# 贝特瑞负极六号开发文档

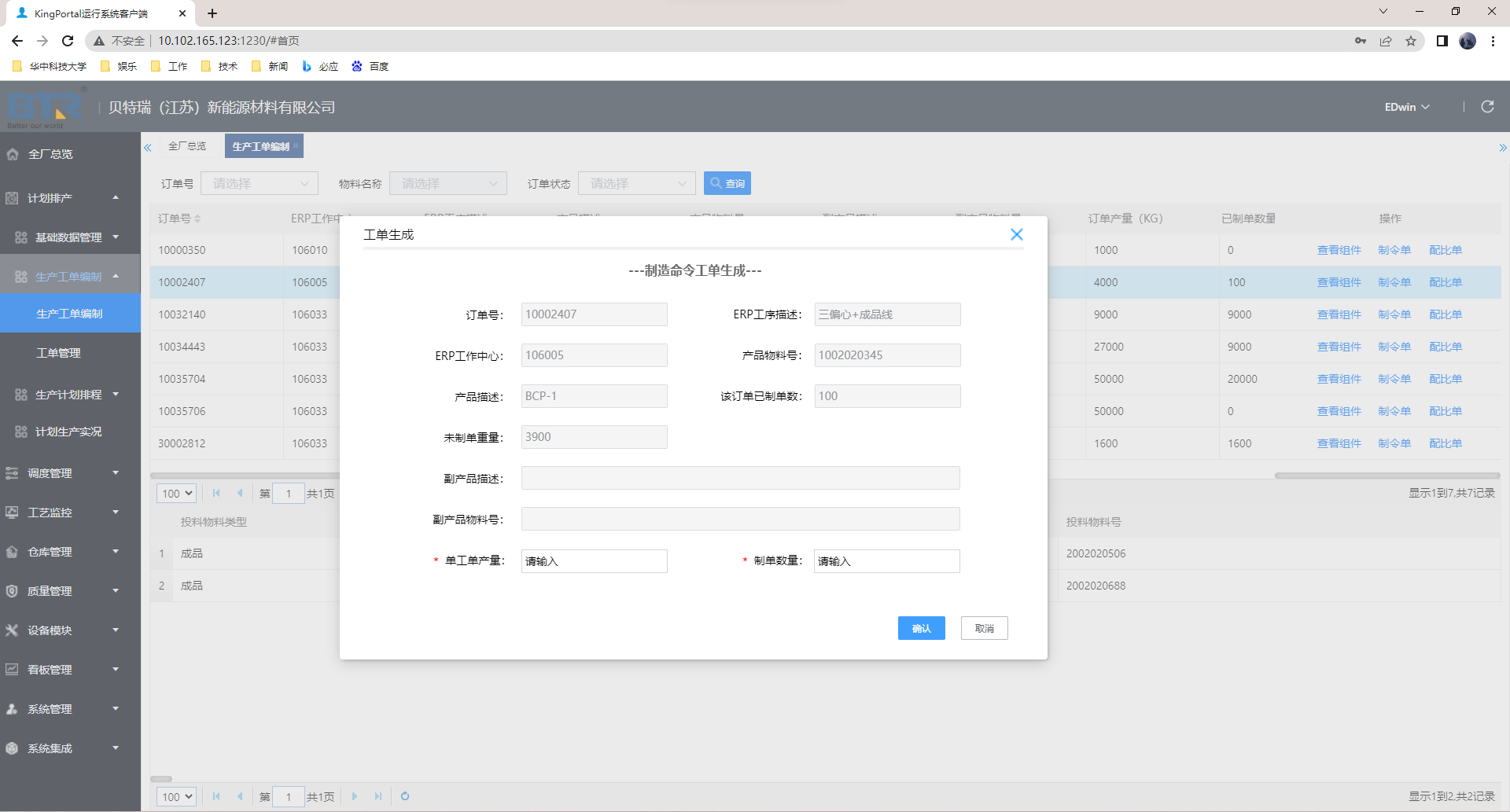
## 计划模块

### 1.1 工单编制

#### 数据库表

|  |  |
| --- | --- |
| OrderAFVCData | ERP工艺路径 |
| OrderHeaderData | ERP销售订单 |
| OrderHeaderData | ERP的BOM信息 |
| Order\_information | 销售订单整合接口 |
| productOrder\_realTime | MES生成工单实时表 |

#### 前端界面



图表 1 工单编制

图形用户界面, 应用程序

描述已自动生成

图表 2 工单管理

#### 业务流程

生产订单由SAP生成，MES根据SAP生成的生产订单，将订单拆分成不同重量的制造命令单或生产配比单，每个工单下又分为重量相同的小批次。生产订单可分为多次拆解。

#### 开发思路

通过ERP接口工具接收到和生产订单信息的三个表后，通过OrderHeaderData表中的触发器，匹配OrderAFVCData和OrderHeaderData中的工艺路径以及BOM信息，将订单信息，ERP工作中心，BOM用料信息整合到表Order\_information。

工单编制，根据输入的单包重量和包数，将ERP生产订单拆分成不同重量的制令单或配比单（可以分多次拆解），大批次命名规则为6#+年月+流水号，可调用自定义函数getID生成，小批次命名规则为大批次+包号。

工单确认，更新标志位；工单关闭或废弃，调用自定义函数CloseProductOrder。

### 工单排产

#### 数据库表

|  |  |
| --- | --- |
| materialObj | 工单生产所需物料信息表 |
| put\_realTime | 投料信息表 |
| get\_realTime | 收料信息表 |

#### 前端界面

图形用户界面, 应用程序

描述已自动生成

图表 3 工单排产

图形用户界面, 表格

描述已自动生成

图表 4 物料排产

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图表 5 制令单/配比单信息

图形用户界面, 文本, 电子邮件

描述已自动生成

图表 6 月度生产计划

图形用户界面, 应用程序, 表格

描述已自动生成

图表 7 计划生产实况

#### 开发思路

**物料排产：**物料排产前先查询是否已经排过一部分物料，即查询materialObj表中是否有该工单的排产数据，若没有则需要查询OrderRESBData表获取BOM信息，根据BOM信息中的物料号查询线边仓storage\_batch表中质检合格的原材料。为每个小批次的原材料选择预排产重量，更新线边仓库存中对应小批次的预排产重量。每次进行物料排产的时候，若该批次的预排产重量等于该批次实际重量，则不允许再使用该物料进行排产。对于已经排产过的物料，可以点击删除，释放预排产的重量，用于其他工单的排产。每次点击指定计划后，会将原来的批次计划全部删除，将新的物料批次计划存入materialObj中。并判断是否BOM 中所有的物料都已排好重量，若是则更新工单的物料排产标志位。

**时间排产：**确定工单的计划开始时间和计划结束时间。

**批次生产：**对已完成物料排产和时间排产的工单，生成投料和收料批次计划，根据工单中的ERP工作中心，在basic\_info站点信息表中获取投收料站点的范围。

**工单关闭/取消：**删除put\_realtime和get\_realtime以及materialObj中的该工单数据，还原工单的排产状态标志位。

**工单结单：**在SAP系统中关闭生产订单，MES中无需操作。

#### 业务流程

对需要排产的工单，先做确认。然后为工单所需的每种原材料指定对应的线边仓库存批次和重量，物料排产后，进行时间排产以及最后生成计划投料收料批次。已经生成排产计划的工单可以点击计划重排

### 库存信息

#### 数据库表

|  |  |
| --- | --- |
| Storage\_batch | 六号线边仓 |
| WMS\_batch | 六号立库（MES） |
| V\_Inventory\_All | 六号立库（WMS） |
| V\_storage | 四号线边仓 |
| V\_GS\_List | 四号立库 |

#### 前端界面

#### 业务流程

#### 开发思路

查询立库和线边仓的库存。

## 调度模块

### 实时调度

**质检推送：**质检推送的手动接口，根据大批次或小批次（只有对于ERP的原材料质检推送来说才会按大批次推送合格/不合格重量）推送质检结果至SAP、MES、WMS系统中，分别走QC\_determination，WMS\_Qcinfo的接口。

**销售成品出库：**按照MES系统指定的出库单，查询出库单中所绑定立库质检合格的物料，不生成出库单，而是直接调用WMS\_outTask函数生成一条出库指令。

**任务取消：**对于任务状态为初始的指令才能取消，一次性将所有需要取消的指令全部插入到指令取消接口表task\_cancel中。

### 历史调度

查询已完成的出入库指令，即查询任务状态为80、90、91的。

### 异常查询

查询执行异常的出入库指令，即查询任务状态为90的。

## 仓库模块

### 原材料入库

#### 数据库表

|  |  |
| --- | --- |
| materialMasterData | ERP物料基础信息 |
| MATDOC | 原材料采购订单接口 |
| Storage\_batch | 线边仓库存 |
| Materialinformation | 物料基础信息 |

#### 业务流程

#### 开发思路

选择采购订单对原材料进行批次的录入，设置单包重量和包数，生成小批次号，完成原材料入库操作。

获取原材料的ERP大批次号，根据查询storage\_batch已录入的最大包号，在后面接着生成新的小批次号，小批次号命名规则为ERP大批次号+包号。若勾选打印按钮，则将打印标志位置为1，KS后台服务会将这些数据转存到条码打印表中，打印机自动打印条码。勾选质检后，调用质检任务生成函数QCtaskGenrate，生成原材料质检任务。

### 物料信息维护

维护表Materialinformation信息。

### 成本中心退料

#### 数据库表

|  |  |
| --- | --- |
| storage\_batch | 线边仓库存 |
| storage\_task | 仓库任务单据 |
| Batch\_PutTask | ERP调度接口 |

#### 业务流程

生产上剩余的原材料，或者发现有不符合生产要求的原料，需要制定退库单，退回到原材料仓库中，首先制单，审核，确认后，线边仓调度出库，退回到原材料线边仓中。

#### 开发思路

首先生成退料单，单号调用函数getID自动生成，其他信息需手动录入。输入后保存到表storage\_task中，任务状态此时为未完成并。现场人员扫码入库后，再线边仓调度表中插入调度记录，触发器将执行结果回传给ERP的Batch\_PutTask接口，以及插入MES的storage\_batch表中。

### 领料单

#### 数据库表

|  |  |
| --- | --- |
| productOrder\_realTime | 实时工单表 |
| materialObj | 原材料批次信息表 |
| storage\_batch | 线边仓库存表 |
| Theline\_Task | 线边仓调度指令表 |
| Batch\_PutTask | ERP调度回传接口 |

#### 业务流程

生产工单在确认、审核、物料排产、时间排产完成后你，方可指定领料单，对于已经领料或已经制定过领料单的工单不允许重复制单。制定完领料单后，需要进行审核。审核通过后，现场人员根据领料单，在原材料线边仓找到对应的小批次的物料，扫码领料到领料单上指定的车间线边仓。领料单信息可以打印。

#### 开发思路

**领料单制定：**根据工单表productOrder\_realTime中已确认、审核、物料排产、时间排产的工单，查询该工单的原材料信息materialObj表，获取投料的物料信息（物料代码，投料重量，领料车间即工作中心所在的车间），生产领料单插入到storage\_task表中。同时将库存stroage\_batch中的该小批次的占用标志位置为1.

**领料单审核：**更新storage\_batch审核标志位。

**领料单执行：**线边仓扫码出库后，往线边仓调度表Theline\_Task中插入一条数据，该表中的触发器会对插入的小批次进行校验，若该小批次属于领料单，则更新领料单状态为已完成，同时更新该小批次的库位信息，给ERP的Batch\_PutTask表回传一个311类型的调度信息。

### 退库单

#### 数据库表

|  |  |
| --- | --- |
| storage\_batch | 线边仓库存表 |
| Theline\_Task | 线边仓调度指令表 |
| Batch\_PutTask | ERP调度回传接口 |

#### 业务流程

当原材料领料后，出现异常需要退回到原材料仓库，此时需要开一个退库单，将其退回到原材料仓。

当成品到成品发货仓时，发现有包装破损或有异常问题，需要开退库单，由制单人指定终点仓位。

#### 开发思路

同领料单逻辑。

### 成品入库

#### 数据库表

|  |  |
| --- | --- |
| printLabel | 条码打印信息表 |
| printCode | 条码ID表 |
| batch\_SL | 现场收料表 |
| storage\_batch | 线边仓库存表 |
| WMS\_inrequire | WMS入库请求接口 |
| storage\_task | 仓库任务单表 |
| WMS\_feedback | WMS指令执行反馈接口 |
| Batch\_PutTask | SAP生产调度接口表 |
| WMS\_batch | 立库库存表 |

#### 业务逻辑

成品需要以收料新增的方式生成成品的小批次信息，并打印条码，从产线上下来的成品，贴上对应的条码，然后扫码收入线边仓中，线边仓的成品如果要入立库，则在立库的入口处扫码发送入库请求，MES收到请求后会返回该批次的相关信息（若没有相关信息则不允许入立库）。

#### 开发思路

**成品条码生成：**按照工单号作为大批次，按顺序生成包号，以大批次号+包号作为小批次号。生成小批次后，将小批次信息存入printLabel表中，其中条码ID从表printCode中获取。KS后台服务会将该表的数据导出到指定文件夹。生成的条码会贴在下线的成品上。

**生产收料：**成品下线后，触摸屏扫码标签后，会往batch\_SL表中插入信息，校验后该表触发器会将成品信息插入线边仓库存storage\_batch中。

**成品入库：**立库扫码后，会通过接口WMS\_inrequire发送一条入库请求，该表触发器对插入信息进行校验（查询storage\_task看是否已经生成成品入库单，若有则将该批次信息回写到该条记录后面的字段中，若没有则返回错误信息）。WMS系统收到返回的成品信息后执行入库操作，入库结果通过接口WMS\_feedback回传给MES，该接口的触发器会根据回传结果，更新storage\_task成品出库单的标志位，给SAP接口Batch\_PutTask回传生产入库信息，从我们线边仓库存storage\_batch中删除该批次信息，插入到立库库存WMS\_batch中。

### 物权转移

#### 数据库表

#### 业务流程

物料在不同仓位之间的移动，除去特定的业务（入领料，退库等），其他都属于物权转移。物权转移包含六号工厂内的转移以及六号四号工厂之间的转移。同样是先制定物权转移单，然后审核，现场线边仓扫码出库，更新库位信息（若是转到四号则是删除六号库存，插入四号库存），回传ERP。需要注意的时，物权转移只能针对之间合格的物料。四号和六号不同车间之间的转移没有限制。

#### 开发思路

首先根据线边仓库存表storage\_batch创建物权转移单，即往storage\_task表中插入一条任务，审核后更新该表审核标志位，至此制单完成。

线边仓扫码后会往Theline\_Task插入一条记录，该表触发器会查询storage\_task是否有该小批次的任务单，若有物权转移单，则判断是否为跨车间（即六号到四号）的单据，若是则需要要从六号线边仓库存storage\_batch中删除该条线边仓的记录，然后插入到四号的storage\_batch,，否则只需要更新自身六号storage\_batch表的仓位字段ERPposition即可。

### 成品出库

#### 数据库表

|  |  |
| --- | --- |
| WMS\_outstore\_order | WMS出库请求接口 |

#### 业务逻辑

首先在MES系统上制作成品出库单，WMS系统扫码后，物料出立库进入线边仓。

#### 开发思路

首先在MES系统生成成品出库单，即插入storage\_batch一条数据，同时通过出库接口WMS\_outstore\_order发送批次的出库请求，WMS接收到出库请求后执行出库动作，完成后通过接口WMS\_feedback回传给MES，该接口的触发器会根据回传结果，更新storage\_task成品出库单的标志位，给SAP接口Batch\_PutTask回传生产入库信息，往我们线边仓库存storage\_batch中插入该批次信息，删除立库库存WMS\_batch。

## APP模块

### 工艺参数

暂未开发

## 质检模块

### 销售退货

#### 数据库表

|  |  |
| --- | --- |
| SalesDelivery\_history | SAP销售订单接口 |
| finished\_Product\_use | 销售点货表 |
| Information\_return | 销售上下架接口信息 |

#### 业务逻辑

根据退货单，以收料新增的方式导入退货信息，生成新的小批次。回传SAP。

#### 开发思路

首先查询SAP的销售订单接口SalesDelivery\_history(字段LFART为LR或NCR的才是退货单)，以收料新增的方式（类似原材料入库和成品入库），根据 SalesDelivery\_history中的ERP大批次生成小批次。录入后插入到线边仓库存，同时回传至SAP的Information\_return。

### 检测项维护

#### 数据库表

|  |  |
| --- | --- |
| QC\_testitem | 质检项 |

#### 业务逻辑

生成质检任务前，需要提前配置该物料号的质检项，根据质检项的标准生成质检任务及质检结果。

#### 开发思路

每个质检项都分为IQC、IPQC、OQC，分别对应三种质检任务。质检项有几个重要标志位：质检项自动判定标识、物料号自动判定标识、是否必须。其与质检项结果、质检任务结果如下：

首先质检项结果根据质检项自动判定标识来决定是否自动生成，若为1则根据检测值、上下限自动生成质检项结果；

其次质检任务结果生成条件是，若物料自动判定标识为1，则说明该质检任务需要自动生成质检任务结果（一个质检任务只有一个物料号），此时我们再去看是否所有是否必须标志位为1的质检项的质检项结果（质检项结果根据上面那一步自动生成），若所有必须的质检项结果都为合格（不能为待检或空），则自动判定该质检任务结果为合格，否则判定为不合格。

### 异常记录

#### 数据库表

|  |  |
| --- | --- |
| QC\_realTimetask | 质检实时表 |

#### 业务逻辑

对于异常的质检任务，可以申请走MRB流程，生成MRB任务单。

#### 开发思路

在质检任务界面，点击MRB申请，往QC\_realTimetask表中插入一条任务类型为6的任务。

## 业务自定义函数说明

### 销售发货

#### 数据库表

|  |  |
| --- | --- |
| SalesDelivery\_history | SAP销售订单接口 |
| finished\_Product\_use | 销售点货表 |
| Information\_return | 销售上下架接口信息 |

#### 业务逻辑

首先在MES系统上进行点货，为SAP销售订单绑定质检合格且未锁定的物料。执行销售出库，直到该发货单所有的物料全部发完，才能通过手动回传接口回传至SAP

#### 开发思路

首先查询SAP的销售订单接口SalesDelivery\_history(字段LFART为LF或ZLF或NLCC的才是发货单)，对该销售订单进行点货，查询storage\_batch和WMS\_batch（只能查询质检合格以及未被占用和未锁定的），根据选择的小批次插入到点货表finished\_Product\_use中，同时更新销售订单接口SalesDelivery\_history中的已点货重量，一个销售订单可以分多次点货，点货详情界面，可以将已点货的小批次取消点货（即从finished\_Product\_use中删除，同时更新SalesDelivery\_history中的已点货重量）。点击出库后，若该批次在线边仓。则直接删除该库存storage\_batch，更新finished\_Product\_use中的出库标志位，只有发货单中全部的小批次都出库，才能点击手动回传按钮，回传SAP中的Information\_return销售上下架接口表。

### 原材料/制程/成品质检

#### 数据库表

|  |  |
| --- | --- |
| QC\_realTimetask | 质检实时表 |
| QC\_result | 质检结果表 |
| Batch\_SL | 收料表 |

#### 业务逻辑

原材料收料新增的时候，勾选生成质检任务，系统会自动生成原材料的质检任务。

半成品/成品在收料时候，会自动生成之间任务。

执行质检任务，可选择批量合格/不合格，此时直接对质检任务进行判定。若挨个进行质检的话，需要输入质检值或质检结果（若两个都不输入则不能选择自动判定，只能选择合格或不合格），这时候系统会根据事先配置好的质检项自动判定标志，来自动生成质检任务的结果。

生成结果后，会将质检结果回传至sap,mes,wms。

#### 开发思路

**原材料质检任务生成**：收料新增处调用QCtaskGenrate函数，传入质检信息，按小批次生成质检任务；

**成品/半成品质检任务生成：**batch\_SL触发器在收料时该表插入一条数据，该触发器根据插入的小批次号等信息，查询QC\_tsetItem表获取该物料号的质检项，同时往QC\_realTimetask质检任务表中插入一条质检任务。

**执行质检任务：**输入检测值和检测结果（可只输入一个，若不选择自动判定可都不输入，直接根据任务结果生成质检项的结果），调用质检结果生成函数QCresulctGenerate，生成质检结果并将结果更新到QC\_realTimetask和QC\_result中。

**质检结果回传：**在质检结果生成的函数中，会调用QCresultBack函数将质检结果回传至wms和mes，即更新storage\_batch的质检结果和通过接口wms\_QCinfo将质检结果分别回传至mes和wms 。

对于原材料而言，回传sap需要大批次下所有的小批次质检结果都生成后才能一次性以大批次回传，而且只回传该大批次的合格不合格数量（不具体到哪一个小批次）。该逻辑写在数据库的函数ERP质检回传中，通过SQLServer自动任务定时执行，该函数会查询MATDOC\_history原材料采购订单接口获取原材料大批次的重量，然后查询QC\_realTimetask获取合格数量和不合格数量，只有当合格数量加上不合格数量等于大批次总重量的时候，才会回传至SAP的QC\_determination质检回传接口，向该表插入两条数据（即合格一条，不合格一条，若重量为0则不需要插入，就是说若全部合格只会插入一条数据）。质检改判同上逻辑。

对于成品/半成品而言，以小批次回传SAP，其逻辑写在QC\_realTimetask触发器中，直接往QC\_determination中插入小批次的质检结果，一个小批次插入一条数据。质检改判逻辑同上。

质检改判只能改判一次。

### 质量统计

#### 数据库表

|  |  |
| --- | --- |
| QC\_realTimeTask | 质检任务表 |
| batch\_SL | 收料表 |
| batch\_TL | 投料表 |

#### 业务逻辑

按照大批次统计质检结果，以图标和柱状图的形式展示。

#### 开发思路

略。

### 收料

#### 数据库表

|  |  |
| --- | --- |
| Batch\_SL | 收料表 |

#### 业务逻辑

现场收料前需要扫描吨包上的一维码，校验成功后才可收料。

#### 开发思路

插入batch\_SL一条数据。

### 质量追溯

#### 数据库表

同上。

#### 业务逻辑

正向追溯按照工单号进行查询，查询投收料记录，逆向追溯则是按销售订单号查询投收料记录。

#### 开发思路

联查QC\_realTimeTask和batch\_SL、batch\_TL，根据工单号，小批次号相关联。

### 设备维修

#### 数据库表

#### 业务逻辑

#### 开发思路

### FeedAndReceipt（投收料批次生成）

#### 描述

\* @description: 投料/收料 计划/实际 批次生成，投料按原材料小批次投，收料按工单（mes大批次收料）

\* @param {[string]} jobID - 制令单号/配比单号（工单号,MES成品批次号）

\* @param {number} rule - 投料收料批次生成规则 1：按ERP工作中心生成 2：按工序生成 3：按设备生成

\* @return {\*}

#### 说明

对于已完成时间排产，物料排产的工单而言，才可以调用此函数生成投收料批次信息。

### 投料

#### 数据库表

|  |  |
| --- | --- |
| Batch\_TL | 投料表 |

#### 业务逻辑

现场投料前需要扫描吨包上的一维码，校验成功后才可投料。

#### 开发思路

插入batch\_TL一条数据。

### CloseProductOrder（工单关闭/废弃函数）

#### 描述

**\* @description:** 工单关闭/废弃函数，更新工单信息表productOrder\_realTime的标志位，并将工单信息表productOrder\_realTime、投料实时表put\_realTime、收料实时表get\_realTime 存如历史表中，并删除实时表中的数据。

**\* @param {object[string]}** jobID - 制令单号/配比单号（工单号,MES批次号）

**\* @param {number}** mode - 0：完成后结单 1：废弃（取消）

**\* @return {boolean}**

#### 说明

工单生产完成后调用该函数，会将生产过程中所用到的数据全部存入历史表中，若是删除/废弃工单，则是直接删除该工单。

### OverQC（质检改判函数）

#### 描述

\* @description: 质检改判，改判结果回传ERP、MES、WMS，其中ERP按照大批次重量回传；调用该函数之前需要判断改判结果是否和原结果不一致，函数默认认为改判结果不一致

\* @param {object[]} QCtaskid - 质检任务单号数组

\* @param {number} overQCresult - 改判结果 0:不合格，1:合格

\* @return {boolean} 回传结果

#### 说明

改判只能执行一次，不支持多次改判（主要是SAP不支持）

### rePlan（计划重排）

#### 描述

**\* @description:** 计划重排

**\* @param {object[string]}** jobIDArr 工单数组

**\* @return { boolean }**

#### 说明

对于为投料的工单，允许重新排产。

### QCtaskGenrate（质检任务生成函数）

#### 描述

文本

描述已自动生成

#### 说明

实现生成原材料质检任务，半成品取样任务，半成品质检任务，成品取样任务，成品质检任务函数

### WMS\_outTask（WMS出库任务指令生成）

#### 描述

\* @description: 手动生成一条出库任务下发至WMS出库指令接口表中

\* @param {object[object]} outInfo - 出库信息 [{jobIDS: '小批次号', endLocation: '结束地址'}, ...]

\* @return {boolean}

#### 说明

手动生成一条出库任务下发至WMS出库指令接口表中

### 常用函数（数据集操作函数，查询函数等）

 \* @Author: EDwin

 \* @Date: 2022-02-21 11:34:51

 \* @LastEditors: EDwin

 \* @LastEditTime: 2022-03-11 16:33:42

 \*/

/\*\*

 \* @type: KP自定义函数

 \* @description 对数据集进行筛选

 \* @param {object} dataSet 数据集

 \* @param {object[object]} filter - 筛选条件[{field: '字段名1', value: '字段值1', match: '='}, {field: 字段名2, value: '字段值2,字段值3,字段值4,...', match: 'in'}, ...]，匹配条件可为 =、!=、<、>、<=、>=、in、not in、like（模糊查询），此处条件连接均未AND，若条件连接未OR，则使用两次该函数，将两次执行的结果合并(concat)成一个数据集即可

 \* @return {object} 筛选后的数组对象

 \*/

function dataFilter(dataSet, filter) {

    try {

        if (filter.length === undefined) throw new Error('[dataFilter]  传入参数filter类型错误！');

        var indataSet = dataSet;

        for (var j = 0; j < filter.length; j++) {

            var len = indataSet.length;

            if (filter[j].value == '全部' || filter[j].value === '') {

                continue;

            } else {

                switch (filter[j].match) {

                    case '=':

                        for (var i = len - 1; i >= 0; i--) {

                            if (indataSet[i][filter[j].field] != filter[j].value) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case '!=':

                        for (var i = len - 1; i >= 0; i--) {

                            if (indataSet[i][filter[j].field] == filter[j].value) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case '>':

                        for (var i = len - 1; i >= 0; i--) {

                            if (indataSet[i][filter[j].field] <= filter[j].value) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case '>=':

                        for (var i = len - 1; i >= 0; i--) {

                            if (indataSet[i][filter[j].field] < filter[j].value) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case '<':

                        for (var i = len - 1; i >= 0; i--) {

                            if (indataSet[i][filter[j].field] >= filter[j].value) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case '<=':

                        for (var i = len - 1; i >= 0; i--) {

                            if (indataSet[i][filter[j].field] > filter[j].value) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case 'like':

                        for (var i = len - 1; i >= 0; i--) {

                            var reg = eval('/' + filter[j].value + '/ig');

                            if (reg.test(indataSet[i][filter[j].field]) === false) {

                                indataSet.splice(i, 1);

                            }

                        }

                        continue;

                    case 'in':

                        for (var i = len - 1; i >= 0; i--) {

                            var value = filter[j].value.split(',');

                            var flag = 0;

                            value.forEach(function (item) {

                                if (item == indataSet[i][filter[j].field]) flag = 1;

                            });

                            if (flag === 0) indataSet.splice(i, 1);

                        }

                        continue;

                }

            }

        }

        return indataSet;

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 将数据集插入到数据库中

 \* @param {object[object]} dataSet - 需要插入的数据集

 \* @param {string} dataBaseName - 数据库名称

 \* @return {boolean}

 \*/

async function SqlInsert(dataSet, dataBaseName) {

    try {

        if (dataSet.constructor === Array && dataSet.length > 0) {

            var field = [];

            for (var key in dataSet[0]) field.push(key);

            var sqlStr = `INSERT INTO ${dataBaseName} (${field.join(',')}) VALUES `;

            dataSet.forEach(function (item) {

                var value = [];

                for (var key in item) {

                    value.push("'" + item[key] + "'");

                }

                sqlStr += `(${value.join(',')}),`;

            });

            sqlStr = sqlStr.substring(0, sqlStr.length - 1);

            var res = await toDataSet('$System.BTR', sqlStr);

            if (res === true) {

                return true;

            } else {

                return false;

            }

        } else {

            throw new Error('[SqlInsert] 传入的数据集为空或格式错误！');

        }

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 将数据集转换成字典MAP

 \* @param {object[string]} primaryKey - 主键名数组

 \* @param {object[object]} dataSet - 数据集

 \* @return {map} 返回字典，若失败则返回false

 \*/

function toMap(primaryKey, dataSet) {

    if (primaryKey.length > 0) {

        var map = {};

        dataSet.forEach(function (item) {

            var key = '';

            for (var i = 0; i < primaryKey.length; i++) {

                key += item[primaryKey[i]];

            }

            map[key] = item;

        });

        return map;

    } else {

        return fales;

    }

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 将带JSON字符串的数据集转换成能直接在数据网格展示的数据集

 \* @param {object[object]} dataSet - 需要更新的数据集

 \* @param {object[]} field - JSON字符串字段名 ['私有成员对象字段名1'， ...] (其中私有成员对象字段值格式化之后应该是一个对象而不是数组对象！)

 \* @return {object[object]} 成功则返回数据集，失败返回false

 \*/

function JSON\_to\_dataSet(dataSet, field) {

    try {

        var indataSet = dataSet;

        for (var i = 0; i < indataSet.length; i++) {

            field.forEach(function (item) {

                if (indataSet[i][item] === undefined) {

                    throw new Error('[JSON\_to\_dataSet]  私有成员对象名' + item + '不存在！');

                } else if (indataSet[i][item] == '' || indataSet[i][item] == null) {

                    //若私有成员对象字段为空，则直接删除

                    delete indataSet[i][item];

                } else {

                    //判断私有成员对象字段是否为正确的JSON格式字符串

                    if (!isJSON(indataSet[i][item])) {

                        //throw new Error('[JSON\_to\_dataSet]  第' + i + '个对象JSON字符串格式不正确！' + indataSet[i][item]);

                        delete indataSet[i][item];

                    } else {

                        var obj = JSON.parse(indataSet[i][item]);

                        delete indataSet[i][item];

                        for (var key in obj) {

                            if (indataSet[i][key] != undefined) throw new Error('[JSON\_to\_dataSet]  第' + i + '个对象的' + key + '键名重复！');

                            indataSet[i][key] = obj[key];

                        }

                    }

                }

            });

        }

        return indataSet;

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 将数据集转换成带私有成员对象JSON字符串的数据集用于直接存入数据库

 \* @param {object[object]} dataSet - 需要更新的数据集

 \* @param {object[string]} field - JSON字符串字段名 [{field: 'privateObj', key: ['字段名1', '字段名2', ...]}, ...]

 \* @return {object[object]} 成功则返回数据集，失败返回false

 \*/

function dataSet\_to\_JSON(dataSet, field) {

    try {

        var indataSet = dataSet;

        for (var i = 0; i < indataSet.length; i++) {

            field.forEach(function (item) {

                var arrKey = item.key;

                var obj = {};

                arrKey.forEach(function (val) {

                    obj[val] = indataSet[i][val];

                    delete indataSet[i][val];

                });

                indataSet[i][item.field] = JSON.stringify(obj);

            });

        }

        return indataSet;

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 数据去重，代替sql中的distinct

 \* @param {object[object]} dataSet - 数据集数组对象

 \* @param {object[]} field - 需要去重的字段名称['字段名', ...] 只有当所有字段值都一样时才会认为是重复项而去掉

 \* @return {object[]} 数组对象

 \*/

function sqlDistinct(dataSet, field) {

    var map = {};

    var resData = [];

    for (var i = 0; i < dataSet.length; i++) {

        var key = '';

        field.forEach(function (item) {

            key += dataSet[i][item];

        });

        if (map[key] === undefined) {

            map[key] = 1;

            resData.push(dataSet[i]);

        }

    }

    return resData;

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 聚合函数，分组统计，实现GROUP BY SUM功能

 \* @param {object[object]} dataSet - 数据集

 \* @param {object[]} sumField - 需要求和名且保留展示的字段

 \* @param {object[]} groupField - 需要分组的字段名（GROUP BY后的字段名称）

 \* @return {object[]} 数组对象

 \*/

function sqlGroupby(dataSet, sumField, groupField) {

    try {

        var map = {};

        var resData = [];

        for (var i = 0; i < dataSet.length; i++) {

            var key = '';

            groupField.forEach(function (item) {

                if (dataSet[i][item] === undefined) throw item + '字段名未定义！';

                key += dataSet[i][item];

            });

            if (map[key] === undefined) {

                if (sumField.length <= 0) throw '求和数组参数不能为空！';

                var obj = {};

                groupField.forEach(function (item) {

                    obj[item] = dataSet[i][item];

                });

                sumField.forEach(function (item) {

                    obj[item] = dataSet[i][item];

                });

                map[key] = obj;

            } else {

                sumField.forEach(function (item) {

                    map[key][item] += dataSet[i][item];

                });

            }

        }

        for (var key in map) {

            resData.push(map[key]);

        }

    } catch (e) {

        console.log(e);

        return;

    } finally {

        return resData;

    }

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 数据联合，实现inner JOIN功能

 \* @param {object[]} outDataSetLeft - 左表数据集

 \* @param {object[]} outDataSetRightArr - [ 右表数据集1, 右表数据集2, 右表数据集3, ... ]

 \* @param {object[]} fieldArr - 联合的字段[['左表1字段', '右表1字段'], ['左表2字段', '右表2字段'], ['左表3字段', '右表3字段'], ...] (ON左右两侧的字段名)

 \* @return {object[]} 数组对象

 \*/

function sqlInnerjoin(outDataSetLeft, outDataSetRightArr, fieldArr) {

    //存中间变量

    var dataSetLeft = outDataSetLeft;

    var dataSetRightArr = outDataSetRightArr;

    if (dataSetRightArr.length !== fieldArr.length) throw new Error('[sqlLeftjoin] 数据集数量和联合字段数量不一致！');

    if (dataSetRightArr != '' && fieldArr != '') {

        dataSetRight = dataSetRightArr[0];

        field = fieldArr[0];

        var leftField = [];

        var rightField = [];

        //保存左右数据集的键名

        for (var key in dataSetLeft[0]) {

            leftField.push(key);

        }

        for (var key in dataSetRight[0]) {

            rightField.push(key);

        }

        var resData = [];

        outside: dataSetLeft.forEach(function (leftItem) {

            inside: for (var i = 0; i < dataSetRight.length; i++) {

                //若左表和右表字段匹配则存入左表数据（ON后面的联合条件），否则存入null

                if (leftItem[field[0]] == dataSetRight[i][field[1]]) {

                    flag = 1;

                    rightField.forEach(function (key) {

                        //若字段名不重复，则直接赋值，否则字段名+’1‘后再赋值

                        if (leftItem[key] === undefined) {

                            leftItem[key] = dataSetRight[i][key];

                        } else {

                            leftItem[key + '1'] = dataSetRight[i][key];

                        }

                    });

                    resData.push(leftItem);

                } else {

                    continue inside;

                }

            }

        });

        dataSetRightArr.splice(0, 1);

        fieldArr.splice(0, 1);

        dataSetLeft = sqlInnerjoin(resData, dataSetRightArr, fieldArr);

    } else {

        return dataSetLeft;

    }

    return dataSetLeft;

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 数据联合，实现LEFT JOIN功能

 \* @param {object[]} outDataSetLeft - 左表数据集

 \* @param {object[]} outDataSetRightArr - [ 右表数据集1, 右表数据集2, 右表数据集3, ... ]

 \* @param {object[]} fieldArr - 联合的字段[['左表1字段', '右表1字段'], ['左表2字段', '右表2字段'], ['左表3字段', '右表3字段'], ...] (ON左右两侧的字段名)

 \* @return {object[]} 数组对象

 \*/

function sqlLeftjoin(outDataSetLeft, outDataSetRightArr, fieldArr) {

    //存中间变量

    var dataSetLeft = outDataSetLeft;

    var dataSetRightArr = outDataSetRightArr;

    if (dataSetRightArr.length !== fieldArr.length) throw new Error('[sqlLeftjoin] 数据集数量和联合字段数量不一致！');

    if (dataSetRightArr != '' && fieldArr != '') {

        dataSetRight = dataSetRightArr[0];

        field = fieldArr[0];

        var leftField = [];

        var rightField = [];

        //保存左右数据集的键名

        for (var key in dataSetLeft[0]) {

            leftField.push(key);

        }

        for (var key in dataSetRight[0]) {

            rightField.push(key);

        }

        var resData = [];

        outside: dataSetLeft.forEach(function (leftItem) {

            var flag = 0; //看右表中有没有和左表某一行匹配的行的标志位

            inside: for (var i = 0; i < dataSetRight.length; i++) {

                //若左表和右表字段匹配则存入左表数据（ON后面的联合条件），否则存入null

                if (leftItem[field[0]] == dataSetRight[i][field[1]]) {

                    flag = 1;

                    rightField.forEach(function (key) {

                        //若字段名不重复，则直接赋值，否则字段名+’1‘后再赋值

                        if (leftItem[key] === undefined) {

                            leftItem[key] = dataSetRight[i][key];

                        } else {

                            leftItem[key + '1'] = dataSetRight[i][key];

                        }

                    });

                    resData.push(leftItem);

                } else {

                    continue inside;

                }

            }

            if (flag == 0) {

                //右表中没有与左表某一行匹配的行，需要键右表中的字段加到左表中并赋值为null

                rightField.forEach(function (key) {

                    if (leftItem[key] === undefined) {

                        leftItem[key] = null;

                    } else {

                        leftItem[key + '1'] = null;

                    }

                });

                resData.push(leftItem);

            }

        });

        dataSetRightArr.splice(0, 1);

        fieldArr.splice(0, 1);

        dataSetLeft = sqlLeftjoin(resData, dataSetRightArr, fieldArr);

    } else {

        return dataSetLeft;

    }

    return dataSetLeft;

}

/\*\*

 \* @type: KP自定义函数

 \* @description: 排序，实现ORDER BY功能

 \* @param {object[object]} dataSet - 数据集

 \* @param {object[object]} field - 排序字段及规则 [{name: '字段名1', rule: 'ASC'}, {name: '字段名2', rule: 'DESC'}]

 \* @return {object[]} 数组对象

 \*/

function sqlOrder(dataSet, field) {

    var resData = [];

    for (var i = 0; i < field; i++) {

        resData = dataSet.sort(function (a, b) {

            if (field[i].rule === 'ASC') {

                return a[field[i].name] > b[field[i].name] ? 1 : a[field[i].name] < b[field[i].name] ? -1 : 0;

            } else if (field[i].rule === 'DESC') {

                return a[field[i].name] > b[field[i].name] ? -1 : a[field[i].name] < b[field[i].name] ? 1 : 0;

            }

        });

    }

    return resData;

}

/\*\*

 \* @description: 遍历指定路径下的所有文件

 \* @param {string} filePath - 指定路径

 \* @param {string} keyWord - 指定文件中的关键字

 \* @param {string} extname - 指定文件扩展名(.xml)

 \* @param {string} exclude - 排除文件中的关键字

 \* @param {string}

 \* @return {\*} 文件路径+文件名+后缀 数组

 \*/

function fileDisplay(filePath, keyWord, extname, exclude) {

    try {

        var fs = require('fs');

        var path = require('path');

        //根据文件路径读取文件，返回文件列表

        fs.readdir(filePath, function (err, files) {

            if (err) throw '读取文件夹错误！';

            //遍历读取到的文件列表

            files.forEach(function (filename) {

                //获取当前文件的绝对路径

                var filedir = path.join(filePath, filename);

                //根据文件路径获取文件信息，返回一个fs.Stats对象

                fs.stat(filedir, function (eror, stats) {

                    if (eror) throw '获取文件stats失败';

                    var isFile = stats.isFile(); //是文件

                    var isDir = stats.isDirectory(); //是文件夹

                    if (isFile) {

                        var suffix = path.extname(filedir); //获取文件后缀名

                        if (extname === undefined || suffix == extname) {

                            fs.readFile(filedir, function (err, data) {

                                if (err) throw err;

                                data = data.toString();

                                var patt = new RegExp(keyWord, 'i');

                                if (patt.test(data)) console.log(filedir);

                            });

                        }

                    }

                    if (isDir) {

                        fileDisplay(filedir, keyWord, extname); //递归，如果是文件夹，就继续遍历该文件夹下面的文件

                    }

                });

            });

        });

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @description: 按控件内容筛选函数

 \* @param {object[object]} OCX - 控件名称及对应的字段名 [{name: 'Combobox1', field: 'taskID', match: '='}, ...] 匹配条件可为 =、!=、<、>、<=、>=、like（模糊查询）

 \*                               若需要对单选框的文本内容进行筛选，则name属性需为 '控件名称\_text' 这种形式

 \*                                若需要对单选框的索引进行筛选，则name属性需为 '控件名称\_index' 这种形式

 \* @param {object[object]} dataSet - 数据集

 \* @param {object[object]} PageName - 画面名称

 \* @return {object[object]} 筛选后的数据集

 \*/

function OcxFiltering(OCX, dataSet, PageName) {

    try {

        var condition = []; //筛选条件数组对象

        for (var i = 0; i < OCX.length; i++) {

            //控件名字

            var OCXName = OCX[i].name;

            var obj = {

                field: OCX[i].field,

                match: OCX[i].match,

            };

            var name1 = $System.$PicManager[PageName].$Children[OCXName].prototypeName;

            //判断控件类型  Combobox Textbox UIRadioButtonGroup DateBox DateTimeBox

            switch (name1) {

                case 'Combobox':

                    obj.value = $System.$PicManager[PageName].$Children[OCXName].GetCurrentText();

                    break;

                case 'Textbox':

                    obj.value = $System.$PicManager[PageName].$Children[OCXName].Text;

                    break;

                case 'UIRadioButtonGroup':

                    if (OCXName.indexOf(text) != -1) {

                        obj.value = $System.$PicManager[PageName].$Children[OCXName].SelectedText;

                    } else if (OCXName.indexOf(index) != -1) {

                        obj.value = $System.$PicManager[PageName].$Children[OCXName].SelectedIndex;

                    }

                    break;

                case 'DateBox':

                    obj.value = $System.$PicManager[PageName].$Children[OCXName].Value;

                    break;

                case 'DateTimeBox':

                    obj.value = $System.$PicManager[PageName].$Children[OCXName].Value;

                    break;

                default:

                    return;

            }

            condition.push(obj);

        }

        return dataFilter(dataSet, condition);

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @description: 创建多级目录（同步函数）以递归的方式

 \* @param {string} dirname - 绝对路径

 \* @return {boolean} 若成功或已存在则返回true

 \*/

function mkDirsSync(dirname) {

    const fs = require('fs');

    const path = require('path');

    if (fs.existsSync(dirname)) {

        return true;

    } else {

        if (mkDirsSync(path.dirname(dirname))) {

            fs.mkdirSync(dirname);

            return true;

        }

    }

}

/\*\*

 \* @description: post方式调用KC请求式脚本

 \* @param {string} scriptName - 请求式脚本名称（也就是函数名）

 \* @param {object} InParam - 传入参数结构体

 \* @param {function} callback - 回调函数

 \* @return {\*}

 \*/

function KCrequest(scriptName, InParam, callback) {

    var host = 'http://192.168.3.123:1230';

    var api = '/' + scriptName; //KC请求式计算的脚本名称

    var type = 'post';

    var postType = 'JSON';

    var kcdata = {

        InParam: InParam, //传入到KC的参数

        RequestType: 'normal',

        RequestID: $Function.GUID(),

    };

    var iclient = KMClientInterface.getInstance();

    iclient.httpRequestExec(host, api, type, kcdata, postType, function (resData) {

        callback(resData);

    });

}

/\*\*

 \* @description: 获取当前日期函数

 \* @return {string} 返回格式为"2021-01-11"

 \*/

function GetDataFunc() {

    var date = new Date();

    var year = date.getFullYear();

    /\* 在日期格式中，月份是从0开始的，因此要加0

     \* 使用三元表达式在小于10的前面加0，以达到格式统一  如 09:11:05

     \* \*/

    var month = date.getMonth() + 1 < 10 ? '0' + (date.getMonth() + 1) : date.getMonth() + 1;

    var day = date.getDate() < 10 ? '0' + date.getDate() : date.getDate();

    // 拼接

    return year + '-' + month + '-' + day;

}

/\*\*

 \* @description: 获取当前日期时间函数

 \* @return {string} 返回格式为"2021-01-11 18:06:53"

 \*/

function GetDataTimeFunc() {

    var date = new Date();

    var year = date.getFullYear();

    /\* 在日期格式中，月份是从0开始的，因此要加0

     \* 使用三元表达式在小于10的前面加0，以达到格式统一  如 09:11:05

     \* \*/

    var month = date.getMonth() + 1 < 10 ? '0' + (date.getMonth() + 1) : date.getMonth() + 1;

    var day = date.getDate() < 10 ? '0' + date.getDate() : date.getDate();

    var hours = date.getHours() < 10 ? '0' + date.getHours() : date.getHours();

    var minutes = date.getMinutes() < 10 ? '0' + date.getMinutes() : date.getMinutes();

    var seconds = date.getSeconds() < 10 ? '0' + date.getSeconds() : date.getSeconds();

    // 拼接

    return year + '-' + month + '-' + day + ' ' + hours + ':' + minutes + ':' + seconds;

}

/\*\*

 \* @description: 写入日志信息(同步函数)

 \* @param {\*}

 \* @return {\*}

 \*/

function logWrite(dirname, text) {

    const fs = require('fs');

    const path = require('path');

    if (!fs.existsSync(dirname)) {

        if (mkDirsSync(path.dirname(dirname))) {

            fs.mkdirSync(dirname);

        }

    }

    fs.writeFileSync(path, GetDataTimeFunc() + '  操作人：' + $System.userName + '\r\n', { flag: 'a', encoding: 'utf-8', mode: '0666' });

    fs.writeFileSync(path, text, { flag: 'a', encoding: 'utf-8', mode: '0666' });

    fs.writeFileSync(path, '\r\n\r\n', { flag: 'a', encoding: 'utf-8', mode: '0666' });

}

/\*\*

 \* @description: 获取各种表单号、任务编号 规则：类型 + 日期 + 流水号

 \* @param {string} type - 编号类型 1:工单号 2：原材料质检 3：成品出库单 4：半成品质检 5：出库单（移动类型包含成本中心领用，委外...） 6：成品质检 7: MRB 8：退库单 9：成品入库单 10：领料单 11：成本中心退料单 12：物权转移单

 \* @return {string} id - 单号

 \*/

async function getID(type) {

    //单号前缀配置数组

    var TaskIdConfig = ['6#', 'YLZJ', 'CPCK', 'BCPZJ', 'CKD', 'CPZJ', 'MRB', 'TKD', 'CPRKD', 'LLD', 'CBZXTL'];

    var dataBaseName = '[dbo].[serial\_number]';

    try {

        //获取年月日

        var date = new Date();

        var year = date.getFullYear();

        var month = date.getMonth() + 1 < 10 ? '0' + (date.getMonth() + 1) : date.getMonth() + 1;

        var day = date.getDate() < 10 ? '0' + date.getDate() : date.getDate();

        var field = 'num' + type; //该类型单号对应的字段名

        var num = 1; //流水号

        var startTime = year + '-' + month + '-01';

        var endTime = '';

        if (parseInt(month) + 1 > 12) {

            endTime = year + 1 + '-01-01';

        } else {

            endTime = year + '-' + (parseInt(month) + 1 < 10 ? '0' + (parseInt(month) + 1) : parseInt(month) + 1) + '-01';

        }

        var sqlStr = `SELECT ${field} FROM ${dataBaseName} WHERE DT < '${endTime}' AND DT >= '${startTime}'`;

        var resData = await toDataSet('BTR', sqlStr);

        if (resData.length == 0) {

            await toDataSet('BTR', `INSERT INTO ${dataBaseName} (DT) VALUES ('${GetDataFunc()}')`);

            num = 1;

        } else {

            resData.forEach(function (item) {

                item[field] > num ? (num = item[field]) : 1;

            });

        }

        //将当日流水号加1

        var data = await toDataSet('BTR', `UPDATE ${dataBaseName} SET num${type} = '${num + 1}' WHERE DT < '${endTime}' AND DT >= '${startTime}'`);

        num < 10 ? (num = '0' + num) : 1;

        return TaskIdConfig[type - 1] + year + month + num;

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @description:

 \* @param {\*} type

 \* @param {\*} msg

 \* @return {\*}

 \*/

function tip(type, msg) {

    var msg1, type1;

    var duration1 = 1500;

    switch (msg) {

        case undefined:

            msg1 = '提交成功';

            if (type === undefined) {

                type = 'success';

            }

            break;

        default:

            msg1 = msg;

    }

    switch (type) {

        case undefined:

            type1 = 'info';

            break;

        case 'success':

            type1 = type;

            break;

        case 'error':

            type1 = type;

            duration1 = 4000;

            break;

        case 'warning':

            type1 = type;

            duration1 = 2500;

            break;

        default:

            type1 = 'info';

    }

    var options = {

        type: type1,

        message: msg1,

        offset: 20,

        duration: duration1,

        showClose: true,

    };

    ShowMessage(options);

    return;

}

/\*\*

 \* @description: 修改对象指定的键名

 \* 函数本身是一个深拷贝，通过对其每层中对象的“键”做匹配替换即实现了多层的“键”替换，另外这里如果传空数组此函数就是一个深拷贝。

 \* @param {object[object]} data - 需要修改键名的数组对象

 \* @param {object[]} keyMap - {旧键名1：'新键名1', 旧键名2：'新键名2', 旧键名3：'新键名3'}

 \* @return {object[object]}

 \*/

function copyTrans(data, keyMap) {

    try {

        for (var i = 0; i < data.length; i++) {

            var obj = data[i];

            for (var key in obj) {

                var newKey = keyMap[key];

                if (newKey) {

                    obj[newKey] = obj[key];

                    delete obj[key];

                }

            }

        }

        return data;

    } catch (e) {

        console.log(e);

        return false;

    }

}

/\*\*

 \* @description: 忽略大小写判断字符串str是否包含subStr

 \* @param subStr 子字符串

 \* @param str 父字符串

 \* @returns boolean

 \*/

function coverString(subStr, str) {

    var reg = eval('/' + subStr + '/ig');

    return reg.test(str);

}

/\*\*

 \* @description: 生成随机GUID码

 \* @return {string} 返回GUID码

 \*/

function Guid() {

    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c) {

        var r = (Math.random() \* 16) | 0,

            v = c == 'x' ? r : (r & 0x3) | 0x8;

        return v.toString(16);

    });

}

/\*\*

 \* @description: KP执行SQL语句

 \* @param {string} dataSource - 数据源名称

 \* @param {string} sqlStr - SQL语句

 \* @return {\*} 若为查询则返回数据集（数组对象），若为增删查则返回true/false，若查询失败则返回false且弹窗提示，控制台打印错误信息

 \*/

function KPtoDataSet(dataSource, sqlStr) {

    try {

        //select校验

        var reg = eval('/' + 'select' + '/ig');

        if (reg.test(sqlStr)) {

            //查询语句

            var res = SyncSQLExecute(dataSource, 0, sqlStr, res);

            if (res.errorCode != 0) throw 'SQL语句执行失败： ' + sqlStr;

            var data = res.data.records;

            if (data.length == 0) {

                $Function.tip('warning', '查询无数据！ SQL语句为：' + sqlStr);

            }

            return data;

        } else {

            //增删改语句

            var res = SyncSQLExecute(dataSource, 1, sqlStr, res);

            if (res.errorCode != 0) throw 'SQL语句执行失败： ' + sqlStr;

            return true;

        }

    } catch (e) {

        console.log(e);

        $Function.tip('error', e);

        return false;

    }

}

/\*\*

 \* @description: 执行数据库操作函数(异步)

 \* @param {object[]} dataSource - 数据源名称

 \* @param {string} sqlStr - SQL语句

 \* @return {object} 若为查询则返回数据集（数组对象），若为增删查则返回true/false，若查询失败则返回false且弹窗提示，控制台打印错误信息

 \*/

async function toDataSet(dataSource, sqlStr) {

    //请求调用mssql

    var sql = require('mssql');

    //数据库连接配置信息

    var config = ['10.102.165.122', 'sa', 'Sa123', 'BTR'];

    var DBconfig = {

        server: config[0],

        authentication: {

            type: 'default',

            options: {

                userName: config[1],

                password: config[2],

            },

        },

        options: {

            database: config[3],

            encrypted: false,

            trustedConnection: true,

            encrypt: false,

            enableArithAbort: true,

            trustServerCertificate: true,

        },

        pool: {

            min: 0,

            max: 10,

            idleTimeoutMillis: 3000,

        },

    };

    //申请连接池

    var conn = new sql.ConnectionPool(DBconfig);

    var req = new sql.Request(conn);

    await conn.connect();

    return new Promise(function (resolve, reject) {

        req.query(sqlStr, function (err, recordreset) {

            if (err) {

                console.log(err);

                resolve(false);

            } else {

                resolve(recordreset.recordset === undefined ? true : JSON.parse(JSON.stringify(recordreset.recordset)));

            }

        });

    });

}

/\*\*

 \* @description: 判断是否为正确的JSON格式字符串

 \* @param {string} str - JSON字符串

 \* @return {\*}

 \*/

function isJSON(str) {

    if (typeof str == 'string') {

        try {

            var obj = JSON.parse(str);

            if (typeof obj == 'object' && obj) {

                return true;

            } else {

                return false;

            }

        } catch (e) {

            return false;

        }

    }

}

/\*\*

 \* @description: 将字典转换成数据集

 \* @param {object[]} map - 需要转换的字典

 \* @return {object[object]} 返回数据集

 \*/

function toArray(map) {

    var dataSet = [];

    for (var key in map) {

        dataSet.push(item);

    }

    return dataSet;

}

/\*\*

 \* @description: 导入函数

 \* @param {object} config - 字段配置对象{ excel字段名1：数据网格field字段名1，...}

 \* @param {string} datagridName - 数据网格名字

 \* @param {string} dataBaseName - 数据库名字（若为空则不存数据库）（按数据网格字段名存数据库）

 \* @param {function} callback - 回调函数

 \* @return {\*}

 \*/

function importExcel(config, datagridName, dataBaseName, callback) {

    try {

        ImportByExcel(function (data) {

            if (!data) throw new Error('[importExcel]  导入数据为空！');

            copyTrans(data, config);

            $(eval(datagridName).id).datagrid({

                data: data,

            });

            if (dataBaseName != '') {

                var res = $Function.SqlInsert(data, dataBaseName);

                if (!res) throw new Error('[importExcel]  导入字段与数据库字段不匹配！存入数据库失败！');

                callback(res);

            }

        });

    } catch (e) {

        callback(e);

    }

}

/\*\*

 \* @description: 导入函数

 \* @param {\*}

 \* @return {\*}

 \*/

module.exports = {

    fileDisplay,

    OcxFiltering,

    mkDirsSync,

    KCrequest,

    GetDataFunc,

    GetDataTimeFunc,

    logWrite,

    getID,

    tip,

    copyTrans,

    coverString,

    Guid,

    toDataSet,

    dataFilter,

    SqlInsert,

    JSON\_to\_dataSet,

    toMap,

    dataSet\_to\_JSON,

    sqlDistinct,

    sqlDistinct,

    sqlGroupby,

    sqlInnerjoin,

    sqlLeftjoin,

    sqlOrder,

    isJSON,

    toArray,

};

### QCresultBack（质检结果回传MES、WMS）

#### 描述

文本

描述已自动生成

#### 说明

该函数只回传MES和WMS系统。

### QcresultGenerate（质检结果生成函数）

#### 描述

屏幕上有字

描述已自动生成

#### 说明

根据自动判定标识生成质检结果，在手动输入质检项结果，或接收质检中心传过来的质检项结果后，自动生成质检项结果和质检任务结果。生成逻辑在上述4.2中已详细阐述。

## 触发器说明

### ERP生产订单传入MES表Order\_information

#### 说明

该触发器将OrderAFVCData，OrderHeaderData，OrderRESBData整合成一条数据插入到Order\_information。其中若有多个副产品名称，副产品代码和副产品重量，则用 / 合并成一条数据。

### WMS入库请求

#### 说明

根据WMS插入的入库请求，查询storage\_task表是否有对于批次的已审核的入库指令，若有则查询storage\_batch表将相关信息返回到该行数据中。

### 成品/半成品质检回传ERP

#### 说明

将成品/半成品生成的质检结果按小批次回传至SAP的QC\_determination表中。

### 根据插入的标志位更新自身materialMasterData

#### 说明

根据插入的数据，若UPDKZ字段为C则插入该表，若为U则根据物料号更新该条记录，若为D则根据物料号删除该条记录。

### 将物料信息同步到Materialinformation和manuf\_BOM\_copy1中

#### 说明

根据插入materialMasterData表的数据更新Materialinformation和manuf\_BOM\_copy1。

### 领料单任务状态更新

#### 说明

根据插入的数据，查询该领料单下的所有小批次物料是否都已领料，若是则更新storage\_task的领料单任务状态为已完成。

### 收料校验

#### 说明

现场触摸屏收料后，往batch\_SL表中插入数据，根据插入的小批次，校验SAP工作中心下的收料站点、批次是否在收料批次表get\_realTime中、是否重复收料。校验完成后，回传SAP的生产入库接口Production\_storage\_withdrawal，质检任务生成，插入到QC\_RealTimeTask和QC\_result、插入到线边仓库存中。更新收料批次表get\_realTime。

### 四号用料计划排产重量更新

#### 说明

若四号排产用到六号的物料，会往该表中插入一条数据，根据插入的小批次号，更新线边仓库存storage\_batch或WMS\_batch中的计划排产重量。

### 投料校验

#### 说明

现场触摸屏投料后，往batch\_TL·表中插入数据，根据插入的小批次，校验SAP工作中心下的收料站点、批次是否在收料批次表put\_realTime中、是否重复投料。校验完成后，回传SAP的生产入库接口Batch\_PutTask。更新线边仓库存。更新投料批次表put\_realTime。

### 线边仓调度

#### 说明

PDA扫码后，插入Theline\_task，根据小批次校验，校验无误后，更新线边仓库存表的仓位信息。

### 指令反馈更新其他表标志位

#### 说明

WMS将出入库指令执行的结果插入该表中，触发器根据插入的任务单号，更新storage\_task表的任务状态，同时更新线边仓和立库库存信息storage\_batch和WMS\_batch，同时将调度信息回传至SAP接口Batch\_PutTask。注意调度的时候需要根据质检结果选择不同的SAP仓位。

### 产出冲销

#### 说明

判断如果插入数据的移动类型为102或532，则为产品/副产品冲销，删除对应的批次信息。

### 更新出入库指令状态

## ERP接口工具说明

## WMS接口工具说明