活动的进程中发生状态改变，由 DocumentEngine 实现，这种情况下采用一个状态机模型。

DocumentEngine:

- 实现接口 DocAction，可以执行 preParelt()，processIt()，completeIt()等操作
- 初始化后业务对象通过调用 DocumentEngine 中的同名方法改变其状态，如 BO 的 processIt()执行 DocumentEngine 的 processIt()，随后调用 BO 的方法象 complete()
    - 属性
        - DocAction m_document // 业务对象转换为 DocAction DocumentEngine 实现接口 DocAction 同时拥有属性 DocAction

    - String m_status (缺省: STATUS_Drafted, 在DocAction中定义)
    - String m_message
    - String m_action
- get和set方法
    - setDocStatus(String ignored)不做任何事
    - isDrafted()返回STATUS_Drafted.equals(m_status); // STATUS_Drafted, 定义在DocAction
    - isInvalid() 返回 STATUS_Invalid.equals(m_status);
    - isInProgress() 返回 STATUS_InProgress.equals(m_status);
    - isApproved()返回 STATUS_Approved.equals(m_status)
    - 等
- 方法
  处理业务对象（document）的方法
    - 构造器
      DocumentEngine(DocAction po,String docStatus)赋值m_document属性
    - processIt()
    - completeIt()
    - isValidAction(String action)
      使用public String[] getActionOptions()检查活动在BO当前状态是否有效。如:
      当前状态state=STATUS_Drafted, 接下来可能的活动有{ACTION_Prepare, ACTION_Invalidate, ACTION_Complete, ACTION_Unlock, Action_Void}（定义在DocAction）。之后BO的当前状态定义可能的活动。
- 调用方法processIt()触发调用BO中的一个方法
  DocumentEngine的典型应用
  如在MInvoice processIt(String processAction)中:
    - DocumentEngine emgine=new DocumentEngine(this, getDocStatus());由BO当前状态创建一个DocumentEngine的一个实例.
      Return engine.processIt(processAction,getDocAction());
      processAction是工作流预期的活动, getDocAction()活动由用户

指定

- Document: processIt(processAction,docAction)//本身逻辑是基于工作流活动是否有效执行有效性验证，若工作流活动无效，而用户活动有效，m_action赋值，参考 isValidAction()
- 之后调用 processIt(m_action)
- DocumentEngine:processIt(String action)//DocAction 的实现取决于m_Action相关方法调用，在这里
  - 状态变更
  - 调用 BO 的同名方法
    - if (ACTION_Unlock.equals(m_action)) return unlockIt(); 调用 m_document.approveIt()
    - if (ACTION_Invalidate.equals(m_action)) return invalidateIt(); 赋值 m_document.setDocStatus (STATUS_Invalid)
    - if (ACTION_Complete.equals(m_action) ....) completeIt(), 调用 m_document.completeIt() of BO.
    - 等等
    - 执行此方法后赋值 BO 新的状态
    - 某些 DocAction 方法如 DocumentEngine 的 completeIt() 调用 isValidAction (String action) 检查活动有效性
    - 特殊的的活动是 ACTION_Complete, 因为它不再调用 completeIt(),而是调用 m_document.save()和 postIt(). 因此可能会出现这样的情况，在使用 DocumentEngine 时一个活动会进行两次有效性检查。

MWFActivity(Workflow Activity 工作流活动)类

MWorkflow 类

Package org.compiere.wf

public class MWFActivity extends X_AD_WF_Activity implemtents Runnable

public class X_AD_WF_Action extends PO

类没有实现DocAction接口

- 定义
  public class MWFActivity extends X_AD_WF_Activity implements Runnable
  继承X_AD_WF_Activity
  X_AD_WF_Activity类由GenerateModel生成且继承PO
- 属性
  - private m_po //指向执行此活动的BO get函数getPO(Trx trx)
  - m_docStatus
    在run()中取值或执行DocAction的Work()时取值
    - doc=(DocAction)m_po;//取BO实例
      success=doc.processIt(m_node.getDocAction());//执行活动
      setTextMsg(doc.getSummary());
      processMsg=doc.getprocessMsg();
      m_docStatus=doc.getDocStatus();
    - private Trx m_trx
    - private MWFNode m_note //指向引用活动的节点
    - private StateEngine m_statu=null
    - private MWFprocess m_process=null
    - private DocAction m_postImmediate=null
  - 方法
    - 构造器
      - MWFActivity(MWFprocess process, int AD_WF_Node_ID)
        在这里调用接口X_AD_WF_Action中方法的实现
      - 更多构造器
      - 构造器调用方法
        - 服务器Workflowprocessor
        - 客户端WFActivity（可能）
        - 网格VDocAction

    - WebInfo
    - 其他类
- getPO(Trx trx)
  取 BO 的一个引用，这里 MWFActivity 与 MTable 和 PO 交互
- getAD_Table_ID()从 X_AD_WF_Activity 返回对象在 AD_Table 数据库表 AD_Table 中的 ID，X_AD_WF_Activity 调用 PO 方法 get_Value(“AD_Table_ID”)，此函数做值运算
    - 根据 ID 取 BO 对象
    - getRecord_ID()返回 BO 的 ID
      X_AD_WF_Activity 调用 PO 中的方法 get_Value(“Record_ID”)，此函数做值运算
    - 之后取 PO 的引用
- run()
  在 formFrame 类中调用，startBatch()，在任何类型的网格（grid）事件（events)、editors 等之后，startBatch 由 VSetup， actionPerformed()方法依次调用
  调用 performWord()
  在 MWFprocess.startNext()中调用，从按钮触发
- performWork()
  调用:
  success=doc.processIt(m_node.getDocAction())
- getActiveInfo(Properties ctx, int AD_Table_ID, int Record_ID)为组合框控件准备数据
- public void setWFState(String WFState)参考下面的例子
- 例: MinOut.processIt()
  摘要: 工作流的处理程序由服务器激活，服务器负责轮询要执行的所有活动，每个验证有效的活动交由 MWF 处理，MWF 负责程序与触发器所有活动的下一个活动的有效性验证，下一个活动最终调用 processIT()。

  在 documentEngine 我们已经介绍了之后的动作（BO 的 processIt()调用 DocumentEngine 的构造器，之后调用 documentEngine.processIt()等等）

整个进程的调用链如下:
A)

- Adempiere服务器: public void run()
  - 给定时间之后循环调用doWork()
- 工作流处理程序(Workflowprocessor): protected void doWork()
  - 调用Workflowprocessor.wakeup()
- 工作流处理程序(Workflowprocessor) private void wakeup
  - 中断程序，通过一个SQL语句标识所有附带不活跃活动的工作流节点的没有执行的活动
    数据库表: AD_WORKFLOW->AD_WF_NODE->AD_WF_ACTIVITY，由于工作流的启动由一个进程(process)触发，所以，活动可以在节点上跟踪到，节点可以在一个工作流跟踪到，工作流可以在一个进程跟踪到。
  - 对每个活动来说
    activity.setWFState(String WFState)
- MWFActivity:public void setWFState(String WFState)
  - 检查新状态WFState是否有效
  - 若有效，执行Ctx的进程并
  - 调用checkActivities(String trxName)
- MWFprocess:public void checkActivityes(String trxName)
  - 扫描当前活动属于的进程的所有活动
    MWFprocess.getActivities()，此方法中包含SELECT * FROM AD_WF_Activity WHERE AD_WF_process_ID=?
  - 此句没看明白，The first activity after one that is marked as “completed”, is given as parameter to MWFprocess.startNext(). 应该是说当前活动之后的第一个活动状态设置为“completed”，并作为参数传递给MWFprocess.startNext()。
  - 管理其他活动的状态
- MWFprocess:private boolean startNext()

- 上一个活动状态赋值为“processed”，save()
- 取下一个活动
- 执行逻辑“与”或逻辑“异或”操作
- 检查活动是否是链（难道是工作流进程）的下一个
- 开启新的线程启动活动
  new Thread(new WFActivity(...)).start();

- public synchronized void Thread().start()，本段代码中注释：“it calls the run() Method of this thread”，这里调用线程的 run()方法
- MWFActivity:public void run()
  - 调用 performWork()
  - 参数 m_trx
  - 代码注释：“Feedback to process via setWFState -> checkActivities”通过 setWFState 和 checkActivityes 反馈信息
- MWFActivity:private boolean performWork()
  - 取属性 m_node 值，取 action
    String action=m_node.getAction()
  - 可能的活动：Document Action（单据活动），Report（报表），process（进程），Email（电子邮件），变量赋值，用户选择
    系统提供象 task（任务），Sub Workflow（子工作流），User Workbench（用户工作台），User Form（用户表单）和 user Windows（用户窗口）之类的活动，但没有实现
  - 如果当前活动与 Action 相同，调用 doc.processIt(DocAction of node)，参数是 m_node.getDOcAction()

    注意：Actions 是报表（Report）、进程(process)、单据活动(Document Action)等等，而 DocActions 是准备(prepare)、完成(complete)等等
  - 有时会准备“post immediate”

其他关于 setWFState(String WFState)的调用

- MWFActivity: public void run()
  - 调用 MWFActivity.setWFState(String WFState)
  - 之后调用 performWork()
- MWFprocess: setWFState(String WFState)
  - 为进程状态赋值同时刷新所有活动
  - 调用 MWFActivity.setWFState(String WFState)
  - 可能的情况 Status=closed
  - 这里我没有跟踪（作者注）
- InOutGenerate.completeShipment()

MWorkflow 类

Package org.compiere.wf

public class Mworkflow extends X_AD_Workflow

public class X_AD_Workflow extends PO

参考类 processModalDialog，那里 MWorkflow 初始化（instantiated）

- 属性
  - m_nodes: MWFNode 元素数组
  - private static Ccache<String, Mworkflow[]> s_cacheDocValue
- 方法
  - loadNodes()
    根据 Workflow_ID 读取数据库表 AD_WF_Node 中所有节点写入 m_nodes
  - public MWFNode[] getNodes() 返回保存的 m_nodes 中的所有节点到数组 MWFNode[]
  - public MWFNode[] getNextNodes(int AD_WF_Node_ID, int AD_Client_ID) 标识下一个要执行的节点
  - public MWFprocess start(processInfo pi)
    启动一个工作流
    - 创建 MWFprocess 实例
      retValue=new MWFprocess(this, pi)
    - 保存

- 执行本实例的 startWork()
  这里会验证当前工作流的第一个节点是否可以启动，标识第一个节点的活动并执行
  参考类 processModalDialog 中调用 actionButton()的描述
- public int getPrevious(int AD_WF_N_ID, int AD_Client_ID)
  从 m_nodes 中取前一个节点

- afterSave()
  在某些情况下保存所有节点
- 一些 get 方法

ModelValidationEngine 类

Package org.compiere.model
public class ModelValidationEngine

- ModelValidationEngine 类常在业务类中调用（prepareIt()、completeIt()、closeIt()等方法中调用），如下:
  ModelValidationEngine.get().fireDocValidate(this,
  ModelValidator. TIMING_BEFORE_CLOSE);
- get()方法调用（也可以创建）一个 ModelValidationEngine 实例
- get().fireDocValidate(a String)的结果赋值给业务类的属性 m_processMsg
- 属性 s_engine 中包含本类的一个实例，系统中应该只有一个 ModelValida tionEngine 类的实例
- 构造器中初始化 ModelValidator:
  ModelValidator validator=(ModelValidator)class.newInstance();
  这里好象有问题，因为在这里初始化了一个接口，但是接口只能 implemented 不能初始化
  解决方案: 在客户端窗口中可以设置一个客户 Validator 类，这个类包含一个客户端验证，如果不在这里则忽略验证
  例: public class MyValidator implements ModelValidator
  之后调用 Validator 中的 initialize()
- fireDocValidate()
  每个 validator 都调用此方法

- ModelValidator 接口
  package org.compiere.model

public interface ModelValidator
ModelValidator 由开发人员实现

这一段没看懂，应该是在解释 ADempiere 系统与外部的接口，开发人员可以通过接口在系统中写自己的类，实现接口的方法来实现自定义功能，如：登记会计账目或改变登记的逻辑

如果在系统中改变了会计规则，一定要重新 RUN_setup，应用程序服务器才可以识别这些变化

ModelValidator 接口就是用于为系统核心编写外部逻辑的

关于 ModelValidators 的例子参考:
http://adempiere.svn.sourceforge.net/viewvc/adempiere/trunk/extend/src/compiere/model/MyValidator.java?view=markup

- 用于 ModelValidationEngine 构造器内

  ```
  {...
  Class clazz = Class.forName(className);
  ModelValidator validator = (ModelValidator)clazz.new
  Instance();
  initialize(validator, clients[i]);
  ...}
  ```

- 定义一个常量
  public static final int TYPE_BEFORE_NEW=1
  这些常量用于 fireDocValidate 调用时的参数
- 声明方法:
  - public String docValidate(PO po, int timing)
  - public void initialize(ModelValidationEngine engine, Mclient client)
  - public String modelChange(PO po, int type)throws Exception
- 用于实现定制验证
  一个 validation 类的例子: compiere.model.ModelValidator

VdocAction 类

package org.compiere.grid.ed
class VdocAction extends Cdialog implements ActionListener

根据上下文显示单据活动的有效操作

例: 在订单窗口中点击完成按钮，会弹出一个 VDocAction 的表单，在表单中可以选择 DocAction

点击 OK 会调用当前 BO 的选中的 DocAction，此操作引起当前 BO 状态变化

- 控制执行窗口 (Controlls process window)
- 属性
  - GridTab m_mTab
  - m_AD_Table_ID
    - 在构造器中由 Env.getContextAsInt()初始化
    - DocumentEngine.getValidAction()中的 dynInit()作为参数使用
    - private boolean m_OKpressed=false
    - private boolean m_batch=false
  - 图形化元素: panels、组合框、scroll panes、text areas 等等
- 构造器使用如下信息创建一个对话框:
  - Panel
  - BorderLayout
  - ConboBox
  - TextArea
  - Jbutton
  - 等等
- 方法
  - 构造器
    VdocAction(int WindowNo, GridTab mTab, Vbutton button, int Record_ID)调用 jbInit()和 dynInit(Record_ID)等
  - jbInit()
    由构造器调用
    初始化窗口
    为 actionLabel 赋值

- dynInit()由构造器调用
  根据 BO 的状态识别有效活动，（documents)，检测工作流状态:
  wfStatus=MWFActivity.getActiveInfo()

  调用 DocumentEngine.getValidActions()，该方法实现:
  - 根据 DOcState 等内容标识 identify 活动
    in status_NotApproved->Action_Prepare 和 Action_Void
  - 根据数据库表和状态，增加辅助活动，数据库表包括: Order, MinOut(Shipment), Invoice, Payment, GL-journal, Allocation, Bank Statement, Inventory Movement, Physical Inventory。
  - 至少创建缩写为 CO，CL，DR 等的组合框
- actionPerformed()做一些查询
- save()
  通过一个复杂的机制触发，写入数据库
  m_mTab.setValue(“DocAction”,s_Value[index])

菜单操作显示一个窗口

- org.compiere.apps
  AMenuStartItem.run()
  列 Action 取自数据库表 AD_Menu（或 AD_WF_Node，如果不是菜单的话）
  根据 Action 读取相关列: AD_WINDOW_ID, AD_process_ID, AD_WORKFLOW_ID, AD_FORM_ID
  若是窗口，调用 startWindow(0,Window-NO)
  Window-NO 由调用链上传递而来
- org.compiere.apps
  AMenuStartItem.startWindow(int AD_Workbench_ID, int AD_Window_ID)
  用于工作台和普通窗口，若 AD_Workbench_ID==0，计算 AD_Window_ID 的值

  类 AWindow 的实例 frame 调用一个方法，
  frame.initWindow(AD_Window_ID, null)，这一段也有问题，原文如下:
  A method is called by the instance called frame of the class AWindow: frame.initWindow(AD_Window_ID, null)，没写此方法返回什么值，做什么事

第三个参数是可选的查询，final window需要，这里只设置为null，作者没写什么是一个final window

- org.compiere.apps
  AWindow.initWindow (int AD_Window_ID, MQuery query)
  在AWindow的构造器初始化类的变量m_APanel
  在initWindow方法中调用了m_APanel.initPanel(0,AD_Window_ID, query)

  注: 如何在代码任意位置调用一个窗口

```
import org.compiere.model.*;  // because of  MQuery
import org.compiere.apps.*;   // because of   AWindow
:
:
MQuery my_query = new  MQuery ("C_BPartner"); // e.g..
C_BPartner
my_query.setRecordCount(1); // if only one BP is to be shown
my_query.addRestriction("C_BPartner_ID", "=", 1000598);  // BP
ID
AWindow frame = new AWindow();
boolean OK = false;
OK = frame.initWindow(123, my_query); // 123=BP window
frame.validate();
AEnv.showCenterScreen(frame);
```

  若数据库表、限制和窗口没问题，这段代码显示带有新记录的窗口
  问题: MQuery是一个类（这个是个什么问题）

- org.compiere.apps
  APanel.initPanel(int AD_Workbench_ID, int AD_Window_ID, MQuery query)
  这里可以看出，没有实现工作台的选择方案，这只是个窗口，生成标签，可以定义查询:
  query=initialQuery(query, gTab)
  看作者的意义，一个查询QUERY是定义在一个tab里的，没看代码，不知道怎么实现的

- Mquery Apanel.initialQuery(MQuery query, GridTab mTab)
  若MQuery已经存在、活跃且小于10个对象，直接返回query

```
if (query != null && query.isActive() && query.getRecordCount() <
10) return query;
```

  其他情况下则是一个所谓的材积表（volume table）且显示一个过滤对话框


```
Find find = new Find (Env.getFrame(this), m_curWindowNo,
mTab.getName(), mTab.getAD_Table_ID(), mTab.getTableName(),
where.toString(), findFields, 10);
```

上面的代码处理材积表（volume table）

APanel 类

```
package org.compiere.apps
```

```
public final class Apanel extends Cpanel implements DataStatusListener,
ChangeLinstener, ActionListener, Asyncprocess
```

- actionPerformed()
  - 根据窗口的 icon（Save, Print, Attachment, Forward, Backward, 等）的命令调度程序
  - 在私有方法中执行活动
    一些情况下使用 processCtl:process()或当 m_curTab 是一个 grid 时调用 m_curTab.dataSave(manulCmd)
- actionButton()
  - 创建 VDocAction 实例 vda
    显示对话框 vda.setVisible(true)
    在对话框中选择活动（Complete, Void, 等）
  - 调用 processModalDialog dialog=new processModalDialog()
  - 显示最后调用的窗口
    aenv.showCenterWindow(Env.getWindow(m_curWindowNo),dialog)

类 processModalDialog

```
package org.compiere.apps
```

```
public class processModalDialog extends Cdialog  implements
ActionListener
```


摘要: processCtl 管理不同类型的进程，同时初始化启动（instantiation）工作流，工作流 WF 实例化一个 MWF 进程（process），MWFprocess 标识工作流 WF，节点和工作流所有活动，最后 processIt（）执行活动。

接下来的内容主要围绕 actionButton():

- actionPerformed(actionEvent e)
  actionPerformed(actionEvent e)由图形化控件事件触发，调用 processCtl: process()
- processCtl:process(m_ASyncprocess,mWindowNo, parameterPanel, m_pi, null)
  - processCtl: public static 方法 process()用于同步或异步进程
  - m_pi 为类 processInfo
  - 根据 m_pi.AD_process_ID 和 m_pi.Record_ID，获取 MPInstance 的一个实例
  - 在静态方法中实例化一个 processCtl
    processCtl worker=new processCtl(parent, WindowNo, pi, trx)
  - 对异步进程: worker,start()->启动一个新的线程
    - start()调用 new Thread(this).start()
    - 最后调用 processCtl.run()
  - processCtl:public boolean run()
    - 进程 info 由一个复杂的 SQL 查询取得，在数据库表 AD_process 与 AD_PInstance 中取 11 列:
      - 列 1: processname
      - 列 2: Procedurename
      - 列 3: Classname
      - 列 4: AD_process_ID
      - 列 5: isReport
      - 列 6: isDirectPrint
      - 列 7: AD_ReportView_ID
      - 列 8: AD_Workflow_ID
      - 列 9: static case-value
      - 列 10: isServerprocess

        - 列 11: jasperReport
    - 管理进程，工作流，Jasper 报表，报表等，订单、发货清单、出货、项目、付款等进程调用 pi.setPrintPreview(!IsDirectPrint)，最后执行的就是 ReportCtl.start()。不是这些进程系统完成基本报表
    - 如果进程活动是一个工作流，可以这样调用:
      processCtl:startWorkflow(AD_Workflow_ID)
      其他情况参考 processCtl
- processCtl:private boolean startWorkflow(int AD_Workflow_ID)
    - 这是一个远程调用进程，取回服务器连接
    - 另外
      wfprocess=processUtil.startWorkFlow(Env.getCtx(),m_pi,AD_Workflow_ID);
- processUtil:public static MWFprocess startWorkFlow(Properties ctx, processInfo pi, int AD_Workflow_ID)
    - 实例化工作流
      wf=MWorkflow.get(ctx,AD_Workflow_ID)
    - 启动工作流
      wf.start(pi)
      或者推迟启动
      wf.startWait(pi)
- Mworkflow:public MWFprocess start(processInfo pi)
    - 由 processInfo 创建一个 MWFprocess 实例
    - 此实例调用 save()和 startWork()
- MWFprocess:public boolean startWork()
    - 取工作流，取 AD_WF_Node_ID:
      getWorkflow().getAD_WF_Node_ID()
    - 创建 MWFActivity 实例
      new MWFActivity(this, AD_WF_Node_ID)
      这是一个构造器，使用 AD_WF_Node_ID 实例化一个 MWFActivity
    - 活动线程启动
      new Thread(activity).start()
      查阅 MWFAxtivity，了解此线程如何执行: 最后执行相关 BO 的

processIt()

- MWFprocess: private Mworkflow getWorkflow()
  - MWorkflow.get (getCtx(), getAD_Workflow_ID())
- MWorkflow: public static MWorkflow get (Properties ctx, int AD_Workflow_ID)
  - 实例化一个工作流
    new MWorkflow (ctx, AD_Workflow_ID, null)
- MWorkflow: Constructor
  public MWorkflow (Properties ctx, int AD_Workflow_ID, String trxName)
  - 参考类说明
  - 属性赋值

  加载节点:
  loadNodes()

MWFprocess 类

package org.compiere.wf

public class MWFprocess extends X_AD_WF_process

public class X_AD_WF_process extends PO // 实现持久化功能

- 属性
  - private StateEngine m_state = null;
  - private MWFActivity[]m_activities = null;
  - private Mworkflow m_wf = null;
  - private processInfo m_pi = null;
  - private PO m_po = null;
  - private String m_processMsg
- 方法
  - checkActivities()
    启动已经第一个完成的活动的下一个活动
    startNext (activity, activities)
  - public boolean startWork()

这里会验证工作流的第一个节点是否可以启动，检查活动并执行

- public void setAD_WF_Responsible_ID ()
  WF-Responsible Id 赋值

类 Workflowprocessor

package org.compiere.server

public class Workflowprocessor extends AdempiereServer

- 属性
  - private Mworkflowprocessor m_model = null; // model
  - private StringBuffer m_summary = new StringBuffer(); // Last Summary
  - private Mclient m_client = null; // Client-Info
- 方法
  - 构造器
    public Workflowprocessor (MWorkflowprocessor model)
    m_model gesetzt; m_Client is set using the model
  - doWork()
    由 AdempiereServer 调用
    调用 wakeUp().
  - wakeUp()
    - 中断，使用 SQL 查询实例化工作流节点没有执行的活动（with inactive action）
    - 对每一个这样的活动执行:
      activity.setWFState (String WFState)
    - private int sendAlertToResponsible (MWFResponsible responsible, ArrayList<Integer> list, MWFprocess process, String subject, String message, File pdf)
    - private int sendEmail (MWFActivity activity, String AD_Message, boolean toprocess, boolean toSupervisor)

类 MWFNode

org.compiere.wf

public class MWFNode extends X_AD_WF_Node

public class X_AD_WF_Node extends PO // 本类实现持久化功能

- 属性
  - private ArrayList<MWFNodeNext> m_next // 下一个节点
  - private MColumn m_column = null; // 列描述
  - private MWFNodePara[] m_paras = null; // 进程参数
- 方法
  - public String getActionInfo()
    由 toString 调用
    - 取 action
    - 若 action 是一个应用程序进程（Application process），只返回信息
      return "process:AD_process_ID=" + getAD_process_ID()
      - (
        Possible actions (public static final Strings, defined in X_AD_WF_Node)
        - ACTION_UserWorkbench = "B"
        - ACTION_UserChoice = "C";
        - ACTION_DocumentAction = "D";
        - ACTION_SubWorkflow = "F";
        - ACTION_EMail = "M";
        - ACTION_Appsprocess = "P";
        - ACTION_AppsReport = "R";
        - ACTION_AppsTask = "T";
        - ACTION_SetVariable = "V";
        - ACTION_UserWindow = "W";
        - ACTION_UserForm = "X";

- ACTION_WaitSleep = "Z";
)

  这里的值必须与工作流节点的 actions 用的组合框中的值一致

- public MWFNodePara[] getParameters()
  取节点参数
  m_paras = MWFNodePara.getParameters(getCtx(), getAD_WF_Node_ID())
  getAD_Workflow_ID()取 WF-ID using X_AD_WF_Node
- public MWorkflow getWorkflow()
  MWorkflow.get(getCtx(), getAD_Workflow_ID())
- public boolean isUserApproval()
  - 若 Action=ACTION_UserChoice?:
    ACTION_UserChoice.equals(getAction())
  - 不管是不是接受，这是一个查询（It is query, whether it was accepted: ）
    "IsApproved".equals(getColumn().getColumnName())

类 processCtl

package org.compiere.apps (Client-Projekt)

public class processCtl implements Runnable

管理报表与进程

- 属性
  - ASyncprocess m_parent;
  - processInfo m_pi; // Properties and methods for process (transaction-ame, table-ID, AD_process_ID, AD_Workflow_ID, parameter, Record-ID, etc)
  - private Trx m_trx;
  - private Waiting m_waiting;
  - private boolean m_IsServerprocess = false;
- 方法

- run()
  在数据库表 AD_process 中读取 11 列，赋值 m_pi
  - column1: processname
  - column2: Procedurename
  - column3: Classname
  - column4: AD_process_ID
  - column5: isReport
  - column6: isDirectPrint
  - column7: AD_ReportView_
  - column8: AD_Workflow_ID
  - column9: static case-value
  - column10: isServerprocess
  - column11: jasperReport

  根据 m_pi 的不同执行不同的方法，如:
  AD_Workflow_ID 大于 0，启动一个工作流:
  startWorkflow (AD_Workflow_ID)，一个启动 WF 的私有方法:
  processUtil.startWorkFlow(...).

  启动其他:

  - Java-classes
    使用 ProcessCtl 的 run()方法启动，该方法调用 startprocess()，同时调用类 ImportInventory 中的方法 prepare()与 doIt()
    在 AD 的描述文档中有更准确的解释
  - Jasper Reports
    startprocess()
  - normal Reports
    ReportCtl.start()
  - Oracle-DB-Procedures
    startDBprocess()
- 构造器
  public static processCtl process(ASyncprocess parent, int

WindowNo, IprocessParameter parameter, processInfo pi, Trx trx)

- 创建一个MPInstance实例
  MPInstance instance (mithilfe von pi)
- 读参数（save()）
- 同步进程立即执行run()（With synchronized processes the process is executed immediately run() ）

类processUtil

org.adempiere.util

public final class processUtil

只有Logger属性

3个方法

- public static boolean startDatabaseProcedure(processInfo processInfo, String ProcedureName, Trx trx)
  执行过程（Procedure）
- public static boolean startJavaprocess(processInfo pi, Trx trx)
  读取pi中的类名称，取此类，最后将此类实例化为一个进行
- public static MWFprocess startWorkFlow(Properties ctx, processInfo pi, int AD_Workflow_ID)
  - 实例化一个工作流
    wf = MWorkflow.get (ctx, AD_Workflow_ID)
  - 启动工作流: WF is started: wf.start(pi)

发货清单Invoice预览

注: 打印格式可以放在BP（合作伙伴），单据类型（Document Type）和数据库表AD_printform中

发货清单表中的数据预览

AD_TABLE_ID=318 (C_INVOICE)

AD_process_ID: 116 (Rpt C_Invoice)

AD_PRINTFORMAT_ID: 1000071 (Standard Invoice Header). 列 AD_TABLE_ID 引用表 C_Invoice_Header_v，该表代表一个发货清单的打印预览

显示发货清单预览的调用顺序

APanel.actionPerformed()

APanel.cmd_print()

processCtl.process() : AD_TABLE_ID=318 (C_INVOICE), AD_process_ID: 116

processCtl.start()

Thread.start()

:

:

进程执行:

processCtl.run:

ReportCtl.start()

ReportCtl.startDocumentPrint()

ReportEngine.get() (Class method)

ReportEngine-Constructor

ReportEngine.getQuery()

ReportEngine.setPrintData()

DataEngine.getPrintData()

DataEngine.getPrintDataInfo()

回到 getPrintData()，调用 loadPrindData()，该方法执行一个 SQL 查询

回到 ReportEngine.get()，返回 startDocumentPrint()

回到 startDocumentPrint():

ReportCtl.CreateOutput()，显示报表，同时给出选择是直接打印还是打印预览

这一段有问题，看了代码再说

(In the latter case the Code der Viewer takes over:)


```
ReportCtl.preview()
            ReportViewerProvider provider = getReportViewerProvider();
            provider.openViewer(re);
        SwingViewerProvider. openViewer()     -- SwingViewerProvider inherited from
                    ReportViewerProvider
    Viewer-Constructor()
     ReportEngine.getView()
     ReportEngine.layout()
            LayoutEngine-Constructor()
               LayoutEngine.layout()
     Different handling, depending on whether PrintForm is form or table
                  LayoutEngine.layoutForm()
     read configuration of Print Format.
     Position, max. Width, max. Hight, etc. of each Element is identified.
     The local variable element contains Print Format.

  In viewer, the button Print triggers:
  Viewer.actionPerformed ()
    Viewer.cmd_print()
    ReportEngine.print()  -- the print is started
```

上面这段文字可能从德文翻译的时候就没做好，目前看的有点吃力，毕竟我没看过代码，先放着，下面也是在解释同样的内容，“发货清单预览”，只不过更详细。

```
THE BUTTON PRINT PREVIEW IN INVOICE WAS PRESSED HERE
```

点击“打印预览”后的事情:

```
Apanel.actionPerformed(): PrintPreview - 16
Apanel.cmd_print()
  ID=116 [12]
```


AD_TABLE_ID=318 (C_INVOICE 表)

AD_process_ID= 116 (取自 C_Invoice)

processInfo pi:

pi.m_Title=Invoice(Comtomer) SuperUser@Company 名称

pi.m_AD_process_ID=116

pi.m_Record_ID=1004868

processCtl.process()

WindowNo=2 - processInfo[Invoice (Customer) SuperUser@company_name [linux-

jupiter{linux-jupiter-orc1-adempiere}], process_ID=116, Record_ID=1004772,

Error=false,Summary=,Log=0] [12]

(Record_id 为发货清单的 ID) (process_ID=116 ist Rpt C_Invoice)

ReportCtl.start: start()

processInfo[Invoice Print, process_ID=116, AD_PInstance_ID=1004057, Record_ID=1004772,

Error=false, Summary=, Log=[0] [42]

unsynchronized process (异步进程)

AD_PInstance_ID 一直变化

在数据库表 AD_PInstance 中存储了进程 ID (在本例中为 116)和记录 ID (本例中为 1004868), 因此这些值在再装入时不会丢失。

线程执行

processCtl.run()

使用 SQL 查询从 AD_PInstance 读取 AD_Process 的配置:

Name, ProcedureName,ClassName, AD_process_ID, isReport (=Y), isDirectPint (=Y), ReportView-ID (=keine), Workflow-ID (=keine),

isServerprocess(=N)和 JasperReport-Name (none)。这些值赋给类变量 m_pi，Client-和 User-开头的 ID 也已经正确赋值。

之后，检查将要执行的任务（工作流，报表，进程）

因为打印发货清单是一个报表（report），所以调用下一个方法

ReportCtl.start()

Hard-coded dependancy（代码依赖）：根据 process_id，调用方法

对标准报表来说调用：startStandardReport(pi)，该方法创建一个 Report-Engine 实例并调用 CreateOutput(re, pi.isPrintPreview())，该方法转发（forwarded）刚刚创建的 ReportEngine 实例，startStandardReport(pi)立即调用 ReportEngine.get()，

对发货清单（process_ID=116）来说，如下调用:

ReportCtl.startDocumentPrint()

立即调用 ReportEngine.get() .

根据类型(如: Check, Dunning, Remmitance, Project, RfQ,Order/Invoice /Shipment)不同，ReportEngine.get()方法执行一个 SQL 查询。本例中将执行一个发货清单的 SQL 查询，因为 ReportCtl.start()传递过来的类型是 invoice（代码=2）。

关于打印格式的说明: （Print Format）

在 ADempiere 系统中有 3 个地方定义打印格式:

1. 在 BP（Business Partner）中
2. 注册“Client（公司）”时可以设置打印格式
3. 窗口单据类型:

在数据库表 C_DocType 中不仅有 Doc Base Type，还带有打印格式

窗口打印表单（缺省）

数据库表 AD_PRINTFORM 为每个公司（Client）和组织（Organization）预定义打印格式（Print Format）（为发货清单、订单、 发货）以及邮件文本（mailtexts）（为发货清单、订单、发货、项目等等）

在本例中，在数据库表 AD_PRINTFORM 中为客户“MyCustomer”声明发货清单的打印格式 ID=1000071。这个 ID 在表 AD_PRINT_FORMAT 中的“Invoice_Header”，该表的列 ad_table_id 引自数据库表 C_Invoice_Header_v，C_Invoice_Header_v 表示一个发货清单打印预览。

若 BP（商业伙伴）和 DT（Document Type 单据类型）都没有声明，缺省为 AD_PRINTFORM。

说明完毕。

若一个订单需要打印且已经有发货清单，那么这个发货清单将被打印（否则打印订单），在这里将调用 ReportEngine.getDOcumentWhat()，该方法改变发货清单的类型和 Record-ID（这句话好象不对，原文为 It changes the type and Record-ID to Invoice.没看明白改的是谁的类型和记录 ID，要查代码才能知道），其他内容由 ReportEngine.get()处理。

打印所需的单据信息由 SQL 查询获取。

该 SQL 查询做了一个数据库表 c_invoice, ad_printform, ad_client, c_doctype 和 c_bpartner 的并运算（联结 Join），标识 Order（订单），Shipment（发货），Project（项目），Remmitance（这个好象又是德文）和 Invoice（发货清单）的打印格式。

对一个发货清单来说，优先级 1: BP（商业伙伴），2: DocType，3: Print Format(ad_printform)

同时取拷贝数量（不管是从 BP 还是 DT），发货清单的单据 NO 和 BP-ID:

```
SELECT pf.Order_PrintFormat_ID,pf.Shipment_PrintFormat_ID, COALESCE
(bp.Invoice_PrintFormat_ID,  dt.AD_PrintFormat_ID, pf.Invoice_Print
Format_ID), pf.Project_PrintFormat_ID,  pf.Remittance_PrintFormat_ID,
c.IsMultiLingualDocument,   bp.AD_Language,   COALESCE(dt.Document
Copies,0)+COALESCE(bp.DocumentCopies,1),dt.AD_PrintFormat_ID,bp.C_BPartn
er_ID,d.DocumentNo  FROM C_Invoice d  INNER JOIN AD_Client c ON
(d.AD_Client_ID=c.AD_Client_ID)  INNER JOIN AD_PrintForm pf ON
(c.AD_Client_ID=pf.AD_Client_ID) INNER JOIN C_BPartner bp ON
(d.C_BPartner_ID=bp.C_BPartner_ID)  LEFT OUTER JOIN C_DocType dt ON
(d.C_DocType_ID=dt.C_DocType_ID) WHERE d.C_Invoice_ID=? AND pf.AD_Org_ID
IN (0,d.AD_Org_ID)  ORDER BY pf.AD_Org_ID DESC
```

本例中上述 SQL 的结果:

```
#  ORDER_PRINTFORMAT_ID    1000069
#  SHIPMENT_PRINTFORMAT_ID    1000073
```



```
#  PRINTFORMAT_ID        1000100
#  PROJECT_PRINTFORMAT_ID
#  REMITTANCE_PRINTFORMAT_ID  1000077
#  ISMULTILINGUALDOCUMENT     Y
#  AD_LANGUAGE
#  DOCUMENTCOPIES       2
#  AD_PRINTFORMAT_ID
#  C_BPARTNER_ID         1000317
#  DOCUMENTNO          VCF_633487_2007
```

Ad_printformat_id, c_bpartner_id, beleg-No 和 number of copies (若为空设置为 1) 从 SQL 查询中获取并存储在变量中。这只是关于打印的信息，而不是可以打印的数据。

创建类 MPrintFormat, Mquery 和 PrintInfo 的实例，构造器赋值局部变量 query，在本例中是 “C_Invoice_Header_v.C_Invoice_ID=1004868”，或者直接使用发货清单类型（结果: 发货清单中的 C_Invoice_Header_v）和单据类型 + “-ID”（结果: “C_Invoice_ID=1004868”）来创建。

DOC_TABLES 说明

局部变量 query 使用 DOC_TABLES 查找数据库表的方法:

DOC_TABLES[type]

定义变量 TABLES，查找数据库表名称

```
DOC_TABLES = new String[]
 {"C_Order_Header_v",  "M_InOut_Header_v", "C_Invoice_Header_v",
 "C_Project_Header_v","C_RfQResponse_v","C_PaySelection_Check_v",
 "C_PaySelection_Check_v",  "C_DunningRunEntry_v" };
```

因此，如果 type==2（发货清单，则结果表将会是 C_Invoice_Header_v）

说明结束

最后在 ReportEngine.get() 中实例化一个 ReportEngine，类变量赋值给 MPrintFormat, Mquery 和 PrintInfo 的实例（哪个类的哪些类变量没有说明）。返回 ReportEngine 实例。

当调用 ReportEngine 的构造器时:

例:

ReportEngine.<init>(Constructor):

MPrintFormat[ID=1000071,Name=Invoice_Header,Language=Language=[Español (ElSalvador), Locale=es_SV, AD_Language=es_SV, DatePattern=DD/MM/YYYY, DecimalPoint=true],Ite ms=56] —C_Invoice_Header_v.C_Invoice_ID=1004772 [42]

ReportEngine.getQuery()

ReportEngine.setPrintData().

实例化一个 DataEngine 调用 getPrintData()。

DataEngine.getPrintData()

因为不在报表视图，先要识别数据库表的名称

(SELECT TableName FROM AD_Table WHERE AD_Table_ID=516;):

在 AD 中是数据库表 C_Invoice_Header_v

这里数据库表名称变为 C_Invoice_Header_vt，注: 在 AD 中没有数据库表 C_Invoice_Header_vt，只有 C_Invoice_Header_v。

查询变为: C_Invoice_Header_vt.C_Invoice_ID=1004868

调用 getPrintDataInfo()（一个巨大的方法）

接下来使用另一个巨大的方法 loadPrintData()取数据。

下面先来看看 getPrintDataInfo()

DataEngine.getPrintDataInfo()

在变量 final SQL 中创建一个取数据的 SQL 查询，The columns for the needed SQL are taken among others from PrintFormat. 这一句的意思应该是从 PrintFormat 中取 SQL 查询需要的列。

getPrintDataInfo()调用参数如下:

报表名称: Invoice_Header

TableName=C_Invoice_Header_vt

Query=C_Invoice_Header_vt.C_Invoice_ID=1004772 AND C_Invoice_Header _vt.AD_Language='es_SV'

本方法中要创建一个包含 WHERE 条件的 SQL

Format=MPrintFormat[ID=1000071, Name=Invoice_Header, Language =[Español (El Salvador),Locale=es_SV,AD_Language=es_SV,DatePattern=DD/MM/YYYY, DecimalPoint=true],Items=56]

Column order AD_Column_ID=7483

执行参数为 ad_printformat_id 的 SQL（从变量 Format 中取），取列 Print Format, PrintFormatItem, AD_Column 等，因此 SQL 象下面的样子:

```
String sql = "SELECT c.AD_Column_ID,c.ColumnName,"        //  1..2
   + "c.AD_Reference_ID,c.AD_Reference_Value_ID,"        //  3..4
   + "c.FieldLength,c.IsMandatory,c.IsKey,c.IsParent,"      //  5..8
   + "COALESCE(rvc.IsGroupFunction,'N'),rvc.FunctionColumn,"  //  9..10
   + "pfi.IsGroupBy,pfi.IsSummarized,pfi.IsAveraged,pfi.IsCounted, " //
11..14
   + "pfi.IsPrinted,pfi.SortNo,pfi.IsPageBreak, "   //  15..17



 + "pfi.IsMinCalc,pfi.IsMaxCalc, "          //  18..19
 + "pfi.isRunningTotal,pfi.RunningTotalLines, "        //  20..21
 + "pfi.IsVarianceCalc, pfi.IsDeviationCalc, "        //  22..23
 + "c.ColumnSQL "              //  24
 + "FROM AD_PrintFormat pf"
 + " INNER JOIN AD_PrintFormatItem pfi ON
 (pf.AD_PrintFormat_ID=pfi.AD_PrintFormat_ID)"
 + " INNER JOIN AD_Column c ON (pfi.AD_Column_ID=c.AD_Column_ID)"
```



```
 + " LEFT OUTER JOIN AD_ReportView_Col rvc ON
 (pf.AD_ReportView_ID=rvc.AD_ReportView_ID AND
 c.AD_Column_ID=rvc.AD_Column_ID) "
 + "WHERE pf.AD_PrintFormat_ID=?"          //  #1
+ " AND pfi.IsActive='Y' AND (pfi.IsPrinted='Y' OR c.IsKey='Y' OR
pfi.SortNo > 0) "
 + "ORDER BY pfi.IsPrinted DESC, pfi.SeqNo";
```

SQL 执行的结果是定义在 AD 中的列，如下:

```
 AD_COLUMN_ID   COLUMNNAME  AD_REFERENCE_ID (Im Code Display Type)
 7652       BPValue       10 (Text)

 7466       C_Order_ID       30 (Search)
 7463       DateInvoiced       15 (Date)
 7475       Name         10 (Text)
 7448       C_Location_ID     21 (Location (Address) )
 7586       PaymentTerm      10 (Text)
 7460       M_PriceList_ID     19 (Table Direct)
 7483       DocumentNo       10 (Text)
  其他列: AD_REFERENCE_VALUE_ID, FIELDLENGTH, ISMANDATORY,
  ISKEY,    ISPARENT, COALESCE(RVC.ISGROUPFUNCTION,'N'),
FUNCTIONCOLUMN,    ISGROUPBY, ISSUMMARIZED, ISAVERAGED, ISCOUNTED,
ISPRINTED, SORTNO,    ISPAGEBREAK, ISMINCALC, ISMAXCALC, ISRUNNINGTOTAL,
  RUNNINGTOTALLINES, ISVARIANCECALC, ISDEVIATIONCALC, COLUMNSQL
```

Print Format 中的排序列和所有 SQL 中标记为 KEY 的列都包含除了这些选中列中（标记为 “no printed” ）

关键: 列配置与其在 C_Invoice_Header_v 中的定义相关，而不是其原始表格。这意味着列 C_Order_id 不是表 C_Invoice_Header_v 中的 KEY，而是表 C_Order 的。

另一方面，这些行在一个循环中分析，并逐行保存在一个PrintDataColumn实例中(AD_Column_ID, ColumnName, AD_Reference_ID, FieldLength, orderName, isPageBreak)。

这些信息用于生成提取打印数据的SQL语句，该SQL语句在getPrintDataInfo()中创建，保存在变量final SQL中，如下:

```
SELECT
 (SELECT C_Order.DocumentNo||' - '||TRIM( TO_CHAR( C_Order.DateOrdered,
 'DD/MM/YYYY'))
 FROM C_Order
 WHERE C_Invoice_Header_vt.C_Order_ID=C_Order.C_Order_ID) AS
AC_Order_ID,
 // This was the first column, called AC_Order_ID
 C_Invoice_Header_vt.C_Order_ID,
 // The last columns are just displayed instead of c_order_id; see
explanation beneath
 C_Invoice_Header_vt.DateInvoiced,
 C_Invoice_Header_vt.Name,
 B.City||'.' AS Baddress,       // see explanation
 C_Invoice_Header_vt.C_Location_ID,  // see explanation
 C_Invoice_Header_vt.PaymentTerm,
       // for invoice, see explanation
 (SELECT M_PriceList.Name FROM M_PriceList
 WHERE C_Invoice_Header_vt.M_PriceList_ID=M_PriceList.M_PriceList_ID)
   AS CM_PriceList_ID,
 // This is another column, called   CM_PriceList_ID
 C_Invoice_Header_vt.M_PriceList_ID
 FROM C_Invoice_Header_vt
 LEFT OUTER JOIN C_Location B ON
   (C_Invoice_Header_vt.C_Location_ID=B.C_Location_ID)
```



```
        WHERE C_Invoice_Header_vt.C_Invoice_ID=1006390
        AND C_Invoice_Header_vt.AD_Language='es_SV'
        AND C_Invoice_Header_vt.AD_Client_ID IN (0,1000001)
        AND B.C_Location_ID NOT IN
   ( SELECT Record_ID FROM AD_Private_Access WHERE AD_Table_ID = 162 AND
 AD_User_ID <> 100 AND IsActive = 'Y' )
 ORDER BY C_Invoice_Header_vt.DocumentNo
```

创建并返回有列描述的一个 PrintData 实例、final SQL 和数据库表 (C_Invoice_Header_vt)

说明:

如果在 PrintFormat 中有 IDs，显示哪一个列?

A）例: 发货清单打印中的 C_ORDER_ID

每个 AD 中描述的列都有一个域 isIdentifier，若 ID 域存在，isIdentifer 域 = “Y”，这样，isIdentifier 作为一个整个对象的信息源使用，因为只有 ID 域不够用。（这一段应该是说域 isIdentifier 是域 ID 的辅助）

在 getPrintDataInfo()中，将会在列分析中查询当前列是否是一个搜索域（ad_reference_id=30），若是，做如下调用: MlookupFactory.getLookup_TableDirEmbed(), 该方法调用一个 SQL 查询:

```
SELECT
c.ColumnName,c.IsTranslated,c.AD_Reference_ID,c.AD_Reference_Value_ID
 FROM AD_Table t INNER JOIN AD_Column c ON (t.AD_Table_ID=c.AD_Table_ID)
 AND c.IsIdentifier='Y'  WHERE TableName='C_Invoice' // 或相关数据库表
 ORDER BY c.SeqNo;
```

在有 isIdentifier 的情况下 build 一个象 AC_Order_ID 的列，在 C_ORDER 中，列 DocumentNo 和列 DateOrdered 标识为 identifier （ isIdentifier =” Y”），输出出下:

```
SELECT C_Order.DocumentNo||' -
 '||TRIM( TO_CHAR( C_Order.DateOrdered, 'DD/MM/YYYY'))
 FROM C_Order
 WHERE C_Invoice_Header_vt.C_Order_ID=C_Order.C_Order_ID.
```


接着创建普通 SQL 语句 C_Invoice_Header_vt.C_Order_ID.

B）在 C_Invoice 中，列将会是 DocumentNo, DateInvoiced 和 GrandTotal。SQL 语句为:

```
SELECT
  C_Invoice.DocumentNo||' -
  '||TRIM(TO_CHAR(C_Invoice.DateInvoiced,'DD/MM/YYYY'))||' -
  '||TRIM(TO_CHAR(C_Invoice.GrandTotal,'999G999G999G990D00'))
  FROM C_Invoice
  WHERE C_Invoice_Header_vt.C_Invoice_ID=C_Invoice.C_Invoice_ID
```

接着创建普通 SQL 语句 C_Invoice_Header_vt.C_Invoice_ID。

C）若列的引用类型为 Location，程序中的 SQL 中返回 B.City||'.' AS Baddress（这个写法应该不对），接着创建普通 SQL 语句:

C_Invoice_Header_vt.C_Location_ID。引用类型为 Account, Locator oder Pattribute 时的情况类似。

说明结束

DataEngine.getPrintDataInfo()的结尾

回到 DataEngine.getPrintData()，立即调用 loadPrintData()，执行 SQL 语句。

DataEngine.loadPrintData()

SQL 的结果为单行或多行，单行的例子如下:

```
#  BPVALUE       0191 (Key)
#  AC_ORDER_ID    701517
#  C_ORDER_ID    1004302
#  DATEINVOICED    22-AUG-07
#  NAME       NAME OF A COMPANY
#  NAME2       FARMACIA DEL PUEBLO
#  DUNS        126615-2
#  BADDRESS       SANTA ROSA DE LIMA.
#  C_LOCATION_ID    1000300
#  TAXID
```



```
#  DESCRIPTION    COMPRA/VENTA DE MEDICINAS
#  SALESREP_NAME    TORRES CARBALLO, RIGOBERTO
#  PAYMENTTERMNOTE  30 DIAS
#  CC_INVOICE_ID    VCF_633487_2007 - 22/08/2007 - 302.98
#  C_INVOICE_ID    1005618
#  TOTALLINES     268.12
```

在 loadPrintData() 中，所有行数据在一个大循环中读取，列分析在一个小的循环中做。

变量 rowNo 计数数据行，counter 指向下一个列的引用

小循环做列分析:

- 变量 pdc（PrintDataColumn）存储可编辑的列信息（using indexing of the input parameter pd）
  - m_ad_Column_id 是列在 ad_column 中的定义，存储在数据库表 C_Invoice_Header_v 中
  - m_alias 是列在 SQL 语句中的名称，如 AC_ORDER_ID
    别名只用于列是通过连接计算出的时候，根据 ID-identifier 合成
    使用别名可以立即取下一个列，因为在 SQL 中它们是同一类的（参考 getPrintDataInfo() 中的 SQL）
  - m_column_name: 是可以在 Print Format 中找到的列，如 C_ORDER_ID
  - 实际的列是根据它是否是一个 ID 和哪一个域做为一个 identifier 来决定
- 管理 Boolena 值，长文本，日期时间值，字符串与数字
- 小循环的最后，创建一个 pde（PrintDataElement）对象，该对象包含列名称、内容、是否包含 KEY 或页间隔，pde 对象加入到 pd 赋值给变量 m_group
  - m_column_name: 列名称，存储在 Printformat 中，如 C_ORDER_ID
  - m_display_type: = AD_REFERENCE_ID (参考前面的内容)
  - m_value 包含连接列的内容（或者只是 m_value）
    - m_key 如: 1000300
      相关数据库表的 ID，对 c_invoice_id 来说，数据表为 c_invoice，对 c_location_id，数据表为 c_location

- m_value，如: “SANTA ROSA DE LIMA..”

在 loadPrintData()中，行计数就象是节目表。以带注释 “ //Add Total Lines ” 的行开始。计算功能如 Count, Mean, Sum, Minimum, Maximum, Derivation 等以 “// Check last Group Change ” 开头，同时调用 m_group.get Value()

PrintDataGroup.getValue ():

PrintDataFunction.getValue(char function).

回到 ReportEngine.get()中，返回 ReportEngine 的 startDocumentPrint()。

startDocumentPrint(): CallCreateOutput()，根据报表类型是直接打印还是打印预览显示报表。

后来的情况，viewer 使用下面的代码显示:

ReportViewerProvider provider = getReportViewerProvider();

provider.openViewer(re)

provider.openViewer(re) 使用 re.getView()创建布局，变量 re 是类 ReportEngine 的实例包含取数据的 SQL 语句。

执行多个方法如 LayoutEngine.layout(), LayoutEngine.layoutForm().

说明:

LayoutEngine

LayoutEngine 的 3 个方法对打印来说很重要:

- layout()
  在 layoutEngine 的构造器中，提交用于其他类取数据的参数 Data 并保存 m_data，m_data 包含 m_sql，用于数据提取。
  Aging 报表的 SQL 语句:
  SELECT
  T_Aging.DueAmt,
  A.Value||'-'||A.Name AS Aname,
  T_Aging.SalesRep_ID,
  (SELECT C_SalesRegion.Name FROM C_SalesRegion WHERE

```
T_Aging.C_SalesRegion_ID=C_SalesRegion.C_SalesRegion_ID) AS
BC_SalesRegion_ID,
T_Aging.C_SalesRegion_ID,
T_Aging.BPValue,
T_Aging.SalesRep_Name,
T_Aging.SalesRegionName,
T_Aging.DateInvoiced,
(SELECT C_BPartner_Location.Name FROM C_BPartner_Location WHERE
T_Aging.C_BPartner_Location_ID=C_BPartner_Location.C_BPartner_Locat
ion_ID) AS
CC_BPartner_Location_ID,
T_Aging.C_BPartner_Location_ID,
T_Aging.LocationName,
(SELECT C_BPartner.Value||' - '||C_BPartner.Name FROM C_BPartner
WHERE
T_Aging.C_BPartner_ID=C_BPartner.C_BPartner_ID) AS DC_BPartner_ID,
T_Aging.C_BPartner_ID,
SELECT C_BP_Group.Name FROM C_BP_Group WHERE
T_Aging.C_BP_Group_ID=C_BP_Group.C_BP_Group_ID) AS EC_BP_Group_ID,
T_Aging.C_BP_Group_ID

 FROM T_Aging

LEFT OUTER JOIN AD_User A ON (T_Aging.SalesRep_ID=A.AD_User_ID)
WHERE
T_Aging.AD_PInstance_ID=1017227
AND T_Aging.C_BPartner_ID=1000598.0
AND T_Aging.IsListInvoices='N'
AND T_Aging.IsSOTrx='Y'
AND T_Aging.AD_Client_ID IN (0,1000001)
AND A.AD_User_ID NOT IN ( SELECT Record_ID FROM AD_Private_Access
WHERE
AD_Table_ID = 114 AND AD_User_ID <> 100 AND IsActive = 'Y' )
```

若条件是 C_BPartner_ID=1000598.0, 返回正确值

- layoutForm()
- layoutTable()

说明结束

调用:

LayoutEngine.layout()

LayoutEngine.layoutForm(),

LayoutEngine.includeFormat()

DataEngine.getPrintData() (参考上面的内容)

变量 PrintData includedData 包含打印数据

调用 LayoutEngine.layoutTable()

数据分析

若是域，在 LayoutEngine.layoutForm()调用 LayoutEngine.createFieldElement()

LayoutEngine.createFieldElement()

该方法生成输出:

- Conversion into string 转换为字符串
- Display of ID elements 显示 ID 元素
- Numbers are called as words at output:输出中数字做为 words 调用，在这里数字与语言应该是做了一个接口
  Msg.getAmtInWords (m_format.getLanguage(), stringContent) ，根据所选语言调用，对 Spanish 来说调用方法 AmtInWords_ES.getAmtInWords()
- Color 颜色

这一章终于看完了，看得不是很仔细，因为与代码结合的太紧，只有在看过代码后才能理解有些内容，先放着。

PrintFormat

数据表 AD_PRINT_FORMAT

- 包含所有条目填充同名的窗口（contains all entries to fill the equal-named window ）

类 PrintData

包含实际要打印的数据，在这里只解释类变量

类变量:

- PrintDataColumn[] m_column_info
  元素数组，描述要打印的列，PrintDataColumn 只有 get 方法，构造器和类变量:
  - private int m_AD_Column_ID;
  - private String m_columnName;
  - private int m_displayType;
  - private int m_columnSize;
  - private String m_alias;
  - private boolean m_pageBreak;
- Properties m_ctx -- Context 上下文
- String m_name: -- Name of the print format 打印格式名称
- ArrayList<Object> m_nodes -- PrintDataElement 数组
  PrintDataElement 的类变量
  - private String m_columnName;
  - private int m_displayType;
  - private boolean m_isPageBreak;
  - private boolean m_isPKey;
  - m_value;
  - ArrayList<ArrayList<Object>> mrows -- PrintDataElement 数组
    Contains the real data as rows and columns.包含行与列的实际数据
  - m_sql
    SQL 语句，提取数据用:
  - SELECT
    C_Invoice_LineTax_vt.ProductValue,


```
C_Invoice_LineTax_vt.QtyEntered,
C_Invoice_LineTax_vt.Name,
C_Invoice_LineTax_vt.Discount,
C_Invoice_LineTax_vt.PriceEntered,
C_Invoice_LineTax_vt.LineNetAmt,
C_Invoice_LineTax_vt.C_InvoiceLine_ID – was automaticly
fetched
FROM C_Invoice_LineTax_vt
WHERE C_Invoice_LineTax_vt.C_Invoice_ID=1005618 AND
C_Invoice_LineTax_vt.AD_Language='es_SV' AND
C_Invoice_LineTax_vt.AD_Client_ID IN (0,1000001)
```

- m_TableName,如: C_Invoice_LineTax_vt

Ad 中的工作流（WF）

WF 类型判定: （组合框控件是如何设计的？作者按）

下面这段内容是主界面的菜单操作，不翻译，可以在 sourceforge 的 Adempiere video 中找到相关资料

- Login as System Admin
- Menu, Window Workflow
- Zoom at field Window
- In Window, Tab & Field with name Workflow, Tab Workflow, Field Workflow Type zoom to column WorkflowType.
- Table&Column with name AD_Workflow to Tab Column, zoom at Field Reference Key (is AD_Workflow Type)
- Select Reference with namen AD_Workflow Type, Tab List Validation in window. values General, Document process and Document Value can be found there.

ADempiere 系统中有三类 WF 类型

- General（普通进程）G
  普通用户使用的 WFs
  - Accounting Setup
  - BP Setup

    - Price List Setup
    - Product Setup
    - Request Setup
    - Requisition Setup
    - Sales Setup
    - Tax Setup
    - etc.
- 单据进程（Document Process）P
    - process_Cash
    - process_Inventory
    - process_Invoice
    - process_Journal
    - process_Journal Batch
    - process_Movement
    - process_Order
    - process_Payment
    - process_Requisition
    - etc.
- 单据值（Document Value） V
    - 没有条目 (可能是在Garden World?)

工作流窗口

- Workflow标签
    - 工作流类型
    - 数据访问级别（ALL，组织，公司等）
    - 启动节点
    - 工作流进程（在所有条目中为空，只有一个：系统工作进程）

- Node 标签
  - 组合框工作流责任人（申请人，组织）
  - 组合框启动类型（自动，手动）
  - 组合框连接元素（异或、与）
  - 组合框分割元素（异或、与）
  - 组合框操作（Action）（(Apps process, Apps Report, Apps Task, Document Action, Email, Set Variable, Sub Workflow, User Choice, User Form, User Window, Wait (Sleep). ）好象与 MWFActivity: performWork().中的参数对应，提供 Task, Sub Workflow, User Workbench, User Form and User Window，没实现。

    根据选中的操作（action）提供域，在 User Window（用户窗口），组合框中为所有可能的窗口，在单据操作（Document Action）中，组合框中为 Document Action 的相关条目（(Approve, Close, Complete, Invalidate, Post, Prepare, Void, Unlock 等，与 BO 源代码中的 DocAction 方法相同）。
- Transition 标签
  - 下一个节点: 要执行的下一个节点
  - 可能有多个下一个节点，其中一个是标准节点，当节点可执行时，定义节点的可执行的操作（Operation）
- Condition 标签
  只用于
  - 与/或
  - 列
  - 操作符（+，-等）
  - 值

总结:

在工作流（Workflow）标签，定义哪一个节点为启动节点

节点定义操作，执行哪一个操作

对一个单据操作(Document action)来说，选择有 close，prepare 等

结果是调用一个同名的 BO 方法，同时改变状态

Transition 定义下一个节点

ERM

- Static 静态
  所有定义在 AD 中
  - 一个 AD_WORKFLOW 可以有多个 AD_WF_NODE
- Dynamic 动态
  执行过程中创建进程
  - 一个 AD_WF_ACTIVITY 有一个 AD_WF_NODE
  - 一个 AD_WORKFLOW 有多个 AD_WF_process
  - 一个 AD_WF_process 有多个 AD_WF_ACTIVITY

Node 标签中的操作（action）

- 引用: WF_Action
  - List Validation Search Key
  - Apps process P
  - Apps Report R
  - Apps Task T
  - Document Action D
  - Email M
  - Set Variable V
  - Sub Workflow F
  - User Choice C
  - User Form X
  - User Window W
  - User Workbench B (inactive)
  - Wait (Sleep) Z

Bo 工作流文档编制:

The history of the workflow, in which this BO is involved, can be inspected using the icon Active Workflows (two squares connected with a arrow) in any of the BO lines, which implement DocAction. So it can be seen where the workflow is.

这一段应该是说通过 Active Workflow 图标可以查看工作流的历史，由实现 DocAction 的 BO 来完成

Loading:

Apanel: actionPerformed（ActionEvent e）由按钮调度

对工作流按钮来说调用 AEnv: public static void startWorkflowprocess (int AD_Table_ID, int Record_ID)

此处读取一个 AD_WF_process 数据集（当前表的当前数据集）

Saving:

X_AD_WF_process 对象，MWFprocess 的父类是 X_AD_WF_process。但是 X_AD_WF_process 也是 PO 的一个子类，因此保存它的一个实例，在构造器中保存刷新。

工作流举例: Order（订单）

从窗口启动，源码结束

下面是主界面的一些操作

- Sales Order of menu（销售订单菜单）
- Zoom Window Sales Order, Tab Order, Field Table (Content: C_Order). 调整窗口 Sales Order, Tab Order, Field Table
- Table C_Order, Tab Column, column name processing （列名称进程）
- Field Reference is a button; Field process references to C_Order process.
  Zoom into this:
- field Workflow has the content Process_Order (one of the defined general workflows) 域 workflow 中包含 Process_Order 的内容，Process_Order 定义在普通工作流中

窗口工作流编辑

读工作流的节点，转换到下一个节点，图形化显示其 JOIN 约束和 SPLIT 约束和显示

做为连接条件 XOR 表示选中第一个可能的操作（action）

窗口 Report & Process（Window Report & Process ）

调度程序为 processCtl.run()

报表与进程（reports/processes）的特点:

- Document Action（单据操作）
  参考 processCtl
- Java-class
  在 class Name 域中判定定义在 AD 中的报表类，执行（本例中是 org.compiere.process.ImportInventory）
  prepare()和 doit（）方法控制按钮的行为。

  ImportInventory 的方法 prepare()和 doit()是如何调用的:

  - processCtl 管理进程调用
  - 若检测到是一个 java 调用，processCtl 的 run()调用 startprocess()
  - processCtl:private boolean startprocess()
    调用 startJavaprocess(m_pi, m_trx)（本地进程）
  - processUtil: public static boolean startJavaprocess(
    processInfo pi, Trx trx) 调用一个进程调用方法
    startprocess(Env.getCtx(), pi, trx)
  - 接口 processCall 声明 startprocess（），在类中实现 public boolean startprocess (Properties ctx, processInfo pi, Trx trx) ，本例中由 Svrprocess 实现
  - 因为 Java 类 ImportInventory 的定义如下: public class ImportInventory extends Svrprocess 且 Svrprocess 定义如下: public abstract class Svrprocess implements processCall ，特别是类 Svrprocess 实现了 processCall 的方法 startprocess()。
    所以: 调用私有方法 process()，该方法立即调用 prepare()和 doit()，这两个方法都定义在抽象类 Svrprocess 中，子类中必须定义，这在类 ImportInventory 实现，这些方法也由类 ImportInventory 执行。
  - Protected void prepare()
    从进程信息变量 m_pi 中读调用参数，赋值为 Properties
  - protected void doIt()
    实现进程，返回一个字符串，用于错误控制和信息发送

  这是 AD 与程序代码在 java 类调用时的交互情况
  更多内容参考类 processCtl 描述

- Oracle-Procedure Oracle过程
- Workflow 工作流
- Report 报表
- Jasper Report Jasper报表

Call chain for parameter displaying in Report&Process calls

这里只解释AD中的参数是如何根据定义显示的，下一节介绍Report&Process如何工作

- AMenuStartItem.startprocess() org.compiere.apps
- processDialog.init() org.compiere.apps
  取进程信息
- processParameterPanel.init() org.compiere.apps
  取所有进程参数
  SELECT p.Name, p.Description, p.Help, p.AD_Reference_ID,
  p.AD_process_Para_ID, p.FieldLength,
  p.IsMandatory, p.IsRange, p.ColumnName, p.DefaultValue,
  p.DefaultValue2, p.VFormat, p.ValueMin,
  p.ValueMax, p.SeqNo, p.AD_Reference_Value_ID, vr.Code AS
  ValidationCode
  FROM AD_process_Para p
  LEFT OUTER JOIN AD_Val_Rule vr ON
  (p.AD_Val_Rule_ID=vr.AD_Val_Rule_ID)
  WHERE p.AD_process_ID=? AND p.IsActive='Y' // z.B.238
  ORDER BY SeqNo
  对每个参数来说调用createField()。
- processParameterPanel.createField() org.compiere.apps
  先调用GridFieldVO.createParameter()
  调用构造器 new GridField (voF)，调用loadLookup()
  变量 vot.AD_Column_ID引用参数ID，保存在数据表AD_process_Para中而不是 AD_Column中的ID
- GridFieldVO.createParameter() org.compiere.model
  调用 GridFieldVO的构造器，ColumnName, Name, Description,
  AD_Reference_ID, FieldLength, 等数据从参数中读取，类变量lookupInfo包含可能查找的信息

- GridFieldVO.initFinish() org.compiere.model
  report&process-parameters 的域 Reference 值等于 List, Table, TableDirect 或 Search 时，处理查找
  域 reference 值取自数据表 AD_Reference，如“List”和“Search”的 ID 为 17 和 30，调用 getLookupInfo()，结果保存在 MLookupInfo lookupInfo（只是一个变量，没有数据）中，该变量是 GridFieldVO 的一个类变量
- MLookupFactory.getLookupInfo() org.compiere.model
  根据 List, Table, TableDirect 或 Search 采取不同的处理方式
  Lists: getLookup_List()。
  Table or Search getLookup_Table()。
  Reference 域值取自 AD_Reference，如“AD_User_SalesRep”的 ID 为 190。
  其他情况: getLookup_TableDir()。
  lookup 调用之后，进一步完成 SQL 语句（增加 Clint_ID 条件和角色条件）。
  包含参数信息的对象返回到 initFinish()
  - MLookupFactory.getLookup_Table() org.compiere.model
    发送一个 SQL 查询

```
SELECT t.TableName,ck.ColumnName AS KeyColumn,cd.ColumnName AS
DisplayColumn,rt.IsValueDisplayed,
cd.IsTranslated,rt.WhereClause,rt.OrderByClause,t.AD_Window_ID
,t.PO_Window_ID,
t.AD_Table_ID
FROM AD_Ref_Table rt
INNER JOIN AD_Table t ON (rt.AD_Table_ID=t.AD_Table_ID)
INNER JOIN AD_Column ck ON (rt.AD_Key=ck.AD_Column_ID)
INNER JOIN AD_Column cd ON (rt.AD_Display=cd.AD_Column_ID)
WHERE rt.AD_Reference_ID=190 -- "AD_User - SalesRep "
AND rt.IsActive='Y'
AND t.IsActive='Y'
```

    SQL 查询的结果内容如下:
    - TABLENAME AD_User
    - KEYCOLUMN AD_User_ID
    - DISPLAYCOLUMN Name
    - ISVALUEDISPLAYED Y
    - ISTRANSLATED N

- WHERECLAUSE EXISTS (SELECT * FROM C_BPartner bp
  WHERE
  AD_User.C_BPartner_ID=bp.C_BPartner_ID AND
  bp.IsSalesRep='Y')

  这个 Where-Clause 在 SQL Developer 中不能正确执行。在后面构造的 SQL 中，WHERE 条件类似下面的内容:
  SELECT * FROM C_BPartner bp
  inner join AD_User adu on
  (adu.C_BPartner_ID=bp.C_BPartner_ID)
  WHERE bp.IsSalesRep='Y'

  本例中，显示所有在 C_BPartner 表中标记为 SalesRep 的用户。
- ORDERBYCLAUSE --
- AD_WINDOW_ID 108 (the same asWindow Namens Task)
- PO_WINDOW_ID --
- AD_TABLE_ID 114 (the same as table Namens User/Contact)

AD_RefTable 表包含表 AD_Reference 实例的属性，特别是表引用，Key 与显示列，以及对引用有效的 SQL 约束，这些域可以在 ADempiere 的 Reference 窗口的 Table Validation 标签中找到。
这使得上次查询的结果可懂。
若 keyColumn 没有以_ID 结尾，插入 NULL 代替。
若 keyColumn 以_ID 结尾，插入 NULL。

构造如下 SQL 查询:

```
SELECT AD_User.AD_User_ID,
NULL,
AD_User.Value || '-' || AD_User.Name,
AD_User.IsActive
FROM AD_User
WHERE EXISTS (SELECT * FROM C_BPartner bp WHERE
AD_User.C_BPartner_ID=bp.C_BPartner_ID AND bp.IsSalesRep='Y')
ORDER BY 3
```

定义带 reference=Table 的参数，同时引用属性: Table Name, Key Column, Displayname, SQL 约束。

使用(TABLENAME,KEYCOLUMN, etc)和SQL实例化一个MLookupInfo。
SQL结果如下(略写)

```
AD_USER_ID  NULL AD_USER.VALUE||'-'||AD_USER.NAME  ISACTIVE
1000000              -VidesAdmin        Y
1000001      -VidesUser         Y
1000808      05-TORRES        Y
1000311      15-TOLENTINO       Y
101        gardenad-GardenAdmin      Y
102        gardenusr-GardenUser      Y
1000002      labvides-LabVIDESAdmin    Y
1000003      labvides-LabVIDESUser      Y
1000004      lsaravia-LSaravia       Y
1001142      wbeltran-W.BELTRAN      Y
```

等。

- MLookupFactory.getLookup_TableDir() org.compiere.model
  对tableDirect来说，域必须名称为TableName_ID，否则会出错
  列显示，标记为Identifier
- getLookup_List()
  根据ReferenceKey中选中的域，这里显示一个列表。
  域reference Key没有值的时候显示空列表
  分析输入的参数
  - 参数保存使用 (org.compiere.apps) processParameterPanel.saveParameters()，保存在表AD_PInstancePara中。
    根据参数类型的不同，保存在不同的列。
  - 报表执行时参数从表AD_PInstancePara中读取，实现方法: (org.compiere.print) ReportEngine.get()调用 (org.compiere.model) Mquery.get()
  - 接下来的内容参考Printing章节

Callouts

研究ADempiere-wike+own

- “Callout”的意义是什么?

- Callout 是一个 JAVA 方法，该方法在域焦点变化时执行，（域值是否改变）
- Callout 是一个 JAVA 方法，该方法在 ADempiere 窗口的域值变化时执行。一个 Callout 类（extends CalloutEngine）将不同的方法分组，这些方法在使用 UI 修改列的值的时候调用，对一个列（参考 AD_Column. Callout 数据库列和 Table and Column 标签）可以指定一系列的方法（以 “;” 分开）。
- Callout 不用于数据验证，代之动态验证（AD）
- Callouts 可以读当前域或其他域，用于数据条目结果，如计算 GUI 中需要立即显示的总和。
- Callout 部署在包 org.compiere.model，同时还有 CalloutInvoice 和 CalloutEngine
- 如果要在 Callout 中做计算，必须在相关 PO 类（beforeSave()）中重做，这样保证不同 UI 访问如 HTML 界面

- Ad 中的 Callout
  - 系统管理员
  - Table&Column 窗口，Column 标签
  - 域 Callout 包含方法，如数据表 C_Order, Column Target Document Type, 域 Callout 值为 org.compiere.model.CalloutOrder.docType
- Code 中的 Callout
  - Callout 层次举例:
    - public class CalloutOrder extends CalloutEngine
      Package: org.compiere.model
    - public class CalloutEngine implements Callout
      - start()方法的实现分析文本，并使用参数调用**方法**:
        (String) method.invoke(this, args)。
        start()由谁来调用？？？？
      - 调用并执行**方法**（如 docType）
      - 作者没有解释如何调用指定的 Callout
    - public interface Callout
  - Callout 签名
    每个 Callout 有着相同的签名

public String my_callout_method (Properties ctx, int WindowNo, GridTab mTab, GridField mField, Object Value)
类 calloutOrder 中的例子:
public String docType (Properties ctx, int WindowNo, GridTab mTab, GridField mField, Object Value)

- Properties **ctx**
  如下使用:
  MPriceList.getStandardPrecision(**ctx**, M_PriceList_ID)
  Env.setContext(**ctx**, WindowNo, "OrderType", DocSubTypeSO)
  MWorkflow.get (**ctx**, AD_Workflow_ID)
- int WindowNo
  多数由类 org.compiere.util.Env 使用:
  Env.setContext(ctx, WindowNo, "OrderType", DocSubTypeSO)
  Env.getContextAsInt(ctx, WindowNo, "C_BankAccount_ID")
  Env.getContext(ctx, WindowNo, "DiscountSchema")
  Env.getContextAsDate(ctx, WindowNo, "PayDate")
- GridTab mTab
  代表一个记录中的所有列，使用如下:
  (BigDecimal)mTab.getValue("QtyEntered")
  mTab.setValue ("M_Product_ID", new Integer(M_Product_ID))
  mTab.setValue("DateAcct", Value)
- GridField mField
  类 GridField 中的方法 Grid Field Model 使用:
  (getAD_Tab_ID(), getValue(), isDisplayed(), getAD_Column_ID() etc.)。
  例: String colName = mField.getColumnName();
- Object Value
  Value 代表域的值，需要转换为类可用的，根据是下面域的数据类型:
  ((Integer)Value).intValue()
  (Timestamp)Value
  (BigDecimal)Value

- 功能
  - 使用 getValue()与 setValue()解释执行并改变逻辑
  - 避免循环（并不是一定要用）

在 callout 的开始 setCalloutActive(true);
在 callout 的结尾 setCalloutActive(false);

上下文值的验证

- 在 AD 中的配置
  - 显示
    窗口的 Window, Tab&Field
    任何标签，域的显示逻辑包含如下命令:
    - @IsEmployee@=N
    - 1=2 // wird also nie angezeigt
    - @$Element_U2@=Y
    - @processed@=Y & @#ShowAcct@=Y
    - @IsCustomer@='Y'
    - etc
  - 只读
    数据表的 Table&Column
    任何列，域 Derad Only Logic 包含以下命令:
    - @OrderType@='WP'
    - @IsDropShip@=Y
    - @AD_OrgBP_ID@!0
    - @ProductType@=R | @ProductType@=E | @ProductType@=O
    - @CostingMethod@!x & @CostingMethod@!S // was is this? (in MCost, column Current Cost Price)
    - etc.
  - 强制 (Mandatory)
    数据表的 Table&Column
    Checkbox Mandatory
- Code I
  - APanel: initPanel() calls m_curTab.getTableModel().setChanged (false)

- m_cur_tab是初始化时的当前标签
- Window的initWorkbench()调用initPanel()
- Initializes panel。

- public class GridTab implements DataStatusListener, Evaluatee, Serializable
  - query()和getTableModel()调用initTab().
  - GridTab: protected boolean loadTab()
    取所有域，标准域和内容域分开
    标准域: Created, CreatedBy等
    域的调用列表根据: field.getDependentOn()
- public class GridField implements Serializable, Evaluatee
  GridField: public ArrayList<String> getDependentOn():
  {
  :
  Evaluator.parseDepends(list, m_vo.DisplayLogic);
  Evaluator.parseDepends(list, m_vo.ReadOnlyLogic);
  Evaluator.parseDepends(list, m_vo.MandatoryLogic);
  :
  Evaluator.parseDepends(list, m_lookup.getValidation())
  :
  }
- Evaluator
  public static void parseDepends (ArrayList<String> list, String parseString)
  分析对应字符串，以@为间隔

- CodeII
  Evaluate-class
  evaluateLogic()也用于方法 GridTab (isReadOnly()) 和Grid Field (isMandatory())。
  这些调用通常在程序中做
  分析
- Code III
  Evaluate类
  类中包含 isAllVaraiblesDefined()

会计处理机制

论坛中 **Karsten Thiemann** 提供的内容

会计处理机制工作在所有单据数据表（所有单据有一个 POST 按钮），并创建 Fact_Acct 条目，参考 Acctprocessor.java，特别是 PostSession()方法，Post 由 D0c_*类处理（Line 119 String error = doc.post(false, false);）。

Doc 类

package org.compiere.acct

public abstract class Doc

- 所有带会计逻辑的类如: Doc_GLJournal, Doc_Cash, Doc_Bank, Doc_Invoice, Doc_Inventory, Doc_InOut, Doc_Order, Doc_Payment, Doc_Requisition 都继承 Doc，会计逻辑由这些类来实现。
- 每个继承 PO 的 BO，都引用一个 Doc 实例，该实例为 BO 的单据:
  - private Doc m_doc, with its access methods
  - public Doc getDoc()
  - public void setDoc(Doc doc)
- 例: public class Doc_GLJournal extends Doc
- 子类必须重写方法 loadDocumentDetails(), createFacts(), getBalance()以实现自己的会计逻辑
- 属性:
  - 数据表的所有 ID，处理单据时用（Postings, documents）
    public static int[] documentsTableID
    数据表包括: C_Invoice, C_Allocation, C_Cash, C_BankStatement, C_Order, C_Payment, M_InOut, M_Inventory, M_Movement, M_Production, GL_Journal, M_MatchInv, M_MatchPO, C_ProjectIssue, M_Requisition
  - 处理单据的所有表的名称
    public static String[] documentsTableName
  - 单据类型常量
    - C_Invoice: ARI, ARC, ARF, API, APC
      如: 会计支付发货用的
      public static final String DOCTYPE_APInvoice = "API";

        - C_Payment: ARP, APP
        - C_Order: SOO, POO
        - Transaction: MMI, MMM, MMS, MMR
        - C_BankStatement: CMB
        - C_Cash: CMC
        - C_Allocation: CMA
        - GL_Journal: GLJ
        - 等等
    - post状态的常量
        - Notposted: N
          例: public static final String STATUS_NotPosted = "N";
        - NotBalanced b
        - NotConvertible c
        - PeriodClosed p
        - InvalidAccount i
        - PostPrepared y
        - Posted Y
        - Error E
    - 会计类型常量
        - Invoice – Charge
          例: public static final int   ACCTTYPE_Charge = 0;
        - Invoice - AR
        - Invoice - AP
        - AP Service
        - AR Service
        - 等
    - 其他属性
        - 会计计划
          private MAcctSchema[]m_ass = null;

- Properties
  private Properties m_ctx = null;
- 单据的 BO，继承 PO
  protected PO p_po = null;
- Doc Lines
  protected DocLine[] p_lines;
  借记卡与信用卡在这里入账（Debit and Credit is accounted here ）
- Facts
  private ArrayList<Fact> m_fact = null;
- 更多
  Document type（单据类型），-state, -No, description 描述, GL-Category, period, accounting date 记账日期, document date 单据日期, business partner 商业伙伴, bank account 银行账号 ,currency

- 方法
  - public final String post(boolean force, boolean repost)
    只有 postImmediate 调用该方法（作者按）
    - 检查单据状态有效性
    - 检查会计计划和 BO 是否是同一家公司（Client）
    - 锁定数据集
    - 调用 loadDocumentDetails() ，该方法由每个 Doc 类的会计逻辑组成
      Doc_Invoice 举例:
      - 实例化 MInvoice
      - 计算单据类型
      - 计算数量
        - getGrandTotal()
        - getTotalLines()
        - getChargetAmt()
        - Steuern ermittelt（这是个德文，不懂）
        - 装入发货清单条目:

使用 loadLines()赋值 p_lines 和数量
(docLine.setAmount())

- 若 repost==true，调用 deleteAcct()，同时使用 SQL 语句删除 Fact_Acct 中数据表和记录 ID 条目。
- 从会计计划中取数据赋值给 m_fact
调用每个会计计划的 postLogic()，子类调用 createFacts()同时赋值 m_fact，post 状态同时设置。
- 使用 setDoc(this)BO 取单据
- postCommit（Sting status），该方法保存 Facts&Receipts
- 实例化 MNode，数据从 DocumentNo, booking date, quantity, state, periode open?, balanced?中获取。

- public static String postImmediate (MAcctSchema[] ass, int AD_Table_ID, int Record_ID, boolean force, String trxName)
  - 调用 post(force, true)
  - 由 DocumentEngine: postIt()调用
- public ArrayList<Fact> createFacts(MAcctSchema as)
  - 响应会计逻辑
  - 会计类如 DocInvoice 重写
在一个巨大的方法中设置 Facts（会计逻辑）如 ARI, ARC, ARF, API, APC
  - fact.createLine() 用借记和信用（debit and credit ）定义一个 line
  - m_fact 在方法 post()中赋值
- public BigDecimal getBalance()

说明：会计逻辑涉及到专业的会计用语，翻译的不好，一般名词后我会加入英语的部分，请自行理解，如会计计划（Accounting Schema）

类 Fact

package org.compiere.acct

public final class Fact

In the end there is a class Balance

- 属性:
  - private Doc m_doc = null;
  - private MacctSchema m_acctSchema = null;
  - private ArrayList<FactLine> m_lines = new ArrayList<FactLine>() Lines in Fact_Act
- 方法:
  - 构造器
    public Fact (Doc document, MAcctSchema acctSchema, String defaultPostingType)
  - public FactLine creatLine(DocLine docLine, MAccount account, int C_Currency_ID, BigDecimal debitAmt, BigDecimal creditAmt)
  - 更多 More createLine-methods 方法
  - public FactLine[] getLines()
    返回 FactLines 数组
  - public boolean save (String trxName)
    存储所有 LINES
  - 等

类 FactLine

package org.compiere.acct

public final class FactLine extends X_Fact_Acct

public class X_Fact_Acct extends PO

FactLine 持久化

实现数据表 Fact_Acct 的功能

多用于会计类的 createFacts

- 属性:
  - private Maccount m_acct = null
  - private MacctSchema m_acctSchema = null
  - private Doc m_doc = null;

单据头

- private DocLine m_docLine = null;

- 方法
  - public void setDocumentInfo(Doc doc, DocLine docLine)
    Client. Date Acct, Period, Tax, Product, Quantity, BP, Project, Campain, Activity, usw.
  - public void setLocationFromBPartner (int C_BPartner_Location_ID, boolean isFrom)
  - public BigDecimal getSourceBalance()
    The line
  - public BigDecimal getAcctBalance()
    会计账目结算
  - beforeSave()
    实现PO中的方法
  - createRevenueRecognition()
    FactLine.save()调用
  - public boolean updateReverseLine (int AD_Table_ID, int Record_ID, int Line_ID, BigDecimal multiplier)
    Doc_MatchInv 调用
  - 等

DocTypes 和 DocBaseTypes

- DocBase Types
  在AD中定义: AD引用调用 C_DocType DocBaseType作为列表有效性验证，参数如下:
  - DocBase Type Search Key
  - AP Credit Memo APC
  - AP Invoice API
  - AP Payment APP
  - AR Credit Memo ARC

- AR Invoice ARI
- GL Journal GLJ
- Material Movement MMM
- Material Receipt MMR
- etc. (at all 23)

- DocTypes
  所有会计操作都带单据类型（Document Type）
  会计账目中所有可能的单据类型都定义在应用程序级别
  在单据类型中可以定义: Print Format 打印格式 (selectively), document count 单据计数, number of copies 拷贝数量 etc.

  程序举例: （Spanish）
  - DocType DocBase Type
  - CXP Nota de Crédito AP Credit Memo
  - CxP Retención de IVA AP Credit Memo
  - CxP Crédito Fiscal AP Invoice
  - CxP Factura AP Invoice
  - CxC Crédito Fiscal AR Invoice
  - Return Material Sales Order
  - CxP Pedido Purchase Order
  - etc. (at all 44 in in the application)

应用程序字典: 账务报告

点击 Adempiere-Menue/Performance Analysys/Financial Reporting/Financial Report 出现的窗口

这个按钮是做什么用的，在 ADempiere 中什么位置执行？？？

What is this button for and where is the executing place in Adempiere?

答案:

- 做为系统管理员登录
- 应用程序字典/菜单 Application Dictionary/Menue

- 调整（Zoom 右键）到报表窗口（内容即是财务报告）
- 选择域 Create Report（定义为按钮）
- Zoom 列 processing-process
- 显示数据表 PA_Report_Financial Report 的 processing 列
- 在 process 中设置 FinReport
- 没看懂，应该是说在 Application Dictionary/Report & process 中的 Create Report 报表中的 Search Key 列为 FinReport 的实际内容
  In Application Dictionary/Report & process is actually FinReport the content of the column Search Key of the report Create Report (column Description: Create Financial Report).
- Create Report 报表 Classname 列设置为 org.compiere.report.FinReport
  启动 Eclipse/Netbeans，找到文件 FinReport.java，包含类 FinReport，在 /adempiere_trunk/base/src/org/compiere/report。执行本类，会有数据操作
- 报表 Create Report 的列 ReportView 有条目 T_Report
  Zoom T_Report 进入 Temporary Reporting Table，该数据表预定义列的数目（29）和列的格式，报表在其中写入相应的值，作者认为其他报表使用 T_Report(Author)

Application Dictionary 与 Application-Code 的交互使用 Landed Costs 作例子

功能:

- Application Dictionary: 实现 AD 的 Landed Costs 和 JAVA 方法
  - 以系统管理员登录
  - 窗口 Menu/Requisition-to-invoice/Invoice (Vendor)，标签 Landed Costs，在这里可以选择方法并通过按钮 Distributed Costs 启动分配（distribution），触发哪个类？？？？

    以系统管理员登录，打开窗口 Menu
    到窗口 Landed Costs (Vendor)
    标签 Landed Costs
    找到名称: Distribute Costs

Zoom into Column（列名称 processing），进入定义 C_LANDED_COST 的数据表，列 Processing 定义为一个按钮，该按钮调用进程 C_Landed_Cost_Distribution

Zoom into 进程 C_Landed_Cost_Distribution，进入其定义：域 Classname 为 org.compiere.process.LandedCostDistribute。

- 应用程序：准备计算 Landed Costs
  - 在发货清单中必须设置商业过程和 Company Agent 等，使用产品/责任人、数量和价格在标签 invoice lines 中创建 invoice lines。
  - 若 line 需要，在标签 Landed Costs Cost Distribution (值,数量等)，设置 Cost Element (Transport,etc.) 和原材料收据。
    保存：表 C_LandedCost 包含数据
  - 标签 Landed Cost Allocation 必须为空，由程序计算生成。
  - Distribute Costs 按钮：
    应用程序触发 Landed Costs 一个发货清单行。
    若使用多个发货清单行计算整个发货清单，每一行都要使用按钮手工处理。
- 代码
  - processCtl:Call of processUtil.startJavaprocess()
  - processUtil.startJavaprocess()：调用 process.startprocess()
  - LandedCostDistribute (Svrprocess)：startprocess()调用 process().
  - LandedCostDistribute (Svrprocess)：process()调用 prepare() 和 doIt().
  - LandedCostDistribute (Svrprocess)：doIt()创建一个 Landed Cost 实例并调用 m_lc.allocateCosts()。
    因为这是一个 doIt()方法，一定是一个管理进程 AD-process，在程序的某处调用。
    日志条目：
    LandedCostDistribute.doIt:
    MLandedCost[1000000,CostDistribution=Q,M_CostElement_ID=1000006,M_InOut_ID=1000039]
  - MLandedCost：allocateCosts()：调用 invoiceLine.allocateLanded Costs()。

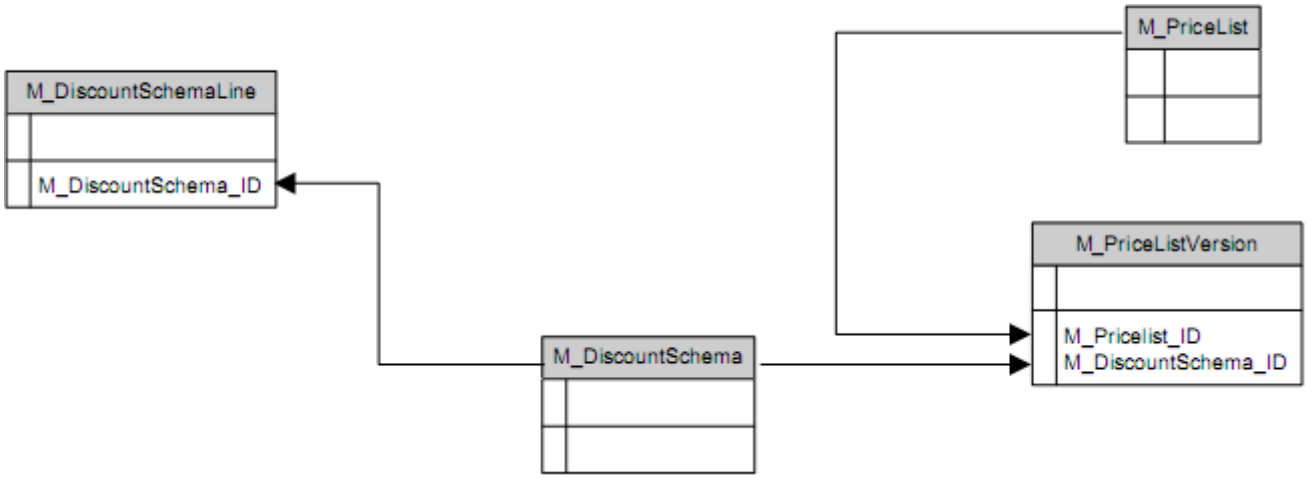
- MInvoiceLine: allocateLandedCosts():
  - 发货清单行的所有定义的 Landed Costs 从数据表 C_LandedCost 中读取并实例化到一个 MLandedCost 数组
  - 删除与发货清单行有关的所有表 C_LandedCostAllocation 的条目
  - 单行与多行是有区别的
  - 单行:
    采用计算后的 Landed Cost 设置出货和一个出货行列表
    计算求和
  - 等

Price lists （价格表）

- 每个产品都是 Product Category 的一部分
- 每个产品可以有多个 Price List 版本, List Price, Standard Price 和 Limit Price
- 在 Material Management/Material Management Rules 有 Price List Schema (数据库表: M_DiscountSchema), 每个 Price List Schema 中, 可以选择域 Valid From, Discount Type 和按钮 Renumber, 每个 Price List schema 的更多的条件可以定义为 schema line(表: M_DiscountSchemaLine), 有效的 BO 如 Business Partner, Product Category or Product 可以定义每个条件, Schema lines 拥有定义执行顺序的序列号（小号启动, 大号结束）, 缺省步骤定义为 10, 一般来说条件主要是价格与折扣（List Price Discount in %, List Price Min Margin, List Price Max Margin 等）。
- Material Management/Material Management Rules 有一个 Price List（表: M_PriceList）, 定义价格列表与版本（表: M_PriceListVersion）, 更重要的是对每个价格表来说产品价格可以嵌入注册版本, 这需要 Price List Schema 和一个（不活跃的）Price List Version（程序中错误的称为 Base Price List）, 在组合框中可以选择所有活跃的 Price List Versions。
  最后一个价格列表可以根据 Price List Schema 和 Price List Version 定义, 点击按钮 Create Price List。
  也可以用数据库跟踪: 在表 M_PRICELIST_VERSION 中有一个域 M_PRICELIST_VERSION_BASE_ID, 该域引用表 M_PRICELIST_VERSION 的一个数据集。比较自

相矛盾的是这个域定义为了外键。

- 总结

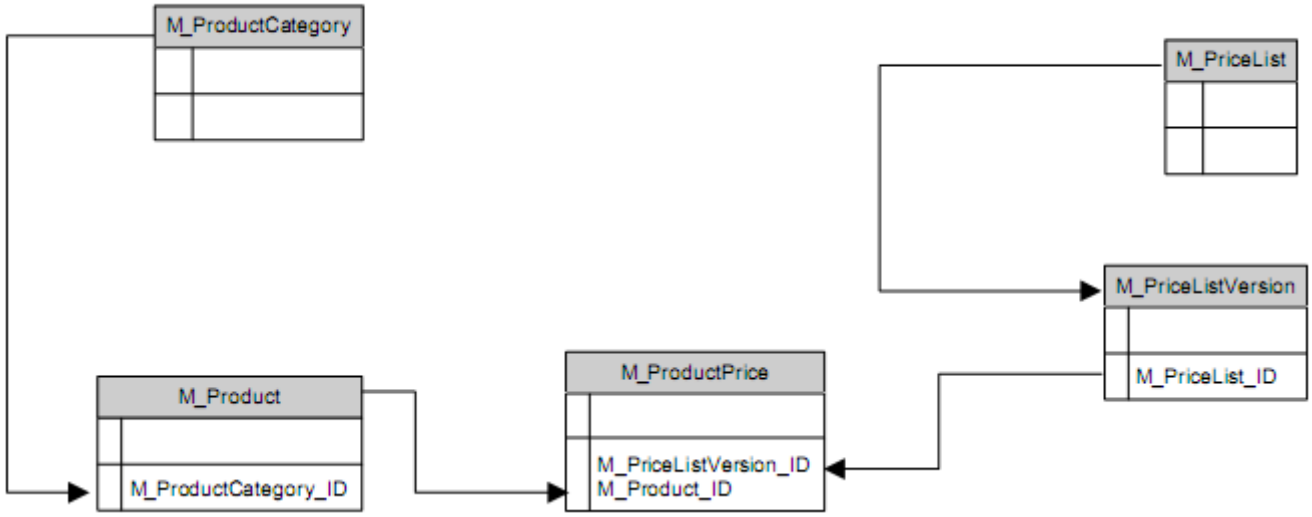


- 每个 PriceListSchema(M_DiscounSchema)都由 Schema Lines (M_DiscounSchema Line)组成
- 每个 Price List 都有 Price List Version

德语看不懂，照搬:

- Ein Price List Schema (M_DiscounSchema) hat mehrere Schema Lines (M_DiscounSchemaLine).
- Eine Price List hat mehere Price List Versions.
- Die Price List Version Verweist auf ein Price List Schema.
- Wenn man also aus eine Price List Version die Preise kalkulieren will, greift Adempiere auf die im Price List Schema definierten Schema Lines zurück

价格计算



下面这一段是德语的，猜测翻译，原文照搬:

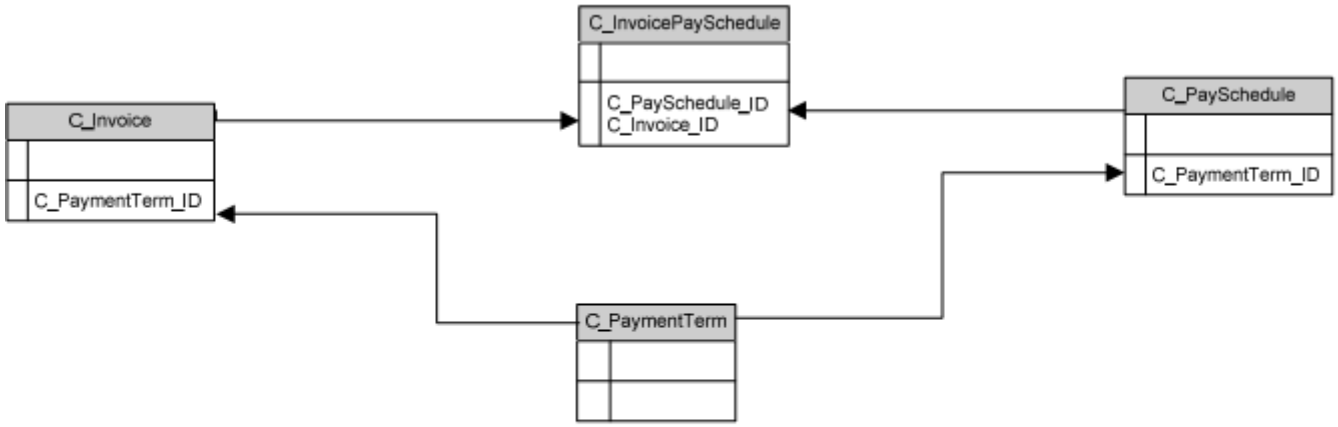
每个产品都属于一个产品分类

每个PirceList都有Pirce List Version

每个产品和每个Price List Version都用于计算Product Price

- Ein Product gehoert zu einer Product Category.
- Eine Price List hat mehere Price List Versions.
- Ein Product und eine Price List Version verweisen auf einen Satz von Product Price. Dazu kommen in M_ProductPrice die Spalten PriceList, PriceStd, PriceLimit, die für Listenpreis, Standardpreis und Mindestpreis stehen.
- In M_ProductPrice wird das Ergebnis der Preiskalkulation festgehalten. Im Fenster Price List, Register Product Price werden die Werte für Listenpreis, Standardpreis und Mindestpreis angezeigt.

支付条款



1.- Ein Payment Term kann mehrere Payment Schedules haben

2a.- Beim Invoice wird ein Payment Term angegeben (Register "Invoice")
2b.- beim Invoice wird aus dem gewählten Payment Term die Schedules definiert (Register "Payment Schedule")

视图 RV_C_INVOICE, RV_OPENITEM

RV_OPENITEM 是一个视图，选中 Menue/Open Items/Open Items 时调用，视图 RV_C_INVOICE 在 RV_OPENITEM 中使用。

1. 在 RV_C_INVOICE 视图中，表 C_INVOICE 与表 C_DocType, C_BPartner,

C_BPartner_Location 和 C_Location 采用内部并计算连接，结果包含所有引用 C_INVOICE 的所有行，所有在 C_INVOICE 中没有更多的行。

语义：可能只有完整的发货清单才能被处理。

几乎所有 RV_C_INVOICE 的列（52 列）来自表 C_INVOICE（60 列），除了 C_CountryID, C_Region_ID, Postal 和 City, 这些内容来自表：C_Location。

在 C_INVOICE 中有一列 DocStatus，值由两个字母组成：CO->完成，DR->草稿，参考本文档的其他内容可以找到一个完整的列表。

创建视图时，若 Multiplier 的值为-1（这时 DocBaseType 列的第三个字母为“C”），表 C_INVOICE 中的列 ChargeAmt, Totallines 和 GrandTotal 可以转变为

负值。

原文如下:

Columns ChargeAmt, Totallines and GrandTotal of the table C_INVOICE can switch to negative values while creating the view, if Multiplier gets the value-1, when the third letter of the column is DocBaseType equals "C".

DocBaseType 列的值:

- CMC (Cash Journal 现金收支),
- APC (Vendor Credit Memo 厂家信用备忘),
- ARC (Credit Memo 信用备忘),
- MMR (Vendor Delivery 厂家发货),
- POO (purchase Order 购买订单),
- SOO (Order Confirmation 订单确认, Proposal 建议, Quotation 报价,等).

CMC、APC 和 ARC 只是一些值，第三个字母为 "C"，好象只有这些单据类型才能用于构造视图 RV_C_INVOICE。

函数 charAt()读字符串中的第 N 个字母（代码很简单: RETURN SUBSTR(p_string, p_pos, 1)）

2. 在视图 RV_OPENITEM 中使用视图 RV_C_INVOICE

这有点复杂

- 视图 RV_C_INVOICE 由一个并集组成
  注意:
  - 选中表列的数据类型必须与并集中所有表相同
  - 返回无重复的简单并集，这说明会过滤相同行
  - 并集 ALL 返回所有行，含重复行

  并集第二部分的列的名称与顺序和第一部分的必须相同
- 并集的第一部分:

- 首先创建视图 RV_C_INVOICE 与表 C_PaymentTerm 之间的一个内部并
  一个支付条款可以有多个 schedules（时间表），支付条款在创建发货
  清单时由注册 Invoice(发货清单)计算。
  在第一个“并运算”时，DueDate, DaysDue, DiscountDate,
  DiscountAmt, PaidAmt, OpenAmt 从选中的 Payment Term 计算而来
- 计算由 paymentTermDueDate()这样的函数实现，该函数调用方法
  org.compiere.sqlj.PaymentTerm.dueDate(i.C_PaymentTerm_ID,
  i.DateInvoiced)，该方法使用发货日期和支付条款来计算到期日期。
- 还使用其他 Oracle 函数如: addDays(i.DateInvoiced,p.Discount
  Days)，该函数取表 C_PaymentTerm 中的 DiscountDays，该函数如下处
  理: RETURN TRUNC(p_date) + p_days。
- 视图 RV_OPENITEM 的一些列定义为更新列: DueDate, DaysDue,
  DiscountDate, DiscountAmt, PaidAmt, OpenAmt。这些列的值由函数
  来计算，与第二个“并运算”的不同的就是这些列的计算。
- 几乎所有其他列都来自视图 RV_C_INVOICE，同时几乎所有视图
  RV_C_INVOICE 的列都来自表 C_INVOICE，可以说视图 RV_C_INVOICE 只是
  表 C_INVOICE 再加上一些计算域。
- where 条件的定义: 选中行中付款未执行 (i.IsPayScheduleValid<>
  'Y')且单据状态不能为“Draft”(i.DocStatus<>'DR')。

- 第二个并运算
  - 视图 RV_C_INVOICE 与表 C_InvoicePaySchedule 做一个内部并运算。
    注: 表 C_InvoicePaySchedule 显示在窗口 Invoice（Customer）中的
    第四个标签（Payment Schedule），在这里选中 Payment Schedule 与
    其他参数如到期日期（maturity date）和折扣日期（discount date）
    ，付款日程依赖付款条款，在对应的注册发货清单（Invoice）中选中。

    窗口 Invoice(Customer）在 Menue/Quote-to-Invoice/Sales
    Invoices/Invoice (Customer)中执行。
  - 使用与第一个并运算相同的列，只是选中列不同，如: DueDate,
    DaysDue, DiscountDate, DiscountAmt, PaidAmt, OpenAmt。
  - 附加 where 条件: payschedule 是否有效

那么这个视图到底做了些什么呢？视图为发货清单集选择到期的发货清单，为每个

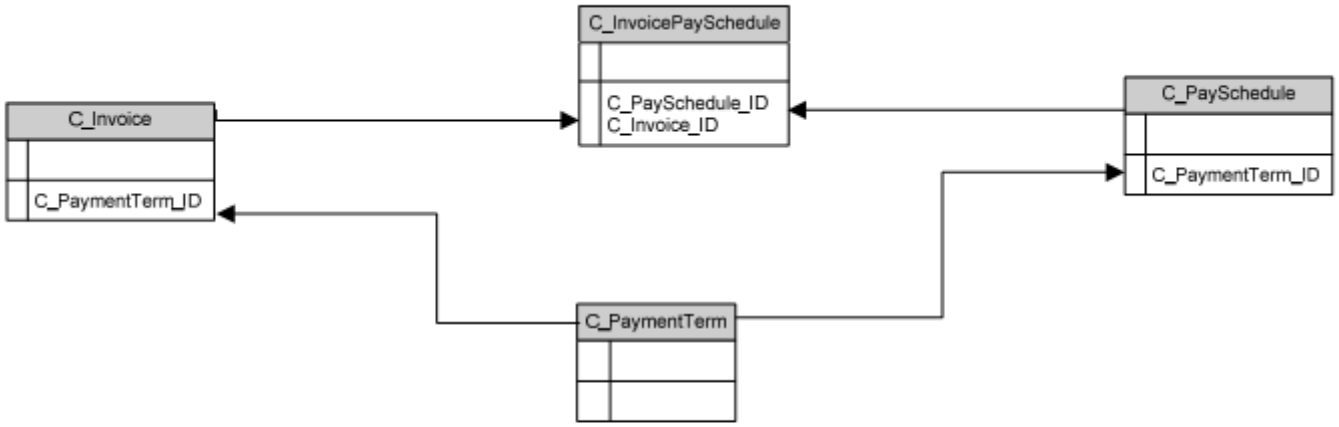
发货清单计算当前日期和折扣，我不知道为什么要计算两次（一次在 Payment Term 另一次在 Schedules）（作者按）

更有趣地是这个视图保存在临时表 T_Aging 中（我认为这是因为性能原因）

字典结果:

- Report&Process RV_T_Aging 使用 Report View T_Aging
- RV_T_Aging 使用 org.compiere.process.Aging 作为类名称，并调用视图 RV_OpenItem
- T_Aging 定义为数据表（而不象其他报表中那样定义为视图）

下面的图表描述了表之间的关系:



图表中的文字为德语，不会翻译。

注: 作为例子，表 C_InvoicePaySchedule 与 Payment Schedule 在字典中的注册定义可以看出并非所有定义在表中的列都可以通过字典访问，在表 C_InvoicePaySchedule 定义中有 17 列，而在字典中只有 13 列，created, createdBy, updated and updatedBy 列不能通过字典访问。

Db 中关于仓库结构说明

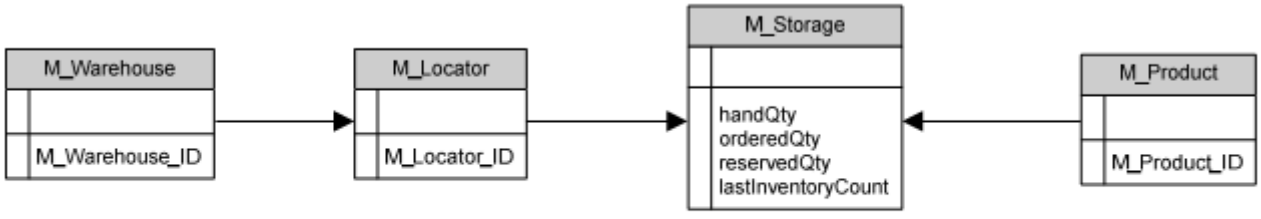
- ADempiere 中的仓库可以在 Material Management/Material Management Rules/ Warehouse and Locator 中找到，在这里可以查看仓库的位置和每个

位置的存储。

- 仓库保存在表 m_warehouse 中
  每个仓库可以有多个位置（Locators），写在表 m_locator 中
  Locators 拥有:
  - Search Value(快速查询用)，值域的格式建议使用三维仓库定义：XX-YY-ZZ，如：08-13-06，定义 Aisle 8，Din 13，Level 6。
  - Aisle 顺序： “X”
  - Din 顺序： “Y”
  - Level 顺序： “Z”
    这种三维仓库可以建立很多位置，每个位置（Locator）和产品（Product）可以定位到一个 storage。

    最好用这种格式作为 Search Key: warehouse xx-yy-zz，在报表中这样显示： “从中央仓库 23-44-3 转运到分销仓库 21-44-4”
- Storages 保存在表 m_storage 中，参数为 on hand quantity 现有数量，reserverd quantity 存货数量，ordered quantity 定货量，last inventory count 产品清单。

  每个产品和位置可以使用不同的 storages，所有不需要 3 维 storage。
  这样做比较实用，例如：相同产品批次可以有不同批号（charge numbers）和保修日期。



- Ein Warehouse hat einen oder mehrere *Locators*.
- Pro Locator und Produkt kann man eine *Storage* anlegen.

图中文字意思猜测：每个仓库有多个 Locator，每个 Locator 和 Produce 由 Storage 管理

m_storage 条目不能删除（(delete from m_strorage where ad_client_id= 1000001 是不可行的），如果 m_storage 条目有错误，ad_client_id, m_product or m_locator 必须改变。

下面的SQL语句可以找出位置与仓库的相关性:

```
select w.m_warehouse_id, w.name, l.m_locator_id
FROM m_warehouse w
INNER JOIN m_locator l ON (w.m_warehouse_ID=l.m_warehouse_ID)
where w.ad_client_id=xxxxx
```

注意 ad_client_id一定要正确。

属性集实例

以下内容是属性实例在哪里定义的说明

- MAttributeSetInstance.setDescription()
  组成属性的描述符
  其他情况调用 MattributeSet.getLotCharStart()和MAttributeSet.getLotCharEnd()
- 在窗口Attribute Set, field Lot Char Start Overwrite和lot Char Start Overwrite定义值，空值为不处理。

其他相关表关系:

```
                    1                                  n
MAttributesetInstance (Lot,Serno, Guaranteedate)------> MStorage
 (m_attributesetinstance_id qtyonhand,  qtyreserved,qtyordered)
```

对相同的产品定位器可以有不同(Mattributeset Instances)

下面的SQL查询用于获取定位器，产品，数量和描述:

```
select loc.value, prd.value, st.QTYONHAND,
asi.m_attributesetinstance_id, asi.description
from  m_storage st
join m_attributesetinstance asi on
(st.M_ATTRIBUTESETINSTANCE_ID=asi.M_ATTRIBUTESETINSTANCE_ID)
```


join m_locator loc on (st.M_LOCATOR_ID=loc.M_LOCATOR_ID)

join m_product prd on (st.M_PRODUCT_ID=prd.M_PRODUCT_ID)

order by asi.lot, asi.guaranteedate, prd.value;

更多关系:

1 n

MAttributesetInstance(Lot,Serno, Guaranteedate) ------> Morderline(m_attributesetinstance_id, qtyordered, qtyreserved, qtydelivered, m_product_id)

完

