FT PharmaSuite®

CONFIGURATION AND EXTENSION - VOLUME 5

RELEASE 10.01.00

TECHNICAL MANUAL

PUBLICATION PSCEV5-GR007B-EN-E-MARCH-2021
Supersedes publication PSCEV5-GR007A-EN-E

Rockwell Automation

Contents

Figures

Introduction

This documentation contains important information about how to configure and adapt a PharmaSuite system including information about actions that need to be performed in FactoryTalk® ProductionCentre.

TIP

PharmaSuite provides the PS Administration application to administer database objects such as access privileges, lists, applications, users, or user groups, see "Implementation Guide PS Administration" [A1] (page 187).

The information is structured in the following sections:

VOLUME 1: STYLE AND LAYOUT

In this volume you will find information about standard styles and layouts that are defined for the user interface of PharmaSuite. You will also learn how to modify the elements of the user interface to fit your purposes.
It consists of the following chapters:

- Changing the General Appearance of PharmaSuite
- Creating and Adapting Forms
- Using Forms in the Production Execution Client
- Using Forms in the Production Management Client
- Adapting the Workflows of the Production Execution Client
- Adapting the Use Cases of the Production Management Client
- Changing the Layout of Forms

VOLUME 2: SEMANTIC (AFFECTING THE WHOLE SYSTEM)

In this volume you will find information about such system functionalities as field attributes, version management, services, audit trail, etc. You will also learn how you can adapt the mechanisms available in PharmaSuite to match your needs, as well as maintain and manage certain elements of the system, such as units of measures, signatures, or users.
It consists of the following chapters:

- Adapting and Adding Field Attributes
- Creating and Extending GUI Activities

- Creating Java Artifacts to Access Application Tables
- Configuring Flexible State Models
- Adapting Versioning Graphs
- Localizing Date and Time Representation
- Managing Services
- Adapting the Order Explosion Service
- Documenting Order- and Workflow-related Changes
- Documenting Skipped and Failed Automatic Transitions of Equipment Entities
- Documenting Object Lock Removal
- Managing Electronic Signatures and Access Rights
- Modifying and Adding Constraints
- Modifying and Adding Checks
- Managing Audit Trail
- Adding Units of Measure and Global Conversion Definitions
- Changing Number Generation Schemes
- Changing or Adding New Reports
- Changing or Adding New Labels
- Working with the Batch Record API
- Adapting the Export for Archive Process
- Adapting the Purge Process
- Defining the Risk Level for Exceptions
- Supporting External Authentication Systems
- Supporting External Exception Review
- Supporting External Interfaces for Equipment Management
- Defining Units of Measure for Potency
- Adapting the About Dialog
- Adding New Scale Drivers
- Providing Localization to Support Different Locales

VOLUME 3: SEMANTIC (AFFECTING SPECIFIC AREAS)

In this volume you will find information about providing additional information in Recipe and Workflow Designer or Production Execution Client, etc. You will also learn how you can adapt windows in Recipe and Workflow Designer, Cockpit actions, or user documentation.
It consists of the following chapters:

- Adding Numeric Fields to the Database
- Configuring the Universe of Recipe and Workflow Designer
- Configuring the Setlist of Recipe and Workflow Designer
- Configuring the Property Windows of Recipe and Workflow Designer
- Configuring the Parameter Panel of Recipe and Workflow Designer
- Managing the Layout of Recipe and Workflow Designer and Data Manager
- Configuring the Expression Editor of Recipe and Workflow Designer and Data Manager
- Using ERP BOMs in Recipe and Workflow Designer
- Configuring the Initialization of Material Parameters in Recipe and Workflow Designer
- Extending Comparison in Recipe and Workflow Designer
- Configuring Menu and Toolbar of Recipe and Workflow Designer
- Providing Lists of Pre-defined Texts to Extend Exceptions, Exception Comments, and Signature Comments
- Configuring the Details Window of Data Manager - Work Center
- Configuring the Change History of Data Manager - Work Center
- Configuring FSMs for Equipment Properties
- Providing Images for Equipment Classes in Data Manager
- Providing Report Designs for (Template) Equipment Entities in Data Manager
- Adding an Equipment Logbook Category Accessible by Phase Building Blocks
- Retrieving Specific Information from the Equipment Logbook
- Configuring Additional Purposes for the Equipment Type Technical Property Type
- Adding a Workflow Type to Workflow Designer and the Production Execution Client
- Managing Cockpit Actions of the Production Execution Client
- Configuring Menu and Toolbar of the Production Response Client

- Adding Filter Attributes to the Exception Dashboard of the Production Response Client
- Material Handling for DCS
- Adapting User Documentation

VOLUME 4: CONFIGURATION FRAMEWORK

In this volume you will find information about the methods for managing application configurations, logging and debugging. You will also learn how to administer configuration keys, and which configuration keys are available in PharmaSuite.
It consists of the following chapters:

- Managing Configurations
- Logging and Debugging
- Configuration Keys of PharmaSuite

VOLUME 5: EXTENSION USE CASES

In this volume you will find step-by-step instructions for selected extension use cases. As a trained FactoryTalk ProductionCentre and PharmaSuite system integrator, you will learn how to adapt PharmaSuite.
It consists of the following chapters:

- Processing Materials with a Second Potency (page 13)
- Adding the Second Potency Attribute to ERP BOMs (page 61)
- Adding the Second Potency Function to the Expression Editor (page 71)
- Managing Batches in the ConditionallyReleased Status (page 75)
- Adapting the Approval Workflow of Master Recipes (page 83)
- Adding the Material Balance Section to the Batch Report (page 107)
- Displaying a Retest Date Column in the Overview Panel of the Exception Dashboard (page 141)
- Adding the Print FDA Report Function to PharmaSuite Clients (page 155)
- Adding the Print QA Report Function to Recipe and Workflow Designer (page 167)
- Automatic Archiving and Purging of Specific Orders and Workflows (page 171)
- Displaying Custom Columns in PharmaSuite's Audit Trail (page 179)
- Using a Locale with Unicode Fonts (page 183)

Reference Documents

Finally, the Reference Documents (page 187) section provides a list of all the documentation that is referenced in this manual.
In Volume 5, each extension use case description comes with its own Reference Documents sections.

Intended Audience

This manual is intended for system engineers who implement customer-specific configurations and extensions to a PharmaSuite standard system.

The system engineers need to have a thorough working knowledge of FactoryTalk ProductionCentre, PharmaSuite, and other components relevant to the configuration or extension use cases.
Some use case require a fully set up PharmaSuite development environment and profound Java know-how.

Typographical Conventions

This documentation uses typographical conventions to enhance the readability of the information it presents. The following kinds of formatting indicate specific information:

Bold typeface	Designates user interface texts, such as ■ window and dialog titles ■ menu functions ■ panel, tab, and button names ■ box labels ■ object properties and their values (e.g. status).
Italic typeface	Designates technical background information, such as ■ path, folder, and file names ■ methods ■ classes.
CAPITALS	Designate keyboard-related information, such as ■ key names ■ keyboard shortcuts.
`Monospaced typeface`	Designates code examples.

Chapter 2

Extension and Naming Conventions

This section describes how to control efforts for migrating your PharmaSuite installation to upcoming versions. If you cannot observe the guidelines for technical reasons, please report the issue to your dedicated delivery team of Rockwell Automation or your system integrator.

PharmaSuite artifacts fall into two main groups: building blocks and PharmaSuite core artifacts.

- When you wish to modify a **building block**, use a copy of the building block. For details, see "Technical Manual Developing System Building Blocks" [A2] (page 187).
- When you wish to modify a **PharmaSuite core artifact**, the extension strategy depends on the artifact itself.
  Modify **DSX objects** and copy **all other PharmaSuite artifacts** (e.g. XML configurations for services).

In all cases, please follow the guidelines to control the migration effort (page 7).

Guidelines to Control the Migration Effort

Please ensure that you observe the guidelines listed below.

1. Retain the published API
   The published API of PharmaSuite is accessible via the PharmaSuite start page ("PharmaSuite-related Java Documentation" [C1] (page 187)).
   - If you adapt PharmaSuite on Java level, you must only use the published PharmaSuite Java API.
     Published interfaces will only be changed if necessary. Changes to these interfaces and classes will be announced in future release notes.
   - Avoid using methods and classes that are not published, classes from the implementation package (*...impl...*), or Pnuts functions from any PharmaSuite subroutine.
     These classes and functions may change in future versions of PharmaSuite without notification.

2. Rather modify than copy
   Whenever you wish to extend/modify any object of PharmaSuite in Process Designer (DSX objects), just override the object, except for the following objects:

   - For the **Application** object **Default Configuration**, apply the mechanism of nested configurations in order to reduce a potential migration effort.
     PharmaSuite provides the PS Administration application to administer database objects such as access privileges, lists, applications, users, or user groups, see "Implementation Guide PS Administration" [A1] (page 187).
     Using Process Designer for these tasks is possible, but does not provide the full functional scope required for PharmaSuite. For information on the administration of database objects with Process Designer in this case, please refer to chapter "Managing Configurations" in Volume 4 of the "Technical Manual Configuration and Extension".
   - For **FSMs** (flexible state models), structural changes (i.e. changes to states and transitions) are not allowed in order to enable a later system migration. Modifications of semantic properties can be applied to the standard FSM. They do not impact a system migration.

   When you migrate your PharmaSuite installation to another version, PharmaSuite Update and Migration displays a warning if a standard object was changed. Subsequently, you can decide whether to adapt your extension, ignore the changes delivered with the new version, or replace your extension with the new version (if applicable).

3. Mark your objects
   When naming your artifacts (e.g. objects in Process Designer, classes, interfaces, methods, functions, building blocks), use specific prefixes for your objects. The main purpose of the naming conventions is to prevent naming conflicts with deliverables from other vendors or with other versions.

   - Define and make use of a vendor code consisting of up to three uppercase letters as prefix (e.g. MYC for the My Company vendor code).
     The **X_** and **RS_** prefixes are reserved for PharmaSuite and PharmaSuite-specific product building blocks, respectively.
   - Do not reuse any of the prefixes of PharmaSuite objects in Process Designer in order to avoid conflicts during migration.
   - This guideline also applies to UDA definitions and column names of application tables.
   - Additional conventions apply to building blocks (page 9).

If you do not observe the guidelines, an update process during system migration may fail due to conflicts.

Building Block-specific Conventions

Besides the general conventions (page 7), additional conventions apply to building blocks related to vendor code, version number, and length restrictions:

1. Vendor code
   - It must be appended to the name of a phase or parameter class used in the UI (enclosed in round brackets).
   - It must be used as prefix for the AT definitions.
   - The package name must also contain a vendor reference. You can either use the vendor code or write out the company's full name.
2. Version number
   - The version number consists of two components, an integral part to refer to a major version and a fractional part to refer to a minor version, e.g. 2.1.
   - It must be appended to the name of the phase or parameter class used in the UI and to the base name used for the generated artifacts.
   - For the UI, the version number is enclosed in square brackets, e.g. [2.1].
   - Internal names must not contain brackets and dots, since Java does not allow the usage of these characters. Therefore, the last four characters are reserved for the version number, with digits 1 and 2 representing the major version and digits 3 and 4 representing the minor version. The format is xxyy, e.g. 0201 for version [2.1], 0100 for version [1.0], or 0113 for version [1.13].
3. Length restrictions
   - The maximum length of the **name** of a phase or parameter class used in the UI is 64 characters.
   - The maximum length of the **base name** of a phase building block or parameter class is 18 characters (14 for the name, 4 for the version).

Examples:

- **Hello World** phase of **My Company** with vendor code **MYC** in version **2.1**

```
<Name>Hello World Phase (MYC) [2.1]</Name>
<PhaseLibBaseName>HelloWorld0201</PhaseLibBaseName>
<ATDefinitionPrefix>MYC</ATDefinitionPrefix>
<PackageName>com.mycompany.phase.helloworld</PackageName>
```

- **My Parameter** parameter class of **My Company** with vendor code **MYC** in version **1.0**

```
<Name>My Parameter (MYC) [1.0]</Name>
<ParamClassBaseName>MyParam0100</ParamClassBaseName>
<ATDefinitionPrefix>MYC</ATDefinitionPrefix>
<PackageName>com.mycompany.parameter.myparam</PackageName>
```

Oracle Database Data Types: varchar2 and nvarchar2

When you define text fields for FactoryTalk ProductionCentre application tables or UDAs, you should be aware that there are significant differences between the **varchar2** and **nvarchar2** data types used by Oracle databases.

- The maximum field length for both database data types is restricted to 4000 bytes.
- A field of the **nvarchar2** data type works as expected: For a 4000 characters text field, you can only insert the maximum of 2000 2-byte UTF8 characters or 1300 3-byte UTF8 characters. PharmaSuite text input fields check the number of bytes.
- For an Oracle 11 database, a field of the **varchar2** data type can only handle 1000 2-byte UTF8 characters or 667 3-byte UTF8 characters. In this case, PharmaSuite text input fields check the byte length and prevent the database exception "ORA-01704: string literal too long".

TIP

Previous versions of FactoryTalk ProductionCentre (prior to 9.3) and PharmaSuite (prior to 5.0) contained **varchar2** database data type definitions instead of **nvarchar2**. Therefore a migrated PharmaSuite system may contain **varchar2** definitions and the maximum length of a **varchar2** database field may be defined in maximum number of **bytes**. In this case, if a text does not only contain 1-byte UTF8 characters, the **byte** length is greater than the **char** length and can cause a "value too large exception" (ORA-01401: inserted value too large for column, ORA-12899: value too large for column).
For this reason, we recommend to migrate all **varchar2** fields to **nvarchar2** fields. A migrated FactoryTalk ProductionCentre database should have the same database schema as the database of a newly installed FactoryTalk ProductionCentre system and only contain **nvarchar2** database data type field definitions.

What Are Extension Use Cases

PharmaSuite comes with an extensive range of functions. However, it cannot cater to all functional requirements that result from customers' specific business needs. For this reason, a system integrator can configure and extend PharmaSuite to make additional features available.
This manual describes common extension use cases that we expect to be implemented in customer solutions of PharmaSuite.

IMPORTANT

Please be aware that the provided code examples are only suggestions for the implementation of an extension use case. They are not necessarily executable.

Each description of an extension use case consists of a step-by-step-instruction appropriate for trained FactoryTalk ProductionCentre and PharmaSuite system integrators. The subsequent Reference Documents section provides information where to find more details that are relevant for implementing the extension use case.

Please keep in mind that the changes related to an extension use case may be GxP-relevant. This is independent of changing one, more than one, or even all of the PharmaSuite applications.

Chapter 4

Processing Materials with a Second Potency

IMPORTANT

This extension use case is not compatible with the current version of PharmaSuite and the **D Identify Material** and **D Weigh** phases that are compatible with the current version of PharmaSuite.

This section describes how to adapt PharmaSuite to process materials with a second potency, which applies to dispensing APIs for vitamin nutraceuticals.

Materials with a second potency are **double-active materials**. For example, Vitamin A is a double-active material since it also contains Vitamin D. Then, Vitamin D is a **dependent-active material**. The planned quantity of Vitamin D needs to be reduced by the already weighed actual quantity of the Vitamin A. In principle, materials with a second potency have to be handled as regular active materials. The only difference is the dependency between both materials (double-active and dependent-active) affecting the sequence of weighing and the reduction of the planned quantity of the dependent-active material. The specific property of such BOM items is handled as a weighing subtype.

The extension use case supports the maintenance of a second potency for materials and batches. Furthermore, the BOM items of the corresponding active materials will be marked as double-active or dependent-active materials in the master recipe.

During execution, the actual second potency of a double-active material (e.g. Vitamin A) will be saved. The planned quantity of the dependent-active (e.g. Vitamin D) will be reduced by the quantity contained in the already identified double-active material.

The description covers the following areas:

- Support of second potency attributes for materials and batches in the Production Management Client and for material parameters in Recipe and Workflow Designer.
  - Add the second potency attribute to materials (page 15), material parameters (page 16), and batches (page 19).
  - Add the second potency attributes to further objects and make the attribute also available for execution: ProcessBomItem (page 18) and OrderStepInput (page 19).

  - Adapt Production Management Client (page 19), Recipe and Workflow Designer (page 16), and Production Execution Client (page 19) to allow input and change of the second potency values.
  - Create a choice list for the weighing subtype (page 23).
  - Add the weighing subtype attribute to material parameters (page 23), ProcessBomItem (page 24), and OrderStepInput (page 19).
- Usage of second potency during execution with the Production Execution Client:
  - Optional
    Only if additional weighing types have been introduced, adapt the related service to determine the target unit of measure used for calculations (page 28).
  - Save the second potency value (page 25) of the identified batch as runtime data (OrderStepInput).
    Thus, the actual quantity of a dependent-active material contained in the double-active material is made calculable..
  - Reduce the planned quantity (page 25) of a dependent-active BOM item when a sublot will be identified for this BOM item for the first time.
    The quantity to be subtracted is calculated from the actual quantities of the corresponding double-active material.
  - Add a check (page 31) to prevent the identification of double-active material sublots in case the planned quantity of the dependent-active material would be exceeded.
  - Add a check (page 31) to force the weighing of a double-active material before any dependent-active materials.
  - Prevent the replacement (page 35) of a double-active material sublot if there are already weighed dependent-active material sublots.
  - When replacing a dependent-active material sublot, calculate the correct remaining quantity (page 35).
  - Optional
    Modify the UI (page 38) of the **Identify material** phase to display the second potency of a double-active material.
  - Optional
    Modify the batch report (page 38) to display the second potency of a double-active material.
  - Optional
    Modify the master recipe report to display the second potency and related data in the **Bill of Materials** section (page 39)

- Optional
  Modify the batch report to display the second potency and related data in the **Order-related BOM Items** section (page 44)

Adding the Second Potency Field Attribute to the Material Object

To add the second potency field attribute to the **Material** (part) object, proceed as follows:

Step 1

Add a new UDA for the second potency to the **Material** object

- For details, see [A1] (page 56)
- The UDA must be of the **MeasuredValue** type.
- Example UDA: MYC_plannedPotency2

Step 2

Configure the new UDA's appearance in the property pane of the Production Management Client

- For details, see [A2] (page 56)

1. Add a Data Dictionary element for the new UDA to the **Part [Default]** Data Dictionary class
   - Example Data Dictionary element: UDA_MYC_plannedPotency2
   - Select the **catWeighing** category to assign the property to the weighing properties.
   - Select the **Visible in Setlist** option.
2. Add messages for the labels of the new UDA to the **DataDictionary_Default_Part** message pack
   - Example message ID: UDA_MYC_plannedPotency2_Label
   - Example message ID (description pane): UDA_MYC_plannedPotency2_Hint

Adding the Second Potency Field Attribute to the Material Parameters

To add the second potency field attribute as an attribute to the material parameters in Recipe and Workflow Designer, proceed as follows:

Step 1

Add an attribute to the material parameters

1. Extend the **X_MaterialParameter** application table
   - For details, see [A1] (page 56)
   - The column must be of the **MeasuredValue** type.
   - Example column: MYC_plannedPotency2

   TIP
   Make sure that name and type of the new column are identical to the name and type of the new UDA of the **Material** object. Then, the UDA's value is automatically copied to the material parameters.

2. Optional
   Initialize material parameters
   - For details, see [A3] (page 56)
   - The *MyCustomerMaterialParameterCustomAttributesHandler* class sets the second potency to 0% if no value was defined in the material master data in the Production Management Client.

```
public class MyCustomerMaterialParameterCustomAttributesHandler
              extends MESMaterialParameterCustomAttributesHandler {
  @Override
  public void copyCustomAttributes(Part material, IMESMaterialParameter parameter) {
    if (parameter.getType() == TYPE.INPUT) {

      if (material.getUDA("MYC_potency2") == null) {
        MeasuredValue defaultValue = MeasuredValueUtilities.createMV("0 %");
        parameter.getATRow().setValue("MYC_plannedPotency2", defaultValue);
      } else {
        parameter.getATRow().setValue("MYC_plannedPotency2",
                              material.getUDA("MYC_potency2"));
      }
    }
    // copy further custom UDAs if needed
  }
}
```

Step 2

Configure the new attribute's appearance in the (material) Parameter Panel and the Setlist of Recipe and Workflow Designer

- For details, see [A4] (page 56)

1. Add a Data Dictionary element for the new attribute to the *MESMaterialParameterCustomerConfig* Data Dictionary class
   - Example Data Dictionary element: MYC_plannedPotency2
   - Select the **Visible in Setlist** option (applies to Setlist and Parameter Panel).
2. Add a message for the label of the new attribute to the **DataDictionary_Default_MESMaterialParameterCustomerConfig** message pack
   - Example message ID: MYC_plannedPotency2_Label

Adding the Second Potency Field Attribute to the ProcessBomItem Object

To add the second potency field attribute to the **ProcessBomItem** object, proceed as follows:

Step 1

Add a new UDA for the second potency to the **ProcessBomItem** object

- For details, see [A1] (page 56)
- The UDA must be of the **MeasuredValue** type.
- Example UDA: MYC_plannedPotency2

Step 2

Optional
The system copies all columns of the material parameter with the same name as named UDA to the **ProcessBomItem** object. If you wish to replace the system behavior extend the *MESMaterialParameterCustomAttributesHandler* class.

- For details, see [A3] (page 56)
- The *MyCustomerMaterialParameterCustomAttributesHandler* class copies the **MYC_plannedPotency2** column of the material parameters to the **ProcessBomItem** object.

```
    public class MyCustomerMaterialParameterCustomAttributesHandler
              extends MESMaterialParameterCustomAttributesHandler {
@Override
public void copyCustomAttributes(IMESMaterialParameter parameter,
                                 ProcessBomItem bomItem) {
    bomItem.setUDA(parameter.getValue(“MYC_plannedPotency2”), “MYC_plannedPotency2”);
}
```

Adding the Second Potency Field Attribute to the OrderStepInput Object

To add the second potency field attribute to the **OrderStepInput** object, proceed as follows:

Step 1

Add a new UDA for the planned second potency and a new UDA for the actual second potency to the **OrderStepInput** object

- For details, see [A1] (page 56)
- The UDAs must be of the **MeasuredValue** type.
- Example UDAs: MYC_plannedPotency2, MYC_actualPotency2

Adding the Second Potency Field Attribute to the Batch Object

To add the second potency field attribute to the **Batch** object, proceed as follows:

Step 1

Add a new UDA for the second potency to the **Batch** object

- For details, see [A1] (page 56)
- The UDA must be of the same type as the other field attributes which is the **MeasuredValue** type.
- Example UDA: MYC_potency2

Step 2

Configure the new UDA's appearance in the property pane of the Production Management Client

- For details, see [A2] (page 56)

1. Add a Data Dictionary element for the new UDA to the **Batch [Default]** Data Dictionary class
   - Example Data Dictionary element: UDA_MYC_potency2
   - Select the **catWeighing** category to assign the property to the weighing properties.
   - Deselect the **Editable** option since batch attributes will be modified with the **Change Batch Attributes** action.

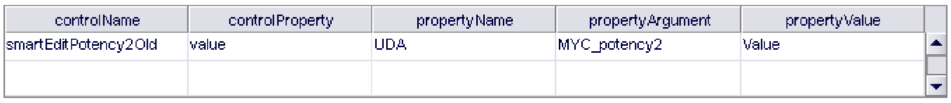
2. Add messages for the labels of the new UDA to the **DataDictionary_Default_Batch** message pack
   - Example message ID: UDA_MYC_potency2_Label
   - Example message ID (description pane): UDA_MYC_potency2_Hint

Step 3

Add the second potency attribute to the **Change Batch Attributes** action of the **Inventory** use case in the Production Management Client

- The **pmc_BatchChangeAttribs** form provides the UI for the **Change Batch Attributes** action.
- For guidance how to find the form, see [A2] (page 56)

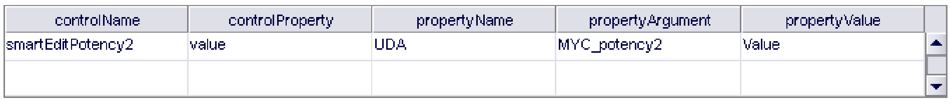
1. Edit the **pmc_BatchChangeAttribs** form.
2. Add an output field for the second potency to the **Current Data** group box
   - Add a label and a **SmartEdit** field.
3. Add an input field for the second potency to the **New Data** group box
   - Add a label and a **SmartEdit** field.
4. Configure the new output field in the **objectBinderEnhancedBatch** object
   - For details, see [A5] (page 56)
   - The value of the **propertyName** must be UDA.
   - The value of the **propertyArgument** must be the UDA name.



controlName	controlProperty	propertyName	propertyArgument	propertyValue
smartEditPotency2Old	value	UDA	MYC_potency2	Value

Figure 1: Configure Current Data of Change Batch Attributes action

5. Configure the new input field in the **objectBinderEnhancedBatchModel** object



controlName	controlProperty	propertyName	propertyArgument	propertyValue
smartEditPotency2	value	UDA	MYC_potency2	Value

Figure 2: Configure New Data of Change Batch Attributes action

   - The value of the **propertyName** must be UDA.
   - The value of the **propertyArgument** must be the UDA name.

Step 4

Optional
Add the second potency attribute to transaction history

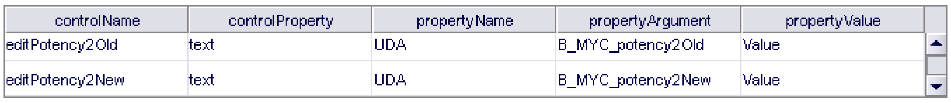
1. Extend the **X_TransactionHistory** application table
   - For details, see [A1] (page 56)
   - The columns must be of the **MeasuredValue** type.
   - Example columns: B_MYC_potency2New, B_MYC_potency2Old

Step 5

Optional (requires previous step)
Add the second potency attribute to the **Transaction History** action of the **Inventory** use case in the Production Management Client

- The **pmc_TransactionHistoryObject** form provides the UI for the **Transaction History** action.
- For guidance how to find the form, see [A2] (page 56)

1. Edit the **pmc_TransactionHistoryObject** form.
2. Add two output fields for the second potency (old and new) to the **Changed Key Data** group box.
   - Add a label and two **SmartEdit** fields.
3. Configure the new fields by means of **ObjectBinderEnhanced** controls.
   - For details, see [A6] (page 56)
   - The value of the **propertyName** must be UDA.
   - The value of the **propertyArgument** must be the column name of the application table.
   - Example columns: B_MYC_potency2New, B_MYC_potency2Old



controlName	controlProperty	propertyName	propertyArgument	propertyValue
editPotency2Old	text	UDA	B_MYC_potency2Old	Value
editPotency2New	text	UDA	B_MYC_potency2New	Value

Figure 3: Configure Changed Key Data of Transaction History action

Step 6

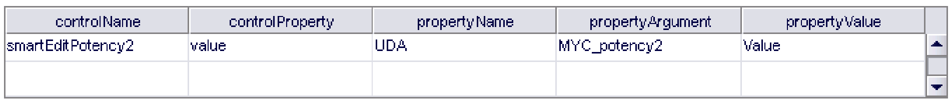
Add the second potency attribute to **Material Receipt** workflow in the Production Execution Client

1. Add an input field for the second potency to the **Create Batch** step
   - The **pec_GoodsReceiptCreateBatch** form provides the UI for the **Create Batch** step.
   - For guidance how to find the form, see [A5] (page 56)
   1. Edit the **pec_GoodsReceiptCreateBatch** form.
   2. Add a label and a **SmartEdit** field.
2. In the script of the form, add the Pnuts code indicated by red font to the *buttonCreateBatchClick()* function. The code creates a map that contains the value of the second potency UDA. Add the map as a parameter to the *createBatch* service method.

```
udaValues = HashMap()
udaValues.put("MYC_potency2", smartEditPotency2.getValue())
thContext = null
orderStep = getFormProperty(propertyGoodsReceiptOrderstep)
if (orderStep != null) {
    thContext = OrderStepExectuionUtils::configureTransactionHistoryContext(thContext,
                orderStep)
}
bb = BatchBuilder(part).batchName(batchName).potency(potency).expiryDate(expiryDate)
    .retestDate(retestDate).udaMap(udaValues).thContext(thContext)
batchResponse = getBatchService().createBatch(bb)

batchResponse = getBatchService().createBatch(batchName, part, potency, expiryDate,
                retestDate, null, udaValues, thContext)
```

3. Bind the new field to the *BatchModel* by means of the **ObjectBinderEnhanced** control. The control is named *objectBinderEnhancedBatch*.
   - For details, see [A5] (page 56)
   - The value of the **propertyName** must be UDA.
   - The value of the **propertyArgument** must be the UDA name.



controlName	controlProperty	propertyName	propertyArgument	propertyValue
smartEditPotency2	value	UDA	MYC_potency2	Value

Figure 4: Configure Second Potency of Create Batch step

4. Optional
   Initialize the input field in the *initForm* method

```
batchModel = BatchModel()
batchModel.setPotency(MeasuredValueUtilities::createMV("0 %"))
zeroPercent = MeasuredValueUtilities::createMV("0 %")
batchModel.setUDA(zeroPercent, "MYC_potency2");
```

Creating the Choice list for the Weighing Subtype

The Weighing subtype controls the specific handling of materials with a second potency like Vitamin A and D. To create the choice list for the weighing subtype, proceed as follows:

Step 1

Create a choice list and its elements for the weighing subtype

- For details, see [A1] (page 56)
- Example choice list: MYC_weighingSubType
- Example choice list elements: (Double Active, 10), (Dependent Active, 20)

Step 2

Add messages for the labels of the new choice list elements to the **MESChoiceListsStrings** message pack.

- Example message IDs: WeighingSubType-Double Active, WeighingSubType-Dependent Active

Adding the Weighing Subtype Field Attribute to the Material Parameters

To add the weighing subtype field attribute as an attribute to the material parameters in Recipe and Workflow Designer, proceed as follows:

Step 1

Add an attribute to the material parameters

1. Extend the **X_MaterialParameter** application table
   - For details, see [A1] (page 56)
   - The column must be of the **Long** type.
   - Example column: MYC_weighingSubType

Step 2

Configure the new attribute's appearance in the (material) Parameter Panel and the Setlist of Recipe and Workflow Designer

- For details, see [A4] and [A1] (page 56)

1. Add a Data Dictionary element for the new attribute to the *MESMaterialParameterCustomerConfig* Data Dictionary class
   - Example Data Dictionary element: MYC_weighingSubType
   - Select the **Visible in Setlist** option (applies to Setlist and Parameter Panel).
   - From the **MES choice list name** drop-down list, select the choice list of the weighing subtype.
   - From the **Editor class name** drop-down list, select the **MES choice editor** option.
2. Add a message for the label of the new attribute to the **DataDictionary_Default_MESMaterialParameterCustomerConfig** message pack
   - Example message ID: MYC_weighingSubType_Label

Adding the Weighing Subtype Field Attribute to the ProcessBomItem Object

To add the weighing subtype field attribute to the **ProcessBomItem** object, proceed as follows:

Step 1

Add a new UDA for the weighing subtype to the **ProcessBomItem** object

- For details, see [A1] (page 56)
- The UDA must be of the **Long** type.
- Example UDA: MYC_weighingSubType

Adding the Weighing Subtype Field Attribute to the OrderStepInput Object

To add the weighing subtype field attribute to the **OrderStepInput** object, proceed as follows:

Step 1

Add a new UDA for the weighing subtype to the **OrderStepInput** object

- For details, see [A1] (page 56)
- The UDA must be of the **Long** type.
- Example UDA: MYC_weighingSubType

Adapting the WDOrderStepInputService Service

The actual second potency of a double-active material is needed to calculate the portion of the second active substance contained in the double-active material (e.g. the portion of Vitamin D contained in Vitamin A). Thus, it is necessary to save the actual second potency value of the identified batch in the **OrderStepInput** object.

When a sublot or batch of a dependent-active material is identified for the first time for a BOM item, the quantity of the dependent-active material contained in the already weighed double-active material has to be calculated. The planned quantity of the dependent-active material to be weighed is reduced by that quantity.

In a scenario like this, several checks are required, e.g. the double-active material position must be completed before the dependent-active material position can be identified and the second active substance portion of the double-active material must not be higher than the planned quantity of the dependent-active material position. For the implementation of the required checks, see "Adapting the Identify Material and Weigh Phases of the Dispense Application" (page 29).

The existing *WDOrderStepInputService* service provides extension hooks to enable the implementation of the required calculation.

To adapt the *WDOrderStepInputService* service, proceed as follows:

Step 1

Create a new class extending the *WDOrderStepInputService* service and override some methods

1. Create a new class extending the *WDOrderStepInputService* service
   - For details, see [A8] (page 56)
   - Example: MYCOrderStepInputService

2. Override the *customizeIdentifySublotFor* method and put the value of the second potency of the identified batch into the field of the actual second potency of the **OrderStepInput** object.
3. Override the *customizeUndoSublotIdentification* method and clear the value of the actual second potency of the **OrderStepInput** object.
4. Override the *getPlannedQuantityForCalculation* method.
   When a position of a dependent-active substance is identified for a BOM item for the first time, the position's planned quantity must be reduced. The *WDOrderStepInputService* service provides the *isFirstOSIForBomItem* API method to detect such a situation.
   In this case, the dependent-active material portion of all actual quantities of all double-active material positions has to be added by means of the actual second potency of the **OrderStepInput** object.
   ```
   MeasuredValue getPlannedQuantityForCalculation(OrderStepInput osi)
   {
     if (DEPENDENT_ACTIVE.equals(weighingSubType(osi))
         && osiService.isFirstOSIForBomItem(osi)) {
       for all valid OSIs of corresponding Double Active {
          sumQuantityDepInDoubleA += actual qty of osi using actual
                                     2nd potency
       return
           osi.getPlannedQuantity.substract(sumQuantityDepInDoubleA)
     } else {
       return  osi.getPlannedQuantity()
     }
   ```

Example code:

```
package com.rockwell.mes.custmes.services;
public class MYCOrderStepInputService extends WDOrderStepInputService {
    @Override
    public MeasuredValue getPlannedQuantityForCalculation(OrderStepInput osi) {
        // ... implement pseudo code shown above
    }
    @Override
    protected void customizeIdentifySublotFor(OrderStepInput orderStepInput,
                      Sublot sublot) {
        try {
            Object pot2 = sublot.getBatch().getUDA("MYC_potency2");
            if (pot2 != null) {
                orderStepInput.setUDA(pot2, CustomizationHelper.ACTUAL_POTENCY_2);
            }
        } catch (DatasweepException e) {
            throw new MESRuntimeException(e);
        }
    }
    @Override
    protected void customizeUndoSublotIdentification(OrderStepInput osi, Sublot sublot)
{
        try {
            osi.setUDA(null, CustomizationHelper.ACTUAL_POTENCY_2);
        } catch (DatasweepException e) {
            throw new MESRuntimeException(e);
        }
```

```
    }
}
```

Step 2

Configure the new service implementation by overriding the default service configuration

- For details, see [A8] (page 56)

1. Create the *extension-config.xml* configuration file and add a reference to the new service configuration file (e.g. *myc-own-services.xml*).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Custom config.xml including extended service definitions -->
<configuration name="extension-config">
  <hierarchicalXml fileName="myc-own-services.xml" />
</configuration>
```

2. Create your service configuration file (e.g. *myc-own-services.xml*) with the implementation class of your service implementation (e.g. *com.rockwell.mes.custmes.services.MYCOrderStepInputService*).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <WDOrderStepInputService>
    <implementationClass>
        com.rockwell.mes.custmes.services.MYCOrderStepInputService
    </implementationClass>
  </WDOrderStepInputService>
</configuration>
```

3. Add your extension configuration files to the classpath by deploying them in a JAR file.

Adapting the WDUOMConversionService Service

For a specific second potency scenario, additional weighing types have been introduced. In order to provide proper quantity calculations for the additional weighing types, it is necessary to adapt how the required unit of measure is determined.

The existing *WDUOMConversionService* service provides an extension hook to enable the implementation of the required functionality. The *getTargetUOMForCalculation* method is called to retrieve the unit of measure needed for quantity calculations of a position of an order step input depending on its weighing type. If the weighing type of the order step input has been added for a project (i.e. differs from compensator, filler, or auxiliary substances), the system calls the *custGetTargetUOMForCalculation* method.

To adapt the *WDUOMConversionService* service, proceed as follows:

Step 1

Create a new class extending the *WDUOMConversionService* service and override some methods

1. Create a new class extending the *WDUOMConversionService* service
   - For details, see [A8] (page 56)
   - Example: MYCWDUOMConversionService
2. Override the *custGetTargetUOMForCalculation* method.

Example code:

```
/**
 * Customization hook, which will be called at getting the unit of measure needed for
quantity
 * calculations of the position of the osi.
 * @param osi the corresponding osi
 * @param weighingType a weighing type
 * @return the unit of measure needed for calculation depending on the weighing type.
 */
protected IUnitOfMeasure custGetTargetUOMForCalculation(OrderStepInput osi,
 * MESChoiceElement weighingType) {
    // may be overwritten for customization purposes
  return null;
}
```

Step 2

Configure the new service implementation by overriding the default service configuration

- For details, see [A8] (page 56)

Adapting the Identify Material and Weigh Phases of the Dispense Application

The following checks must be added to the **Identify material** dispense phase: the double-active material position must be completed before the dependent-active material position can be identified and the dependent-active material portion in the double-active material must not be higher than the planned quantity of the dependent-active material position.

If a sublot will be replaced, it is necessary to modify the calculation of the remaining quantity for dependent-active material positions. The change will affect the **Identify material** phase and the **Weigh** phase.

For this purpose, create new versions of the **Identify material** and **Weigh** phases and adapt the implementation (see [A9] (page 56)).
To add the checks and to modify the calculation in case a sublot will be replaced, it is sufficient to make use of the Phase Copy Tool in order to create a new customer version of the phase.

TIP (JAVA CLASSES)

Java classes of phases end with a version number, e.g. *RtPhaseExecutorMYCWDWeigh0100* for version 1.00. In the following description, *RtPhaseExecutorMYCWDWeigh<version>* is used.

TIP (IMPLEMENTATION)

We recommend to make use of the helper classes available for order step input objects (for details, see [A8] (page 56)).
The *MESAllOSIs* class can be parameterized to filter out non-valid objects and can support the iteration over all valid order step input objects.
The *MESAllSplitOSIs* class, without considering invalid objects, supports the iteration over the order step input objects of the double-active material to calculate the sum of actual quantities of the dependent-active material contained in the double-active material.

For a step-by-step description of the tasks, see:

1. Create new versions of Dispense phases (page 30)
2. Add new checks to the new Identify material phase (page 31)
3. Adapt the calculation related to sublot replacement (page 35)

Creating New Versions of the Dispense Phases

To create new phase versions, proceed as follows:

Step 1

Prepare the creation of a new version of the **Identify material** phase and a new version of the **Weigh** phase by providing the necessary files and directories.

- Perform each sub-step for each phase.

1. Provide the Java package containing the source Java classes of the corresponding phase
   - Unpack the Java source package out of the source JAR file of the corresponding phase building block.
2. Provide the configuration files (XML) of the corresponding phase (*PhaseDataDescription.xml*, *PhaseLibDescription.xml*, and *PhaseOutputDescription.xml*)
   - Unpack the files out of the source JAR file of the corresponding phase building block.
3. Provide the XML files out of the DSX files of the corresponding phase
   - Unpack the *full.dsx* file contained in the corresponding phase installer.
   - Disassemble your DSX file.

Step 2

Create a new version of the **Identify material** phase and a new version of the **Weigh** phase by means of the Phase Copy Tool.

- For details, see [A9] (page 56)

1. Start the Phase Copy Tool and enter the required directory and file information of the source phase.
2. Prepare the configuration parameters required for the new phase and update the following entries:
   - Example phase names: D Identify Material (MYC) [1.0], D Weigh (MYC) [1.0]
   - Example package names: com.mycompany.phase.mycwdmaterialidentification, com.mycompany.phase.mycwdweigh
   - Example phaseLib base names: MYC_WDMatIdent<version>, MYC_WDWeigh<version>

- Example report design names: PS-BatchReport-PhaseWDMatIdentMYC_<version>, PS-BatchReport-PhaseWDWeighMYC_<version>
- Example phase descriptions: MYC: Identify material, MYC: Weigh

3. Update the name mapping for the new Java files.
4. Create the new phases.
5. Copy the generated Java package (in the *java* subdirectory) to the corresponding source directory of the new package for the phase.
6. In Process Designer, copy the standard Report Designs:
   Example:
   PS-BatchReport-PhaseWDMatIdentRS_<version> to PS-BatchReport-PhaseWDMatIdentMYC_<version>
   PS-BatchReport-PhaseWDWeighRS_<version> to PS-BatchReport-PhaseWDWeighMYC_<version>

Adding New Checks to the New Identify Material Phase

The following checks must be added to the **Identify material** Dispense phase: the double-active material position must be completed before the dependent-active material position can be identified and the dependent-active material portion in the double-active material must not be higher than the planned quantity of the dependent-active material position.

To add the checks, proceed as follows:

Step 1

For each check, create a subclass of the *PhaseIdentificationCheck<version>* class containing the check.

- Check: Double-active material position must be completed before a dependent-active material position can be identified

```
public class MYCCheckDependentBeforeDoubleActivePhaseIdentificationCheck<version>
                                   extends PhaseIdentificationCheck<version> {
    public MYCCheckDependentBeforeDoubleActivePhaseIdentificationCheck<version>(
            CheckType argType, String argBeanName,
            MESParamWDCheckConfig<version> validationDef,
            List<Object> argParams, Map<String, Object> argContext) {
        super(argType, argBeanName, "MYCCheckDependentBeforeDoubleActive",
              validationDef, argParams, argContext);
    }
    @Override
    public IIdentificationCheck createIdentificationCheck() {
        MYCCheckDependentBeforeDoubleActive check = new
                                MYCCheckDependentBeforeDoubleActive();
        setIdentificationCheck(check);
        return getIdentificationCheck();
    }
    @Override
    public String getExtendedMessage(Sublot sublot) {
        return "Double active material must be finished before weighing dependent
active
                material.";
    }
    private class MYCCheckDependentBeforeDoubleActive extends
                  AbstractIdentificationCheck {
        @Override
        public void performCheck(IExecuteCheckParameter parameterObject) {
            ExecuteCheckParameter parameters = (ExecuteCheckParameter) parameterObject;
            OrderStepInput osi = parameters.getOrderStepInput();
            IMESChoiceElement subType = CustomizationHelper.getWeighingSubType(osi);
            try {
                if (DEPENDENT_ACTIVE.equals(subType)) {
                    checkDoubleActiveFinished(osi);
                }
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
        private void checkDoubleActiveFinished(OrderStepInput osiD) {
            MESAllOSIs allOSIs = new MESAllOSIs(osiD.getOrderStep(), false);
            boolean notCompletedAFound = false;
            for (OrderStepInput osi : allOSIs) {
                if (DOUBLE_ACTIVE.equals(getWeighingSubType(osi))
                        && (!EnumOrderStepInputStatus.isFinishedStatus(
                              MESNamedUDAOrderStepInput.getStatus(osi)))) {
                    notCompletedAFound = true;
                    break;
                }
            }
            if (notCompletedAFound) {
               getErrorList().add("Double Active Material must be completed before
                                  weighing Dependent Active Material.");
            }
        }
    }
}
```

- Check: Dependent-active material portion in double-active material must not be higher than the planned quantity of the dependent-active material position
  The check performs two calculations:
  - Calculate the portion of the quantity of the dependent-active material contained in the already weighed sublot of the double-active material.
  - Calculate the portion of dependent-active material along with the weighing process of the currently identified double-active material position.

The sum of the quantities of the dependent-active material must be smaller than its planned quantity. Otherwise, the check creates an error.

```
package com.rockwell.mes.phase.product.mycwdmaterialidentification.checks;
public class MYCCheckQtyDependentInDoubleActivePhaseIdentificationCheck<version>
                                     extends PhaseIdentificationCheck<version> {
    public MYCCheckQtyDependentInDoubleActivePhaseIdentificationCheck<version>(
                 CheckType argType, String argBeanName,
                 MESParamWDCheckConfig<version> validationDef, List<Object> argParams,
                 Map<String, Object> argContext) {
        super(argType, argBeanName, "MYCCheckQtyDependentInDoubleActive",
              validationDef, argParams, argContext);
    }
    @Override
    public IIdentificationCheck createIdentificationCheck() {
        MYCCheckQtyDependentInDoubleActive check = new
                            MYCCheckQtyDependentInDoubleActive();
        setIdentificationCheck(check);
        return getIdentificationCheck();
    }
    @Override
    public String getExtendedMessage(Sublot sublot) {
        return "Quantity of dependent active in double active is too high.";
    }
    private class MYCCheckQtyDependentInDoubleActive extends
                  AbstractIdentificationCheck {
        @Override
        public void performCheck(IExecuteCheckParameter parameterObject) {
            ExecuteCheckParameter parameters = (ExecuteCheckParameter) parameterObject;
            MeasuredValue potency2 = getPotency2(parameters.getSublot().getBatch());
            MeasuredValue potency1 = parameters.getSublot().getPotency();
            OrderStepInput osi = parameters.getOrderStepInput();
            IMESChoiceElement subType = getWeighingSubType(osi);
            try {
                if ((NULL_PERCENT.compareTo(potency2) == -1)
                        && DOUBLE_ACTIVE.equals(subType)) {
                    checkQuantityDependentActive(osi, potency2, potency1);
                }
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }

        private void checkQuantityDependentActive (OrderStepInput osi, MeasuredValue
                     potency2, MeasuredValue actualPotency)
                throws InvalidDataException, MESIncompatibleUoMException,
                       CalcNominalQtyCompensatorException,
                       CalcNominalQtyFillerException {
            IWDOrderStepInputService osiService =
                         ServiceFactory.getService(IWDOrderStepInputService.class);
            IWDCalculationService calculationService =
                              ServiceFactory.getService(IWDCalculationService.class);
            OrderStepInput mainOsiDep = getMainOsiDependent(osi.getOrderStep());
            MESAllSplitOSIs splitOSIs = new MESAllSplitOSIs(osi, false);
            MeasuredValue sumDependentInDouble =
                 MeasuredValueUtilities.createZero(
                                mainOsiDep.getPlannedQuantity().getUnitOfMeasure());
            MeasuredValue qtyDependentInDouble;

            for (OrderStepInput aOsiDoubleA : splitOSIs) {
                qtyDependentInDouble = null;
                if (osi.getKey() == aOsiDoubleA.getKey()) {
                    IMeasuredValue[] whatIf =
                     osiService.getCalculatedQuantitiesAndTolerance(osi,actualPotency);
                    IMeasuredValue actualQtyDoubleActive =
                                 calculationService.convertToWeighingUOM(osi.getPart(),
                                                                         whatIf[0]);
                    // quantity of dependent active if current sublot is used
                    // to weigh the remaining quantity of the double active:
                    qtyDependentInDouble =
                         getQtyDependentInDouble(actualQtyDoubleActive, potency2);
                } else if (aOsiDoubleA.getActualQuantity() != null) {
                    qtyDependentInDouble =
                        getQtyDependentInDouble(aOsiDoubleA.getActualQuantity(),
                                                getActualPotency2(aOsiDoubleA));
                }
                if (qtyDependentInDouble != null) {
                    try {
                        sumDependentInDouble = (MeasuredValue)
                               sumDependentInDouble.add(qtyDependentInDouble);
                    } catch (Exception e) {
                        throw new RuntimeException(e);
                    }
                }
            }
            IMeasuredValue reducedPlannedQty =
                   calcCorrectedPlannedQtyOfDependent(mainOsiDep,
                                                   sumDependentInDouble);
            if (reducedPlannedQty.signum() < 0) {
                getErrorList().add(
                   "Quantity of dependent active material in double active material is
too high. Corrected quantity would be "
                   + reducedPlannedQty);
            }
        }
    }
}
```

Step 2

Add the new checks to the check suite. The *PhaseIdentificationCheckSuite<version>* class is responsible for composing the check suite.

```
public class PhaseIdentificationCheckSuite<version> extends
                                       AbstractPhaseIdentificationCheckSuite<version> {
  ...
@Override
    protected void addFixedChecks(final Map<String, Object> context) {
        // OSI related checks from OrderStepInput Service
        for (String beanName : FIXED_IDENTIFICATION_CHECKS) {
            addIdentificationCheck(CheckType.ERROR, beanName, null, null, context);
        }
        // add custom checks
        getIdentChecks().add(
                new MYCCheckDependentInDoubleActivePhaseIdentificationCheck<version>(
                                        CheckType.ERROR, null, null,null, context));
        getIdentChecks().add(
                new
                 MYCCheckDependentBeforeDoubleActivePhaseIdentificationCheck<version>(
                                        CheckType.ERROR, null, null, null, context));
    }
```

Adapting the Calculation Related To Sublot Replacement

If a sublot will be replaced, it is necessary to modify the calculation of the remaining quantity for the dependent-active material positions. The change will affect the **Identify material** phase and the **Weigh** phase.

The replacement of a double-active material sublot without replacing the linked dependent-active material sublot would require a separate treatment. The situation is similar to the relation between active and compensator substances. The Dispense phases prevent the replacement of active substance sublots when compensator sublots are not replaced. Prior to the replacement of an active substance sublot, the compensator sublots must have been replaced before. Therefore, the solution for second potency materials is the same. Double-active material sublots cannot be replaced prior to dependent-active material sublots.

The *SublotReplacementHelper* class of PharmaSuite performs the calculation. To adapt the handling related to sublot replacement, proceed as follows:

Step 1

Create a subclass of the *SublotReplacementHelper* class

- Example: MYCSublotReplacementHelper<version>

1. Override the *isAllowed* method and, in case of double-active material sublots, check that there are no already weighed dependent-active material sublots.
2. Override the *calculateActiveReplacementQuantity* method.
   In case of dependent-active material sublots, the remaining quantity must be calculated separately. If it is the only sublot, use the planned quantity of the recipe. Later on, if a sublot or batch will be identified again for this BOM item (*isFirstOSIForBomItem* returns *true*), then the *getPlannedQuantityForCalculation* method of the *MYCOrderStepInputService* class recalculates all actual quantities of double-active material positions and reduces the planned quantity for the calculation correctly.
   If there are multiple dependent-active material sublots, use the reduced planned quantity of the current BOM item and subtract the actual weight of all sublots. Adding the quantity of the sublot currently being replaced would result in the remaining quantity of the corresponding BOM item.

```
public class MYCSublotReplacementHelper<version> extends SublotReplacementHelper {
  public MYCSublotReplacementHelper<version>(OrderStepInput theOSIToReplace) {
    super(theOSIToReplace);
  }
  /**
   * @return true if it is allowed to replace an active sublot.
   */
  protected boolean isAllowed() {
    return (EnumWeighingTypes.ACTIVE.equals(getCurrentWeighingType()))
      ? (super.isAllowed() && isAllowedActiveDependentActive())
      : super.isAllowed();
  }
```

```
   private boolean isAllowedActiveDependentActive() {
     // return false if there are not replaced sublots of dependent Active
   }
   /**
    * @return remaining quantity
    */
   protected MeasuredValue calculateActiveReplacementQuantity() {
     MeasuredValue remainingQty;
     boolean isDependentActive =
           DEPENDENT_ACTIVE.equals(getWeighingSubType(getOsiToReplace()));
     if (isDependentActive
        && (getQuantities().getNumberActiveSublots(getCurrentPosition()) == 1)) {
       // In case there is exactly one sublot (that will be replaced)
       // the nominal qty will be calculated in WDOrderStepInputService
       // again. Therefore we need to use the initial planned qty of the recipe here.
       remainingQty = getDataAnalyser().getPlannedActive(getCurrentPosition());
     } else {
       try {
         MeasuredValue plannedQtyForCalc =
              getDataAnalyser().getPlannedActive(getCurrentPosition());
         if (isDependentActive) {
           OrderStepInput masterOSI =
              WDOSIServiceHelper<version>.getMasterOSIForOSI(getOsiToReplace());
           // we need the planned quantity reduced by the dependent active
           // contained in the double active
           plannedQtyForCalc =
           WDOSIServiceHelper<version>.getPlannedQuantityForCalculation(masterOSI);
         }
         remainingQty = (MeasuredValue) plannedQtyForCalc.subtract(
           getDataAnalyser().getActualActiveNormalized(getCurrentPosition()));
         // add the normalized qty of the sublot to replace
         remainingQty = (MeasuredValue)
            remainingQty.add(getNormalizedSublotQty());
       } catch (Exception e) {
         throw new MESRuntimeException(
             "Exception during calculation quantity of active substances.");
       }
     }
     return remainingQty;
   }
}
```

Step 2

Create a class that uses the *MYCSublotReplacementHelper<version>* class

- Example: MYCActionSublotReplaceHelper<version>

1. Copy the *createActionPanels4ReplaceSublot* and *replaceTargetSublot* methods of *SublotReplacementHelper<version>* into *MYCActionSublotReplaceHelper<version>*.
2. Replace the creation of *SublotReplacementHelper* with the creation of *MYCSublotReplacementHelper<version>* in the methods of the adapted copy.

```
public class MYCActionSublotReplaceHelper<version> {
...
    public static void
createActionPanels4ReplaceSublot(AbstractWeighActionView<version>
                       actionView, List<Sublot> sublots) {
        ...
            sublotReplacementHelper = new
                                       MYCSublotReplacementHelper<version>(currentOSI);
        ...
    }
    public static void replaceTargetSublot(final Sublot sublot, final OrderStep os,
boolean
                       isCampaign) {
        ...
        SublotReplacementHelper slh = new
                                  MYCSublotReplacementHelper<version>(toBeReplacedOSI);
        ...
    }
}
```

Step 3

Replace the calls to *ActionSublotReplaceHelper<version>.createActionPanels4ReplaceSublot* and *ActionSublotReplaceHelper<version>.replaceTargetSublot* with the corresponding version of the *MYCActionSublotReplaceHelper<version>* class. The **Identify material** and **Weigh** phases are affected (e.g. in *RtPhaseExecutorMYCWDMatIdent<version>*, *RtPhaseExecutorMYCWDWeigh<version>*, *MatIdentActionView<version>*, *WDWeighActionView<version>*).

Step 4

Configure the new sublot replacement implementation to be used by the Production Management Client

- For details, see [A8] (page 56)
- Extend your service configuration file (e.g. *myc-own-services.xml* as in section "Adapting the WDOrderStepInputService Service" (page 25), Step 2, sub-step 2) and configure your implementation class (e.g. *com.rockwell.mes.custmes.services.MYCSublotReplacementHelper*). This ensures

that the replacement of sublots within the Production Management Client also uses your implementation:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <WDOrderStepInputService>
    <implementationClass>
        com.rockwell.mes.custmes.services.MYCOrderStepInputService
    </implementationClass>
  </WDOrderStepInputService>
  <SublotReplacementHelper>
    <implementationClass>
      com.rockwell.mes.custmes.services.MYCSublotReplacementHelper
    </implementationClass>
  </SublotReplacementHelper>
</configuration>
```

Displaying the Second Potency in the UI of the Identify Material Phase

This step is optional.

If you wish to display the second potency in the UI of the phase, extend the *MatIdentView<version>* class.

Displaying the Second Potency in the Phase-specific Report

This step is optional.

If you wish to display the second potency in the phase-specific report, extend the phase data of the phase.

To display the second potency in the phase-specific part of the batch report, proceed as follows:

Step 1

Extend the runtime phase data

- For details, see [A9] (page 56)

1. Update the *PhaseDataDescription.xml* file.
   - Add a field for the second potency of the **MeasuredValue** type
   - Example: actualPotency2
2. Run the phase generator and the parameter class generator, respectively.
3. Copy the generated Java files to the corresponding source directory of the Building Block.

Step 2

Adapt your version of the batch report to match your needs.

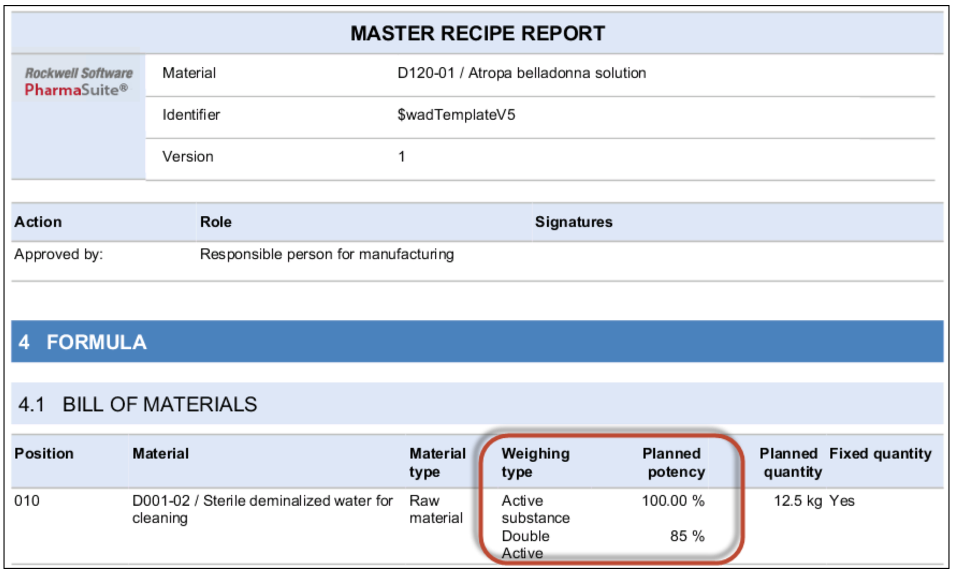
- For details, see [A10] (page 56)

1. In the *fillPhaseData* method of the *RtPhaseExecutorMYCWDMatIdent<version>* class, set the actual value of Potency2 of the batch to the corresponding attribute in the runtime phase data.
2. Optional
   Add a getter for the label of the second potency to the *ReportScriptletMYCWDMatIdent<version>* class.
3. Adapt the report according to [A10] to display the second potency.

Displaying the Second Potency in the Bill of Materials Section of the Master Recipe Report

This step is optional.

In addition to the second potency, the report displays the following related data: weighing type, weighing subtype, and primary potency.



MASTER RECIPE REPORT

Rockwell Software PharmaSuite®	Material	D120-01 / Atropa belladonna solution
	Identifier	$wadTemplateV5
	Version	1

Action	Role	Signatures
Approved by:	Responsible person for manufacturing	

4 FORMULA

4.1 BILL OF MATERIALS

Position	Material	Material type	Weighing type	Planned potency	Planned quantity	Fixed quantity
010	D001-02 / Sterile deminalized water for cleaning	Raw material	Active substance Double Active	100.00 % 85 %	12.5 kg	Yes

Figure 5: Bill of Materials section with weighing (sub)type and potencies

If you wish to display the second potency and related data in the **Bill of Materials** section of the master recipe report, extend the *MasterRecipeConverter* class and extend the **PS-MRReport-BillOfMaterial** sub-report.

- For guidance how to determine the sub-report, see [A11] (page 56).
- For general information about master recipe reports, see [A12] (page 56).

To display the second potency and related data in the master recipe report, proceed as follows:

Step 1

Determine the bean used by the **PS-MRReport-BillOfMaterial** sub-report

- A variable expression defines the report variable to be used by the report file.
  Example:
  $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getParameterAttributeByDataInterpretation($V{currentBean}, "Quantity")
- The *currentBean* variable is defined by a bean class.
  Example:
  ((com.rockwell.mes.services.batchreport.ifc.IMasterRecipeB2MMLJRDataSource) $P{REPORT_DATA_SOURCE}).getCurrentBean()
- The bean class returned here is dynamic depending on context and must be extended by the new attribute for the second potency.
- To inspect the underlying structure of a master recipe record, export the recipe as a BML file with the **PharmaSuite Export/Import Utility** tool. For details, see chapter "Exporting and Importing Master Recipes, Master Workflows, and Custom Building Blocks" in "Technical Manual Administration" [A14] (page 56).

```
...
<Formula>
  <Parameter>
    <ParameterType>Other</ParameterType>
    <ParameterSubType>ProcessBOM</ParameterSubType>
    <Value>
        <ValueString>80</ValueString>
        <ValueString>g</ValueString>
        <ValueString>0</ValueString>
        <DataInterpretation OtherValue="Quantity">Other</DataInterpretation>
        <DataType>Measure</DataType>
        <UnitOfMeasure/>
    </Value>
  ...
```

- The type of the **<Formula>.<Parameter>** element defines which converter method is required.
  To retrieve the type, see the */services.batchrecord/config/BatchML-V0600-BatchInformation.xsd* schema definition file.
  <Formula><Parameter> is of the **BatchParameterType** type.
- A converter method creating the **BatchParameterType** type is needed to process the new attributes.
  To retrieve the method, see *com.rockwell.mes.services.batchrecord.ifc.converter.IMasterRecipeConverter.setBomItemData(BatchParameterType, ProcessBomItem).*

- The method must be extended to transfer the three new attributes (weighing type, weighing subtype, planned potency) from the **ProcessBomItem** object to a **BatchParameterType.Value**.
- The *MasterRecipeConverter* service provides the following convenience methods to add new attribute values for specific data types:
  - *setBomParamValue(BatchParameterType, String, String, String)*
  - *setBomParamMeasuredValue(BatchParameterType, String, MeasuredValue)*

Step 2

Extend the *MasterRecipeConverter* service with your own implementation

- Example service name: MYCMasterRecipeConverter
- Example parameter: MYC_WeighingSubType
- Example parameter: MYC_plannedPotency2
- The names used here are used later by the report. By default, we reuse the UDA names already defined for the **ProcessBomItem**.

```
public class MYCMasterRecipeConverter extends MasterRecipeConverter implements
  IMasterRecipeConverter {
  @Override
  public void setBomItemData(BatchParameterType bitemparm, ProcessBomItem pbomitem) {
    super.setBomItemData(bitemparm, pbomitem);
    // CL: WeighingSubType
    setBomParamValue(bitemparm, "MYC_weighingSubType", getBomItemUDAValue(pbomitem,
      "MYC_weighingSubType"), "string");
    // The 2nd potency
    setBomParamMeasuredValue(bitemparm, "MYC_plannedPotency2",
      getBomItemUDAMeasuredValue(pbomitem, "MYC_plannedPotency2"));
  }
  private MeasuredValue getBomItemUDAMeasuredValue(ProcessBomItem pbomitem,
   String udaname) {
    try {
      return (MeasuredValue) pbomitem.getUDA(udaname);
    } catch (DatasweepException e) {
    }
    return null;
  }
```

Step 3

Configure the new service implementation by overriding the default service configuration

- For details, see [A8] (page 56)

1. Create the *extension-config.xml* configuration file and add a reference to the new service configuration file (e.g. *myc-batchrecord.xml*).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Custom config.xml including extended service definitions -->
<configuration name="extension-config">
   <hierarchicalXml fileName="myc-batchrecord.xml" config-optional="false"  />
</configuration>
```

2. Create your service configuration file (e.g. *myc-batchrecord.xml*) with the implementation class of your service implementation (e.g. *com.rockwell.custmes.services.batchrecord.impl.converter.MYCMasterRecipeConverter*).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
<MasterRecipeConverter>
  <implementationClass>com.rockwell.custmes.services.batchrecord.impl
    .converter.MYCMasterRecipeConverter
  </implementationClass>
</MasterRecipeConverter>
</configuration>
```

3. Add your extension configuration files to the classpath by deploying them in a JAR file.

Step 4

If you display the second potency in the **Order-related BOM Items** section of the batch report (page 44), you can skip this step.

Add messages for the two new column headers to the **PS-BatchReport** message pack

- Example message ID: WeighingType_Label
- Example message: Weighing type
- Example message ID: Planned_Potency_Label
- Example message: Planned potency

Step 5

Modify your version of the master recipe report with Jaspersoft Studio to match your needs

- For details, see [A10] (page 56)

1. In the PharmaSuite report manager, export the **PS-MRReport-BillOfMaterial** sub-report design.
2. In Jaspersoft Studio, add the following variables:
    - Name: bomPotency2
        - Variable Class: java.lang.String
        - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getParameterAttributeByDataInterpretation($V{currentBean}, "MYC_plannedPotency2")
    - Name: bomPotency
        - Variable Class: java.lang.String
        - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getParameterAttributeByDataInterpretation($V{currentBean}, "X_plannedPotency")
    - Name: weighingSubType
        - Variable Class: java.lang.String
        - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getLocalizedChoiceItem("MYC_weighingSubType", Long.valueOf($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getParameterAttributeByDataInterpretation($V{currentBean}, "MYC_weighingSubType")))

    TIP
    To avoid a *NullPointerException*, check that the weighing subtype is not null before you call *Long.valueOf()*.

    - Name. weighingType
        - Variable Class: java.lang.String
        - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getLocalizedChoiceItem("WeighingType", Long.valueOf($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getParameterAttributeByDataInterpretation($V{currentBean}, "X_weighingType")))

3. Place the four fields listed below at an appropriate position of the table layout (e.g. between **Material type** and **Planned quantity**).
   - Add a **Detail 1** field with a **Text Field Expression**:
     "\n" + $V{bomPotency} + "\n"   + \n" + $V{bomPotency2}

> TIP
> Check that the second potency of the BOM item is not null before it is displayed.

   - Add a **Detail 1** field with a **Text Field Expression**:
     $V{weighingType} + "\n" + $V{weighingSubType}

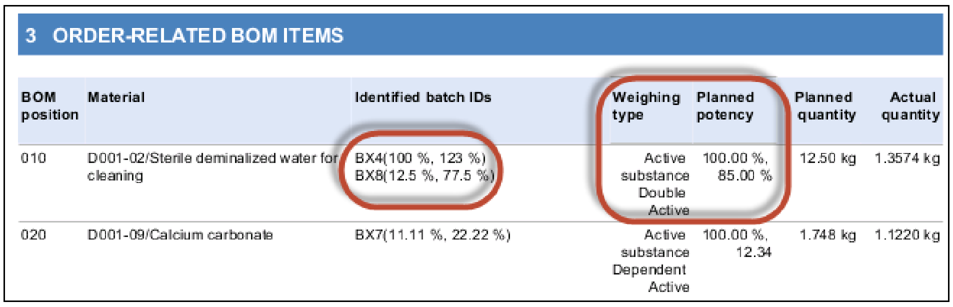
> TIP
> Check that the weighing subtype is not null before it is displayed.

   - Add a column header with a **Text Field Expression**:
     $R{Planned_Potency_Label}
   - Add a column header with a **Text Field Expression**:
     $R{WeighingType_Label}
4. Import the report design by means of the **mes_PS-BatchReportManager** form.

Displaying the Second Potency in the Order-related BOM Items Section of the Batch Report

This step is optional.

In addition to the planned and actual second potency, the report displays the following related data: weighing type, weighing subtype, and planned and actual primary potency. Since the actual potency may vary per identified batch, we display it close to the batch.



3 ORDER-RELATED BOM ITEMS

BOM position	Material	Identified batch IDs	Weighing type	Planned potency	Planned quantity	Actual quantity
010	D001-02/Sterile deminalized water for cleaning	BX4(100 %, 123 %) BX8(12.5 %, 77.5 %)	Active substance Double Active	100.00 %, 85.00 %	12.50 kg	1.3574 kg
020	D001-09/Calcium carbonate	BX7(11.11 %, 22.22 %)	Active substance Dependent Active	100.00 %, 12.34	1.748 kg	1.1220 kg

Figure 6: Order-related BOM Items section with weighing (sub)type and potencies

- For general information about batch records, see [A13] (page 56).
- For general information about batch reports, see [A10] (page 56).

The batch report can only display information from the batch record. In contrast to the BML of the master recipe report, the batch record does not automatically include all UDAs or customer-specific attributes of the underlying PharmaSuite objects.
Thus, new fields in the batch report referencing attributes that are not yet part of the batch record require an extension of the batch record schema.

The mechanism how to get values for fields of the batch report is also different compared to the master recipe report.

- The master recipe report allows to access Java bean attributes in a generic way using **report variables** with expressions like the following one: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getParameterAttributeByDataInterpretation($V{currentBean}, "myAttributeName").
- The batch report uses **report-field-descriptions** as report expression. The report asks for the value of a field, the backend (JRDataSource instantiated by *ObjectFactory*) looks up the report expression (e.g. *orderRelatedBOMItem.PlannedQuantity*) related to the current bean of the section (e.g. *OrderRelatedBOMItemType*) to determine an enumeration item (e.g. *EnumOrderRelatedBOMItemType.PLANNED_QUANTITY*) that is eventually processed to get the corresponding bean property (e.g. *currentBean.getPlannedQuantity()*), and finally returns it as a formatted string back to the report.
  Thus, to deliver values of new fields to the batch report, the report expression, the enumeration, their mapping, and the code to process a field have to be extended by a custom extension of the Java *MESB2MMLJRDataSource* class.

If you wish to display the second potency and related data in the **Order-related BOM Items** section of the batch report, perform the following steps:

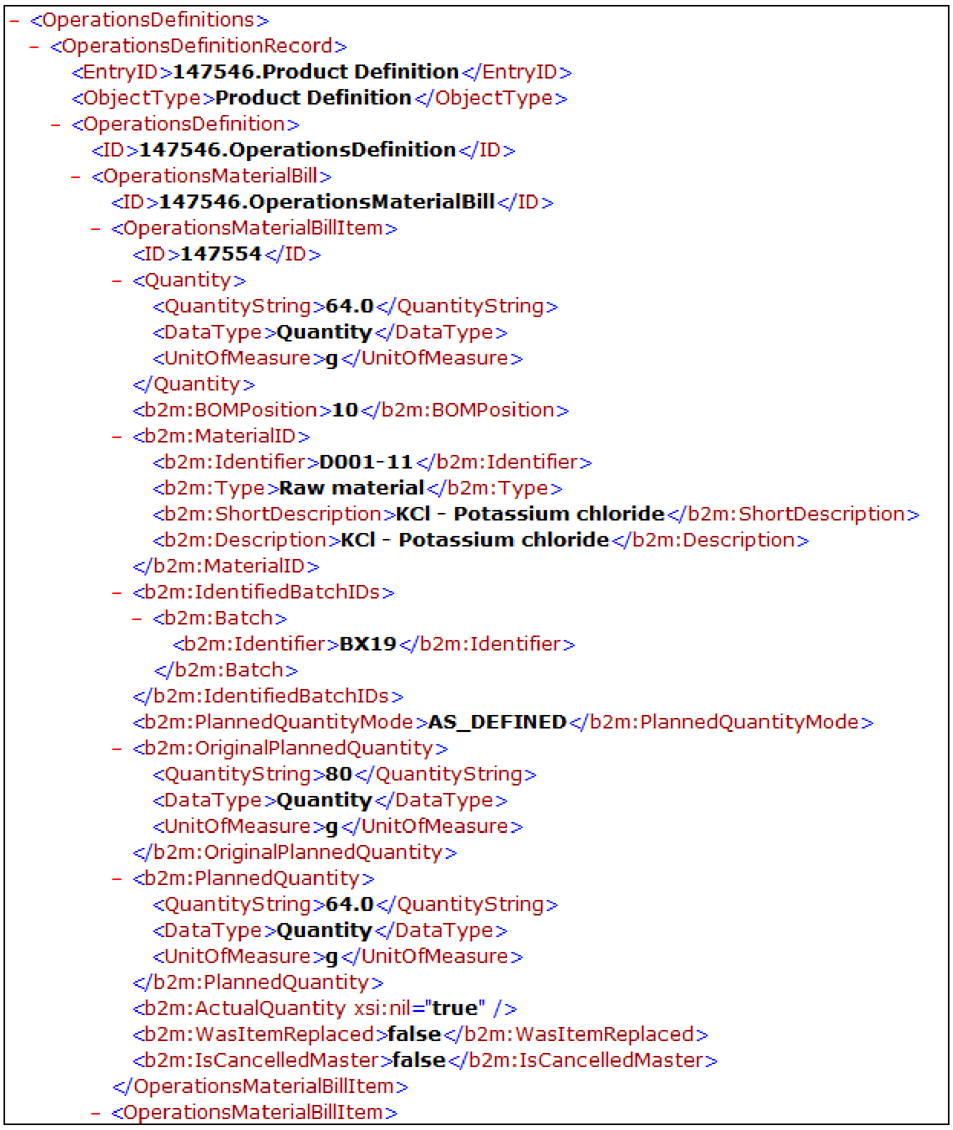
- Extend the batch record schema definition with the new attributes of the **CustomerExtendedOperationsBillOfMaterialItem** and **CustomerExtendedBatchType** types
- Set the values in service classes extending the *OperationsDefinitionsConverter* and *BatchConverter* converters
- Extend the *ReportExpression* class with new field descriptions (in *MESB2MMLJRDataSource*)
- Define a matching enumeration item implementing *IEnumOrderRelatedBOMItemType* (in *MESB2MMLJRDataSource*)
- Add the mapping of the new *ReportExpressions* to the new enumeration items to the *orderRelatedBOMItemTypeMap* in an extension of the *MESB2MMLJRDataSource* class
- Extend the **PS-BatchReport-OrderRelatedBOMItems** sub-report with new column labels and the **PS-BatchReport-OrderRelatedBOMItem** sub-report with new fields.
  For guidance how to determine the sub-report, see [A11] (page 56).

To display the second potency and related data in the master recipe report, proceed as follows:

Step 1

Inspect the BML structure of the batch record

- Run the **demo_PS-ReportRecordTest** form (part of PC_MES_DSX_TESTDATA) to create a batch record based on an executed order. The result displays the record in XML (here: the bean that must be extended for this extension use case).



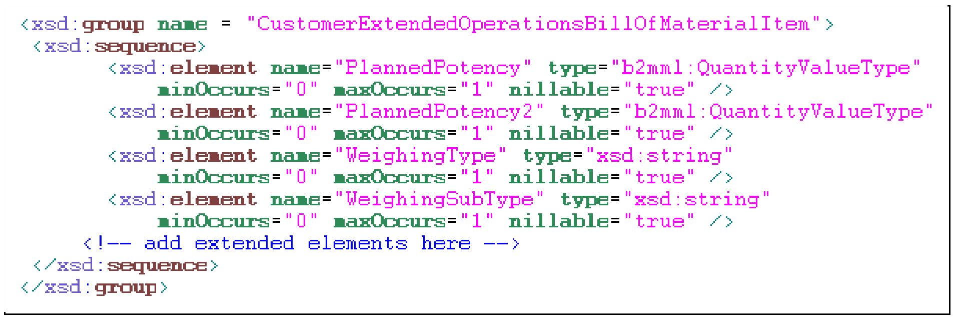
```
- <OperationsDefinitions>
  - <OperationsDefinitionRecord>
      <EntryID>147546.Product Definition</EntryID>
      <ObjectType>Product Definition</ObjectType>
    - <OperationsDefinition>
        <ID>147546.OperationsDefinition</ID>
      - <OperationsMaterialBill>
          <ID>147546.OperationsMaterialBill</ID>
        - <OperationsMaterialBillItem>
            <ID>147554</ID>
          - <Quantity>
              <QuantityString>64.0</QuantityString>
              <DataType>Quantity</DataType>
              <UnitOfMeasure>g</UnitOfMeasure>
            </Quantity>
            <b2m:BOMPosition>10</b2m:BOMPosition>
          - <b2m:MaterialID>
              <b2m:Identifier>D001-11</b2m:Identifier>
              <b2m:Type>Raw material</b2m:Type>
              <b2m:ShortDescription>KCl - Potassium chloride</b2m:ShortDescription>
              <b2m:Description>KCl - Potassium chloride</b2m:Description>
            </b2m:MaterialID>
          - <b2m:IdentifiedBatchIDs>
            - <b2m:Batch>
                <b2m:Identifier>BX19</b2m:Identifier>
              </b2m:Batch>
            </b2m:IdentifiedBatchIDs>
            <b2m:PlannedQuantityMode>AS_DEFINED</b2m:PlannedQuantityMode>
          - <b2m:OriginalPlannedQuantity>
              <QuantityString>80</QuantityString>
              <DataType>Quantity</DataType>
              <UnitOfMeasure>g</UnitOfMeasure>
            </b2m:OriginalPlannedQuantity>
          - <b2m:PlannedQuantity>
              <QuantityString>64.0</QuantityString>
              <DataType>Quantity</DataType>
              <UnitOfMeasure>g</UnitOfMeasure>
            </b2m:PlannedQuantity>
            <b2m:ActualQuantity xsi:nil="true" />
            <b2m:WasItemReplaced>false</b2m:WasItemReplaced>
            <b2m:IsCancelledMaster>false</b2m:IsCancelledMaster>
          </OperationsMaterialBillItem>
        - <OperationsMaterialBillItem>
```

Figure 7: Excerpt of the batch record of an executed order

Step 2

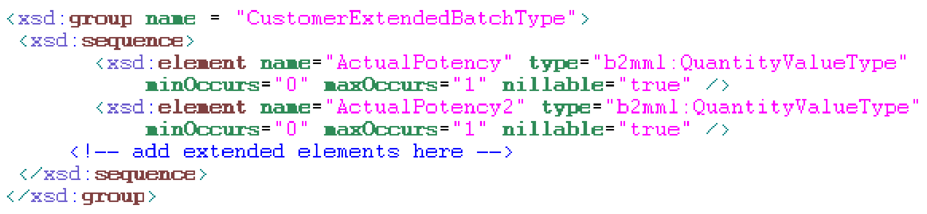
Extend the batch record schema definition with the new attributes of the **CustomerExtendedOperationsBillOfMaterialItem** and **CustomerExtendedBatchType** types

1. To extend types of the batch record schema, modify the *B2MML-V0600-Extensions.xsd* as described in section "Extending the Batch Record Schema" (page 108) in chapter "Adding the Material Balance Section to the Batch Report" (page 107).
   Add new elements to the **CustomerExtendedOperationsBillOfMaterialItem** and **CustomerExtendedBatchType** groups in *B2MML-V0600-Extensions.xsd*



```
<xsd:group name = "CustomerExtendedOperationsBillOfMaterialItem">
 <xsd:sequence>
       <xsd:element name="PlannedPotency" type="b2mml:QuantityValueType"
           minOccurs="0" maxOccurs="1" nillable="true" />
       <xsd:element name="PlannedPotency2" type="b2mml:QuantityValueType"
           minOccurs="0" maxOccurs="1" nillable="true" />
       <xsd:element name="WeighingType" type="xsd:string"
           minOccurs="0" maxOccurs="1" nillable="true" />
       <xsd:element name="WeighingSubType" type="xsd:string"
           minOccurs="0" maxOccurs="1" nillable="true" />
     <!-- add extended elements here -->
 </xsd:sequence>
</xsd:group>
```

Figure 8: CustomerExtendedOperationsBillOfMaterialItem group in B2MML-V0600-Extensions.xsd



```
<xsd:group name = "CustomerExtendedBatchType">
 <xsd:sequence>
       <xsd:element name="ActualPotency" type="b2mml:QuantityValueType"
           minOccurs="0" maxOccurs="1" nillable="true" />
       <xsd:element name="ActualPotency2" type="b2mml:QuantityValueType"
           minOccurs="0" maxOccurs="1" nillable="true" />
     <!-- add extended elements here -->
 </xsd:sequence>
</xsd:group>
```

Figure 9: CustomerExtendedBatchType group in B2MML-V0600-Extensions.xsd

2. Compile the XSD files to generate the java object classes.
   Import the new *ftps-b2mml-v0600-<PharmaSuite version>.jar* library into your project.

Step 3

Extend the *BatchConverter* service with your own implementation

- Example service name: MYCBatchConverter
- Example parameter: MYC_potency2

```
public class MYCBatchConverter extends BatchConverter implements IBatchConverter {
  /** Service to convert a MeasuredValue to the corresponding type in XML */
  private IQuantityValueConverter quantityValueConverterService =
   ServiceFactory.getService(IQuantityValueConverter.class);

  @Override
  public BatchType convert(Batch batch) {
    BatchType convertedBatch = super.convert(batch);
    convertedBatch.setActualPotency(quantityValueConverterService.convert(
     batch.getPotency()));
    convertedBatch.setActualPotency2(quantityValueConverterService.convert(
     getUDAMeasuredValue(batch, "MYC_potency2")));
    return convertedBatch;
  }

  private MeasuredValue getUDAMeasuredValue(Batch batch, String udaname) {
    try {
      return (MeasuredValue) batch.getUDA(udaname);
    } catch (DatasweepException e) {
    }
    return null;
  }
}
```

Step 4

Extend the *OperationsDefinitionsConverter* service with your own implementation

- Example service name: MYCOperationsDefinitionsConverter
- Example parameter: MYC_plannedPotency2
- Example parameter: MYC_weighingSubType

```
public class MYCOrderRelatedBOMItemConverter extends
 com.rockwell.mes.services.batchrecord.impl.converter.OrderRelatedBOMItemConverter
 implements IOrderRelatedBOMItemConverter {
  /** Service to convert a MeasuredValue to the corresponding type in XML */
  private IQuantityValueConverter quantityValueConverterService =
   ServiceFactory.getService(IQuantityValueConverter.class);

  @Override
  public OperationsMaterialBillItemType
    createConvertedOperationsDefinitionRecordFromSplitPositions
     (List<OrderStepInput> allSplitPositions,
      List<OrderStepInput> splitPositionsToConvert, boolean areCancelledMasterOSI,
      boolean areReplaced) throws MESIncompatibleUoMException {
    OperationsMaterialBillItemType orBomItem =
                              super.createConvertedBOMItemFromSplitPositions(
                              allSplitPositions, splitPositionsToConvert, areReplaced);
    OrderStepInput osi = allSplitPositions.get(0);
    orBomItem.setPlannedPotency(quantityValueConverterService.convert((MeasuredValue)
     getUDA(osi, "X_plannedPotency")));
    orBomItem.setPlannedPotency2(quantityValueConverterService.convert((MeasuredValue)
     getUDA(osi, "MYC_plannedPotency2")));
    Object uda = getUDA(osi, "X_weighingType");
    orBomItem.setWeighingType(uda == null ? "" : uda.toString());
    Object uda2 = getUDA(osi, "MYC_weighingSubType");
    orBomItem.setWeighingSubType(uda2 == null ? "" : uda2.toString());
    return orBomItem;
  }

  private Object getUDA(OrderStepInput osi, String udaname) {
    try {
      return osi.getUDA(udaname);
    } catch (DatasweepException e) {
    }
    return null;
  }
}
```

Step 5

The *MESB2MMLJRDataSource* class contains the different mappings between the Jasper report fields and the data source created from PharmaSuite data. It is instantiated using the *ObjectFactory* mechanism, allowing to exchange the implementation via configuration. To extend the class (e.g. as *MYCMESB2MMLJRDataSource*), proceed as follows:

1. Extend the *ReportExpression* class to integrate the new report expressions that will be referenced by your report fields.

```
/**
 * Extension of ReportExpressions to add new fields to the Order related BOM section
 */
public class MYCReportExpressions extends ReportExpressions {
  /** report expression */
  static final String ORDER_RELATED_BOM_ITEM_PLANNED_POTENCY =
   "orderRelatedBOMItem.plannedPotency",
      ORDER_RELATED_BOM_ITEM_PLANNED_POTENCY2 = "orderRelatedBOMItem.plannedPotency2",
      ORDER_RELATED_BOM_ITEM_WEIGHING_TYPE = "orderRelatedBOMItem.weighingType",
      ORDER_RELATED_BOM_ITEM_WEIGHING_SUBTYPE = "orderRelatedBOMItem.weighingSubType";
}
```

2. Implement an enumeration implementing *IEnumOrderRelatedBOMItemType*

```
/** Extended Enumeration of report field descriptions for OrderRelatedBOMItemType */
enum MYCEnumOrderRelatedBOMItemType implements IEnumOrderRelatedBOMItemType {
  // When you add a new element to this enum, do not forget to modify the
  // initOrderRelatedBOMItemTypeMap() and
  // processOrderRelatedBOMItemType(JRField) methods!
  /** report expression */
  PLANNED_POTENCY, PLANNED_POTENCY2, WEIGHING_TYPE, WEIGHING_SUBTYPE
}
```

3. Override the *initOrderRelatedBOMItemTypeMap()* method, which initializes the *orderRelatedBOMItemTypeMap* map to add the new mappings for the **Order-related BOM Items** section.
   Override the *processOrderRelatedBOMItemType(JRField field)* method to get a BOM item value from a Jasper report field using the extended mapping.

```
/**
 * Extension of MESB2MMLJRDataSource to integrate the planned actual Potency2
 * to the order related BOM section
*/
public class MYCMESB2MMLJRDataSource extends MESB2MMLJRDataSource {
  /**
   * @param beanCollection data beans list
   * @param isUseFieldDescription if field descriptor shall be used or the name
   */
  public MYCMESB2MMLJRDataSource(Collection beanCollection, boolean
   isUseFieldDescription) {
    super(beanCollection, isUseFieldDescription);
  }

  /** @param beanCollection data beans list */
  public MYCMESB2MMLJRDataSource(Collection beanCollection) {
    super(beanCollection);
  }

  /** Initialize expression map for OrderRelatedBOMItemType */
  @Override
  protected void initOrderRelatedBOMItemTypeMap() {
    super.initOrderRelatedBOMItemTypeMap();
    Map<String, IEnumOrderRelatedBOMItemType> orderRelatedBOMItemTypeMap =
     getOrderRelatedBOMItemTypeMap();
    orderRelatedBOMItemTypeMap.put(MYCReportExpressions
     .ORDER_RELATED_BOM_ITEM_PLANNED_POTENCY, MYCEnumOrderRelatedBOMItemType
     .PLANNED_POTENCY);
    orderRelatedBOMItemTypeMap.put(MYCReportExpressions
     .ORDER_RELATED_BOM_ITEM_PLANNED_POTENCY2,
     MYCEnumOrderRelatedBOMItemType.PLANNED_POTENCY2);
    orderRelatedBOMItemTypeMap.put(MYCReportExpressions
     .ORDER_RELATED_BOM_ITEM_WEIGHING_TYPE, MYCEnumOrderRelatedBOMItemType
     .WEIGHING_TYPE);
    orderRelatedBOMItemTypeMap.put(MYCReportExpressions
     .ORDER_RELATED_BOM_ITEM_WEIGHING_SUBTYPE, MYCEnumOrderRelatedBOMItemType
     .WEIGHING_SUBTYPE);
  }
  /**
   * Process OrderRelatedBOMItemType expressions
   *
   * @param field to process
   * @return value of the field
   * @throws JRException from parent implementation
   */
  @Override
  protected Object processOrderRelatedBOMItemType(JRField field) throws JRException {
    IEnumOrderRelatedBOMItemType iEnumOrderRelatedBOMItemType =
     getOrderRelatedBOMItemTypeMap().get(field.getDescription());
    OperationsMaterialBillItemType castedCurrentBean = (OperationsMaterialBillItemType)
     getCurrentBean();
    if (iEnumOrderRelatedBOMItemType instanceof MYCEnumOrderRelatedBOMItemType) {
      // If the current description reference an extended BOMItemType element
      MYCEnumOrderRelatedBOMItemType enumValue = (MYCEnumOrderRelatedBOMItemType)
```

```
        iEnumOrderRelatedBOMItemType;
      if (enumValue != null) {
        switch (enumValue) {
        case PLANNED_POTENCY:
          return toString(castedCurrentBean.getPlannedPotency());
        case PLANNED_POTENCY2:
          return toString(castedCurrentBean.getPlannedPotency2());
        case WEIGHING_TYPE:
          return getLocalizedChoiceItem("WeighingType",
           castedCurrentBean.getWeighingType());
        case WEIGHING_SUBTYPE:
          return getLocalizedChoiceItem("MYC_weighingSubType",
           castedCurrentBean.getWeighingSubType());
        default:
          break;
        }
      }
      // for enumValue == null and default from the switch
      return getFieldValue(castedCurrentBean, field);
    } else {
      EnumOrderRelatedBOMItemType enumValue = (EnumOrderRelatedBOMItemType)
       iEnumOrderRelatedBOMItemType;
      if (enumValue != null) {
        switch (enumValue) {
        case IDENTIFIED_BATCH_IDS:
          return batchListToStringList(castedCurrentBean.getIdentifiedBatchIDs()
           .getBatchList());
        default:
          break;
        }
      }
      return super.processOrderRelatedBOMItemType(field);

    }
  }

  private String getLocalizedChoiceItem(String choiceListName, String choiceValue) {
    if (choiceValue == null || choiceValue.isEmpty()) {
      return "";
    }
    IMESChoiceElement chElem = MESChoiceListHelper.getChoiceElement(choiceListName,
     Long.valueOf(choiceValue));
    return chElem == null ? "" : chElem.getLocalizedMessage();
  }

  private String toString(QuantityValueType quantityValue) {
    if (quantityValue == null || quantityValue.isNil()) {
      return "";
    }
    try {
      String qtyString = quantityValue.getQuantityString().getStringValue();
      String uom = quantityValue.getUnitOfMeasure().isNil() ? "" : " " +
       quantityValue.getUnitOfMeasure().getStringValue();
      return BigDecimalUtilities.fromStringAsDecimal(qtyString) + uom;
    } catch (ParseException e) {
      throw new MESRuntimeException(e);
    }
  }

  /**
   * Create a list of the String representation of batches
   *
   * @param batchList the list containing the typed batches
   * @return a list containing the batch identifiers
   */
  private List<String> batchListToStringList(List<BatchType> batchList) {
    ArrayList<String> stringList = new ArrayList<String>();
    for (BatchType batch : batchList) {
      stringList.add(batch.getIdentifier().getStringValue() + "(" +
       toString(batch.getActualPotency()) + ", "
       + toString(batch.getActualPotency2()) + ")");
    }
    return stringList;
  }

}
```

Step 6

Configure the new service implementation by overriding the default service configuration. For this purpose, proceed as follows:

- For details, see [A8] (page 56)

1. Create the extension-config.xml configuration file and add a reference to the new service configuration file (e.g. *myc-batchreport.xml* and *myc-batchrecord.xml*).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Custom config.xml including extended service definitions -->
<configuration name="extension-config">
   <hierarchicalXml fileName="myc-batchreport.xml" config-optional="false"  />
   <hierarchicalXml fileName="myc-batchrecord.xml" config-optional="false"  />
</configuration>
```

2. Create your service configuration file (e.g. *myc-batchrecord.xml*) with the implementation class of your service implementation (e.g. *com.rockwell.custmes.services.batchrecord.impl.converter.MYCBatchConverter* and *.MYCOperationsDefinitionsConverter*).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
<MasterRecipeConverter>
  <implementationClass>com.rockwell.custmes.services.batchrecord.impl
   .converter.MYCMasterRecipeConverter
  </implementationClass>
</MasterRecipeConverter>
  <BatchConverter>
    <implementationClass>com.rockwell.custmes.services.batchrecord.impl
     .converter.MYCBatchConverter
    </implementationClass>
  </BatchConverter>
  <OperationsDefinitionsConverter>
    <implementationClass>com.rockwell.custmes.services.batchrecord.impl
     .converter.MYCOperationsDefinitionsConverter
    </implementationClass>
  </OperationsDefinitionsConverter>
</configuration>
```

3. Create your service configuration file (e.g. *myc-batchreport.xml*) with the implementation class of your service implementation (e.g. *com.rockwell.custmes.services.batchreport.impl.MYCMESB2MMLJRDataSource*) .

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
 <MasterRecipeDocumentWrapper>
  <implementationClass>com.rockwell.custmes.services.batchreport.impl
   .MYCMasterRecipeDocumentWrapper</implementationClass>
 </MasterRecipeDocumentWrapper>
  <MESB2MMLJRDataSource>
    <implementationClass>
      com.rockwell.custmes.services.batchreport.impl.MYCMESB2MMLJRDataSource
    </implementationClass>
  </MESB2MMLJRDataSource>
</configuration>
```

4. Add your extension configuration files to the classpath by deploying them in a JAR file.

Step 7

If you display the second potency in the **Bill of Materials** section of the master recipe report (page 39), you can skip this step.

Add messages for the two new column headers to the **PS-BatchReport** message pack

- Example message ID: WeighingType_Label
- Example message: Weighing type
- Example message ID: Planned_Potency_Label
- Example message: Planned potency

Step 8

Modify your version of the batch report with Jaspersoft Studio to match your needs.

- For details, see [A10] (page 56)

1. In the PharmaSuite report manager, export the **PS-BatchReport-OrderRelatedBOMItem** and **PS-BatchReport-OrderRelatedBOMItems** sub-report designs.
2. In Jaspersoft Studio, add the following items to **PS-BatchReport-OrderRelatedBOMItem**:
   - The description must match the identifiers defined in the *MYCReportExpressions* class (see Step 5):
   - String field: PS_BOMItemPlannedPotency
     - Description: orderRelatedBOMItem.plannedPotency
   - String field: PS_BOMItemPlannedPotency2
     - Description: orderRelatedBOMItem.plannedPotency2
   - String field: PS_BOMItemWeighingType
     - Description: orderRelatedBOMItem.weighingType
   - String field: PS_BOMItemWeighingSubType
     - Description: orderRelatedBOMItem.weighingSubType
   - Place the two fields listed below at an appropriate position of the table layout (e.g. between **Identified batch IDs** and **Planned quantity**).
     - Add a **Detail 1** field with a **Text Field Expression**: $F{PS_BOMItemWeighingType} + "\n" + $F{PS_BOMItemWeighingSubType}
     - Add a **Detail 1** field with a **Text Field Expression**: $F{PS_BOMItemPlannedPotency} + ", " + $F{PS_BOMItemPlannedPotency2}
     - Repeat the last 2 steps for **Detail 2** band at the same position of the table and set **Strike Through** for these fields.

3. Add the following column headers to **PS-BatchReport-OrderRelatedBOMItems** between **OrderRelatedBOMItem_IdentifiedBatchIDs_Label** and **OrderRelatedBOMItem_PlannedQuantity_Label**:
   - Column header with a **Text Field Expression**:
     $R{WeighingType_Label}
   - Column header with a **Text Field Expression**:
     $R{Planned_Potency_Label}
4. Import the report design by means of the **mes_PS-BatchReportManager** form.

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ UDA for second potency (Part (material), ProcessBomItem, OrderStepInput, and Batch object) ■ Extend application table (X_MaterialParameter, X_TransactionHistory) ■ Create choice list ■ Weighing Subtype Field Attribute at Material Parameters	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting and Adding Field Attributes ■ Adding Field Attributes by Means of UDAs ■ Adding Field Attributes to Application Table Objects ■ Choice Lists ■ List Editors	PSCEV2-GR010B-EN-E
A2	■ UDA's appearance in the property pane of the Production Management Client ■ Change Batch Attributes action, Transaction History action	PharmaSuite Technical Manual Configuration & Extension - Volume 1: Adapting the Use Cases of the Production Management Client ■ Configuring the UDA's Appearance ■ Modifying a Production Management Client Form with a Data Node	PSCEV1-GR010B-EN-E
A3	■ Initialize material parameters	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Initialization of Material Parameters in Recipe and Workflow Designer	PSCEV3-GR010B-EN-E
A4	■ Attribute's appearance in the Parameter Panel and Setlist of Recipe and Workflow Designer	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Parameter Panel of Recipe and Workflow Designer ■ Adjusting the Configuration of Custom Attributes	PSCEV3-GR010B-EN-E
A5	■ not used	not used	not used

No.	Keyword	Document Title	Part Number
A6	■ Configuration of objectBinderEnhanced control and transaction history	PharmaSuite Technical Manual Configuration & Extension - Volume 1: Creating and Adapting Forms ■ Adapting UI Controls by Means of Form Controls	PSCEV1-GR010B-EN-E
A7	■ not used	not used	not used
A8	■ Extending the WDOrderStepInputService service ■ Extending the WDUOMConversionService Service ■ Configure new service implementations ■ Order step input objects and their helper classes ■ Calculation related to sublot replacement	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Managing Services ■ Adapting the Behavior of Existing Services ■ Configuring Service Implementations ■ Helper Classes for Working with Order Step Input Objects	PSCEV2-GR010B-EN-E
A9	■ Create new versions of the Identify material and Weigh phases ■ Phase Copy Tool ■ Extend runtime phase data	PharmaSuite Technical Manual Developing System Building Blocks: ■ Creating a New Version of a Phase and their Related Parameter Classes ■ Changing the phase version and the parameters versions - Working with the Phase Copy Tool ■ Creating a New Version of a Phase and their Related Parameter Classes - Changing the phase version and extending the phase runtime data or phase output data	PSBB-PM009A-EN-E
A10	■ Batch report ■ Phase-specific batch report	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Batch Reports in PharmaSuite	PSCEV2-GR010B-EN-E

No.	Keyword	Document Title	Part Number
A11	■ Extend sub-report	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Batch Reports in PharmaSuite ■ Displaying the Structure of a Report	PSCEV2-GR010B-EN-E
A12	■ Master recipe	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Master Recipe Reports in PharmaSuite	PSCEV2-GR010B-EN-E
A13	■ Batch record	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Working with the Batch Record API	PSCEV2-GR010B-EN-E
A14	■ Export master recipe	PharmaSuite Technical Manual Administration: Exporting and Importing Master Recipes, Master Workflows, and Building Blocks	PSAD-RM010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Chapter 5

Adding the Second Potency Attribute to ERP BOMs

This section describes how to add attributes to ERP BOMs, using the example of the second potency attribute. The example is a continuation of the "Processing Materials with a Second Potency" extension use case (page 13).

The second potency attribute of the ERP BOMs is automatically populated into the related attribute of the material parameters.
In case the same attribute also exists as material master data, the default value for the material parameters will be taken from the material master data if no value is defined in the ERP BOM.

The description covers the following areas:

- Support of second potency attributes for materials in the Production Management Client and material parameters and ERP BOMs in Recipe and Workflow Designer.
  - Add the second potency attribute to materials (page 62), material parameters (page 63), and ERP BOMs (page 64).
- Optional:
  Adapt the copying mechanism (page 65) for columns with different names
- Optional:
  Adapt consistency checks (page 65) related to ERP BOMs.

Adding the Second Potency Field Attribute to the Material Object

TIP

This section is identical to section "Adding the Second Potency Field Attribute to the Material Object" (page 15) of the "Processing Materials with a Second Potency" extension use case (page 13).

To add the second potency field attribute to the **Material** (part) object, proceed as follows:

Step 1

Add a new UDA for the second potency to the **Material** object

- For details, see [A1] (page 69)
- The UDA must be of the **MeasuredValue** type.
- Example UDA: MYC_plannedPotency2

Step 2

Configure the new UDA's appearance in the property pane of the Production Management Client

- For details, see [A2] (page 69)

1. Add a Data Dictionary element for the new UDA to the **Part [Default]** Data Dictionary class
   - Example Data Dictionary element: UDA_MYC_plannedPotency2
   - Select the **catWeighing** category to assign the property to the weighing properties.
   - Select the **Visible in Setlist** option.
2. Add messages for the labels of the new UDA to the **DataDictionary_Default_Part** message pack
   - Example message ID: UDA_MYC_plannedPotency2_Label
   - Example message ID (description pane): UDA_MYC_plannedPotency2_Hint

Adding the Second Potency Field Attribute to the Material Parameters

TIP

The steps are similar to the steps described in "Adding the Second Potency Field Attribute to the Material Parameters" (page 16) of the "Processing Materials with a Second Potency" extension use case (page 13).

To add the second potency field attribute as an attribute to the material parameters in Recipe and Workflow Designer, proceed as follows:

Step 1

Add an attribute to the material parameters

1. Extend the **X_MaterialParameter** application table
    - For details, see [A1] (page 69)
    - The column must be of the **MeasuredValue** type.
    - Example column: MYC_plannedPotency2

TIP

Make sure that name and type of the new column are identical to the name and type of the new UDA of the **Material** object. Then, the UDA's value is automatically copied to the material parameters.

Step 2

Configure the new attribute's appearance in the (material) Parameter Panel and the Setlist of Recipe and Workflow Designer

- For details, see [A4] (page 69)

1. Add a Data Dictionary element for the new attribute to the *MESMaterialParameterCustomerConfig* Data Dictionary class
    - Example Data Dictionary element: MYC_plannedPotency2
    - Select the **Visible in Setlist** option (applies to Setlist and Parameter Panel).
2. Add a message for the label of the new attribute to the **DataDictionary_Default_MESMaterialParameterCustomerConfig** message pack
    - Example message ID: MYC_plannedPotency2_Label

Adding the Second Potency Field Attribute to the ERP BOM

To add the second potency field attribute as an attribute to the ERP BOM items, proceed as follows:

Step 1

Add an attribute to the ERP BOM items

1. Extend the **X_ERPBomItem** application table
   - For details, see [A1] (page 69)
   - The column must be of the **MeasuredValue** type.
   - Example column: MYC_plannedPotency2

Step 2

Configure the new attribute's appearance in the (material) Parameter Panel and the Setlist of Recipe and Workflow Designer

- For details, see [A5] (page 69)

1. Add a Data Dictionary element for the new attribute to the *MESERPBomItemCustomerConfig* Data Dictionary class.
   - Example Data Dictionary element: MYC_plannedPotency2
   - Select the **Visible in Setlist** option (applies to Setlist and Parameter Panel).
2. Add a message for the label of the new attribute to the **DataDictionary_Default_MESERPBomItemCustomerConfig** message pack
   - Example message ID: MYC_plannedPotency2_Label

Copying Columns with Different Column Names

This step is optional.

By default, PharmaSuite copies the values of the columns with the same name. If your system is configured with different column names (e.g. MYC_plannedPotency2 and Potency2), you have to adapt the default behavior of PharmaSuite.

To enable the system to copy columns with different column names, proceed as follows:

Step 1

Copy columns with different column names

- For details, see [A3] (page 69)
- The *MyCustomerMaterialParameterCustomAttributesHandler* class copies columns with the names MYC_plannedPotency2 and Potency2.

```
public class MyCustomerMaterialParameterCustomAttributesHandler
              extends MESMaterialParameterCustomAttributesHandler {
    @Override
    public void copyCustomAttributes(Part material, IMESMaterialParameter parameter) {
        super.copyCustomAttributes(material, parameter) {
        // fill further custom columns
        Object newValue = material.getUDA("Potency2");
        parameter.setValue("MYC_plannedPotency", newValue);
    }
    @Override
    public void copyCustomAttributes(IMESERPBomItem erpBomItem, IMESMaterialParameter
                      parameter, boolean copyNullValues)
        super.copyCustomAttributes(erpBomItem, parameter, copyNullValues)
        // fill further custom columns
        Object newValue = erpBomItem.getValue("Potency2");
        parameter.setValue("MYC_plannedPotency", newValue);
    }
}
```

Adapting Consistency Checks Related to ERP BOMs

This step is optional.

PharmaSuite checks the consistency between a material parameter and its ERP BOM item. In case the quantity is not compatible or the material is different, the system displays corresponding messages in the Messages window.
If required, existing checks can be modified or new checks can be added.

To adapt a consistency check, proceed as follows:

Step 1

Adapt a check

- For details, see [A6] (page 69)

- The *mycERPBomItemSecondPotencyCheck* class compares the contents of the **MYC_plannedPotency2** attribute of the material parameter with the corresponding value of the ERP BOM item. If the value of the ERP BOM item is not empty, then both values must be identical. Otherwise, the system displays an error message in the Messages window.

```
package com.rockwell.custmes.apps.recipeeditor.impl.hierarchy.check;
import [...]
public class mycERPBomItemSecondPotencyCheck extends
             AbstractRecipeStructureSingleParameterCheck {
  /** LOGGER */
  protected static final Log LOGGER =
                   LogFactory.getLog(mycERPBomItemSecondPotencyCheck.class);
  /**
   * The IDs of all localized messages (message pack
   * 'ui_RecipeDesigner_Checks'.
   */
  static final String // nl
  ERP_BOMITEM_CHECK_SECOND_POTENCY_DIFFERENT_MSG_ID =
                   "ERPBOMItemSecondPotencyCheck_ValuesDifferent_ErrorMsg";
  // Example: "The {1} has an unsuitable material input parameter, since its
  // second potency do not correspond to the those of the {2} position of the
  // ERP BOM; required potency: {5}, current potency: {4}. ({0})"
  /** Stores the ERP BOM to improve performance. */
  private IMESERPBom erpBom;
  /** The column- / property-name of the second potency. */
  private static final String COLUMN_NAME_SECOND_POTENCY = "MYC_plannedPotency2";
  /**
   * The name of the properties to observe
   */
  private static final List<String> PROPERTY_NAMES_TO_OBSERVE = Arrays.asList(//
      IMaterialParameter.PROPERTY_BOM_POSITION, //
      COLUMN_NAME_SECOND_POTENCY //
      );
  /** Execute check only for recipes with ERP BOM. */
  @Override
  public boolean isApplicableForRecipeStructureModel(IRecipeStructureModel
                recipeStructureModel) {
    return recipeStructureModel.getERPBom() != null;
  }
  /** Execute check only for material parameters. */
  @Override
  protected boolean isRelevantParameterType(ParameterType parameterType) {
    return parameterType == ParameterType.MATERIAL;
  }
  /** Execute check only on phase level. */
  @Override
  protected boolean isRelevantRecipeElement(IRecipeElement recipeElement) {
    // check only on phase level
    return recipeElement.getLevel().equals(HierarchyLevel.PHASE);
  }
  /** Execute check only for the changes of BOM-position or second potency. */
  @Override
  protected boolean isRelevantPropertyChange(PropertyChangeEvent event) {
    String propName = event.getPropertyName();
    boolean check = PROPERTY_NAMES_TO_OBSERVE.contains(propName);
    return check;
  }
  /** Set local attribute 'erpBOM'. */
  @Override
  public void setRecipeStructureModel(IRecipeStructureModel recipeStructureModel) {
    super.setRecipeStructureModel(recipeStructureModel);
    this.erpBom = getRecipeStructureModel().getERPBom();
  }
  /**
   * Check the material parameter, if
   * <ul>
   * <li>it has a ERP BOM item on same position
   * <li>the ERP BOM item on same position has same second potency
   * </ul>
   *
   * @param parameter The parameter to check
   */
  @Override
  protected void checkParameter(IParameter parameter) {
    IMaterialParameter materialParameter = (IMaterialParameter) parameter;
    if (!materialParameter.getSubType().equals(ParameterSubType.IN)) {
      // only check input parameters her.
      return;
    }
    String bomPosition = materialParameter.getBOMPosition();
    if (StringUtils.isBlank(bomPosition)) {
      // local material is allowed and will not checked against ERP BOM
      return;
    }
    IMESERPBomItem aERPBomItem = erpBom.getItem(bomPosition);
    if (aERPBomItem == null) {
      IRecipeElement recipeElement = materialParameter.getParent();
      String levelNameLowerCase =
            HierarchyLevelAndStructureTypeTraits.getLocalizedName
                          (recipeElement.getLevel(), LetterCase.LOWER_CASE, true);
      String[] msgArgs = { recipeElement.getName(), levelNameLowerCase, bomPosition };
      addErrorMessage(materialParameter,
                      CheckMessageType.ERP_BOM_ITEM_NONEXISTING_ITEM,
          "ERPBOMItemPhaseLevelCheck_NonExistingERPBomItem_ErrorMsg", msgArgs);
      return;
    }
    if (isConsistent(aERPBomItem, materialParameter)) {
      Part aERPBomPart = aERPBomItem.getMaterial();
      LOGGER.debug("bom position [OK]:" + bomPosition +
         " has compatible BOM Item assigned. [" + aERPBomPart.getPartNumber() + "]");
    }
  }
  /**
   * Add an error message to the message model
   *
   * @param materialParameter material parameter
   * @param messageTypeString the type of the message
   * @param messageIdString the message string
   * @param msgArgs the message arguments
   */
  protected void addErrorMessage(IMaterialParameter materialParameter, String
                               messageTypeString,
      String messageIdString, Object[] msgArgs) {
    String i18nMsg = getLocalizedMessage(messageIdString, msgArgs);
    ICheckMessagesModel messages = getCheckMessagesModel();
    messages.addParameterRelatedMessage(i18nMsg, messageTypeString, //
        CheckMessageLevel.ERROR, materialParameter, null, this);
  }
  /**
   * Checks, if the attribute "MYC_plannedPotency2" of the material parameter
   * and the ERP BOM item are consistent.
   *
   * @param aERPBomItem a ERP BOM item
   * @param materialParameter material parameter
   * @return <code>true</code> , if the attributes 'MYC_plannedPotency2' are
   *         consistent, else <code> false</code>
   */
  protected boolean isConsistent(IMESERPBomItem aERPBomItem, IMaterialParameter
                               materialParameter) {
    Object originalValue = aERPBomItem.getATRow().getValue(COLUMN_NAME_SECOND_POTENCY);
    if (originalValue == null) {
      return true; // no check necessary
    }
    // same value expected
    Object currentValue =
       materialParameter.getInstance().getATRow().getValue(COLUMN_NAME_SECOND_POTENCY);
    boolean valid = ObjectUtils.equals(originalValue, currentValue);
    if (valid) {
      return true; // values identically
    }
    // values different => error message
    IRecipeElement recipeElement = materialParameter.getParent();
    String levelNameLowerCase =
           HierarchyLevelAndStructureTypeTraits.getLocalizedName
                          (recipeElement.getLevel(), LetterCase.LOWER_CASE, true);
    Part aERPBomPart = aERPBomItem.getMaterial();
    String shortDescription = MESNamedUDAPart.getShortDescription(aERPBomPart);
    String[] msgArgs =
        { recipeElement.getName(), levelNameLowerCase, aERPBomItem.getPosition(),
          (shortDescription == null ? "" : shortDescription), //
          (currentValue == null) ? "" : currentValue.toString(),
           originalValue.toString() };
    addErrorMessage(materialParameter, "0001CUST",
         ERP_BOMITEM_CHECK_SECOND_POTENCY_DIFFERENT_MSG_ID, msgArgs);
    return false;
  }
```

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ UDA for second potency (Part (material) object) ■ Extend application table (X_MaterialParameter, X_ERPBomItem)	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting and Adding Field Attributes ■ Adding Field Attributes by Means of UDAs ■ Adding Field Attributes to Application Table Objects	PSCEV2-GR010B-EN-E
A2	■ UDA's appearance in the property pane of the Production Management Client	PharmaSuite Technical Manual Configuration & Extension - Volume 1: Adapting the Use Cases of the Production Management Client ■ Configuring the UDA's Appearance	PSCEV1-GR010B-EN-E
A3	■ Copy columns with different column names	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Initialization of Material Parameters in Recipe and Workflow Designer	PSCEV3-GR010B-EN-E
A4	■ Attribute's appearance in the Parameter Panel and Setlist of Recipe and Workflow Designer (Material)	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Parameter Panel of Recipe and Workflow Designer ■ Adjusting the Configuration of Custom Attributes	PSCEV3-GR010B-EN-E
A5	■ Attribute's appearance in the Parameter Panel and Setlist of Recipe and Workflow Designer (ERP BOM)	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Setlist of Recipe and Workflow Designer ■ Adjusting the Configuration of Custom Attributes	PSCEV3-GR010B-EN-E

No.	Keyword	Document Title	Part Number
A6	■ Consistency check	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Modifying and Adding Checks ■ Master Recipe and Custom Building Block Checks	PSCEV2-GR010B-EN-E

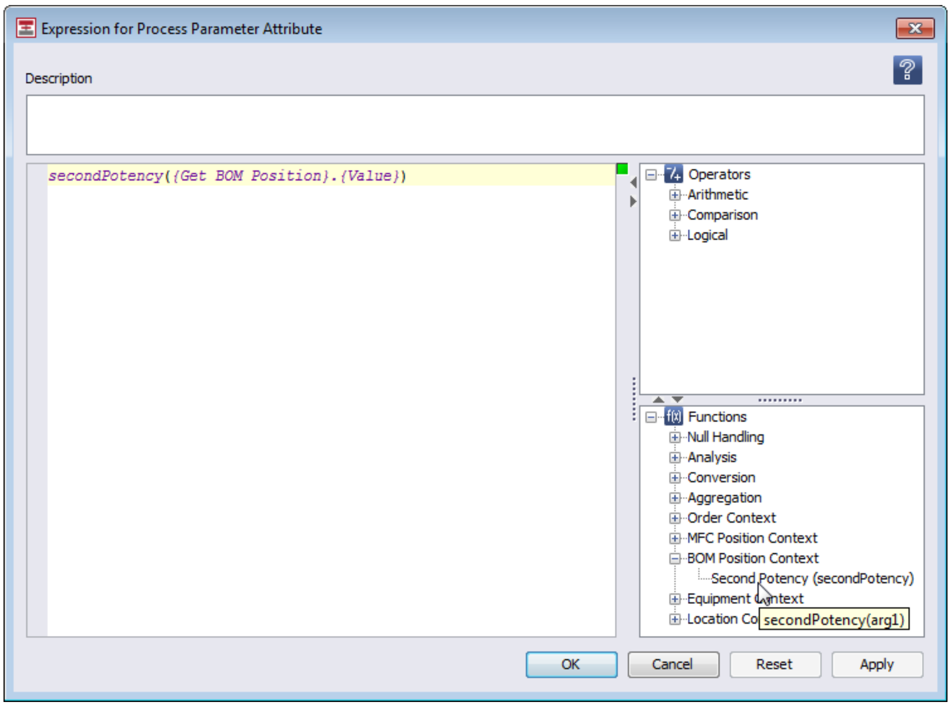
TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Adding the Second Potency Function to the Expression Editor

This section describes how to add the Second Potency function to the Expression editor of Recipe and Workflow Designer. Afterwards you can retrieve the value of the second potency of your material via a context-related function in the Expression editor.

The use case assumes that the "Processing Materials with a Second Potency" extension use case (page 13) has been implemented, especially that a second potency field attribute has been added to the **ProcessBomItem** object (i.e. a UDA named MYC_plannedPotency2 has been added), see "Adding the Second Potency Field Attribute to the ProcessBomItem Object" (page 18).



```
secondPotency({Get BOM Position}.{Value})
```

Figure 10: Second Potency function in Expression editor

To add the Second Potency function to the Expression editor, proceed as follows:

Step 1

Configure your application configuration to use a custom context function configuration

- For details, see [A1] (page 73)

Step 2

Add a **FunctionDescriptor** section to your configuration list

- For details, see [A1] (page 73)

-
  ```
  […]
  <FunctionDescriptor groupId="065_BOMItemContext" sortIndex="100">
    <FunctionName>secondPotency</FunctionName>
    <ClassName>BomItem</ClassName>
    <PropertyPath>UDA_MYC_plannedPotency2</PropertyPath>
  </FunctionDescriptor>
  […]
  ```
- Example group identifier in **BOM Position Context** group: 065_BOMItemContext
- Example UDA of **ProcessBomItem** object: MYC_plannedPotency2

Step 3

Enable localization of the function name in the **ui_ExpressionEditor** message pack

- For details, see [A1] (page 73)
- Example message ID: Language_Function_secondPotency_Label
- Example message: Second Potency

TIP

If a context-related property of interest cannot be retrieved using the given root objects like ControlRecipe, BomItem, Station, or Room, you can still implement an arbitrary user-defined function to retrieve that property using custom code instead of adapting the configuration. For details, see [A2] (page 73).

Since a BOM item context group is not available in <PS> by default, the example creates a new group. Hence, the group name also needs to be localized. In the **ui_ExpressionEditor** message pack, create an appropriate message (example message ID: **LanguageTree_Functions_Grp_065_BOMItemContext_Label**, example message: **BOM item context**).

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Use custom context function configuration ■ Add a FunctionDescriptor ■ Localization	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Expression Editor of Recipe and Workflow Designer and Data Manager Providing a Context-related Function ■ Configuring PharmaSuite ■ Adding a New Function ■ Enable Localization and Online Help	PSCEV3-GR010B-EN-E
A2	■ Context-related property, root object	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring the Expression Editor of Recipe and Workflow Designer and Data Manager Providing an Arbitrary Function	PSCEV3-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Managing Batches in the ConditionallyReleased Status

This section describes how to add the **ConditionallyReleased** batch status to PharmaSuite. This intermediate status allows you to optimize your production processes by making use of certain raw materials even if the final QC test results are not yet available.

TIP

For FSMs (flexible state models), structural changes (i.e. changes to states and transitions) are not allowed in order to enable a later system migration. Modifications of semantic properties can be applied to the standard FSM. They do not impact a system migration.
In the extension use case, we create a new FSM.

The standard FSM for the batch status provides three statuses and transitions between the statuses.

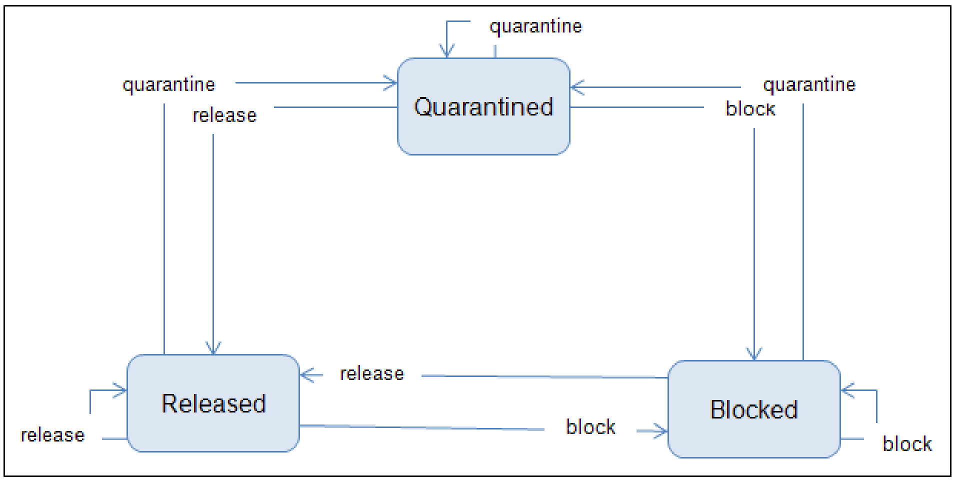


Figure 11: Simplified presentation of the batch quality status FSM (BatchQuality)

With this extension use case, the **ConditionallyReleased** status and the **releaseConditionally** transition will be added to the FSM.

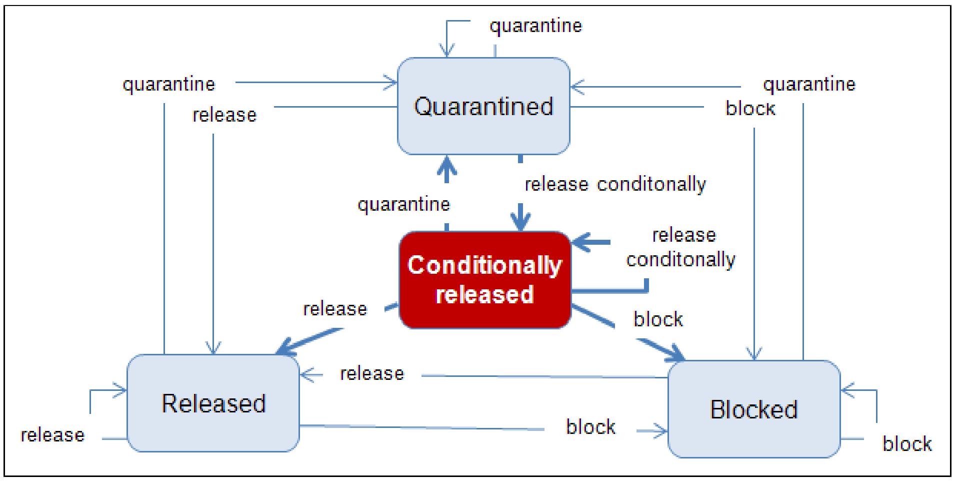


Figure 12: Simplified presentation of the extended batch quality status FSM (MYC_BatchQualityWithCondRelease)

The description covers the following areas:

- Create a new FSM (page 76) to support the **ConditionallyReleased** status and the **releaseConditionally** transition.
- Configure the new FSM (page 79) to be used as the standard FSM for batches.

This section assumes that you are familiar with the terms and concepts of FSMs as described in "Process Designer Online Help" [B1] (page 80) and "Configuring Flexible State Models" in Volume 2 of the "Technical Manual Configuration and Extension" [A1] (page 80).

Creating an FSM with the ConditionallyReleased Status

To create a new FSM for the **Batch** object, proceed as follows:

Step 1

Add messages for the new **ConditionallyReleased** status and the new **releaseConditionally** transition to the **fsm_BatchQuality** message pack

- Example message ID: state_ConditionallyReleased
- Example message: Conditionally released
- Example message ID: transition_ReleaseConditionally
- Example message: Change to "Conditionally released"

Step 2

Copy the **BatchQuality** FSM, add your vendor code to the name, and edit the copy in the following steps.

- Example name: MYC_BatchQualityWithCondRelease

Step 3

Add the **ConditionallyReleased** status to the **MYC_BatchQualityWithCondRelease** FSM

- For details, see [B2] (page 80)
- Example properties of the new status:
  - Message ID: state_ConditionallyReleased
  - Message pack: fsm_BatchQuality
  - Name: ConditionallyReleased

Step 4

Add the **releaseConditionally** transition to the **MYC_BatchQualityWithCondRelease** FSM

- For details, see [B2] (page 80)
- Example properties of the new transition:
  - Message ID: transition_ReleaseConditionally
  - Message pack: fsm_BatchQuality
  - Name: releaseConditionally

Step 5

Add transition instances to and from the **ConditionallyReleased** status of the **MYC_BatchQualityWithCondRelease** FSM

- For details, see [B2] (page 80)
- Example transition instances:

Transition instance	From	To	Transition
1	Quarantined	ConditionallyReleased	releaseConditionally
2	ConditionallyReleased	Blocked	block
3	ConditionallyReleased	Quarantined	quarantine
4	ConditionallyReleased	Released	release
5	ConditionallyReleased	ConditionallyReleased	releaseConditionally

Step 6

Add semantic properties to the new transitions instances of the **MYC_BatchQualityWithCondRelease** FSM

- For details, see [A2], [A3], and [B2] (page 80)
- Example semantic properties:

Transition instance	Semantic property: listener.CheckBatchExpiryDate	Semantic property: listener.CheckBatchRetestDate
1	yes	yes
2	no	no
3	yes	no
4	yes	yes
5	yes	yes

Step 7

Add semantic properties to the **ConditionallyReleased** status of the **MYC_BatchQualityWithCondRelease** FSM

- For details, see [A3] and [B2] (page 80)
- Example semantic properties:
  - sys.allow.Batch.changeExpirationTime
  - sys.allow.Batch.changeName
  - sys.allow.Batch.createSublots
  - sys.allow.Batch.save
  - sys.allow.Batch.transferSublots
  - bq.expiration.expiryDateExpired
  - bq.rank

Step 8

Override the default value of the **bq.rank** semantic property of the **ConditionallyReleased** status of the **MYC_BatchQualityWithCondRelease** FSM to define the rank of the **ConditionallyReleased** status

- For details, see [A2], [A3], and [B3] (page 80)
- The value must be between 20 (Quarantined) and 30 (Released).
- Example rank: 25

Configuring the New FSM as Standard FSM for Batches

To configure the **MYC_BatchQualityWithCondRelease** FSM as the standard FSM for the **Batch** object, proceed as follows:

Step 1

Configure the **BatchQuality** FSM in the **Batch** FSM configuration as non-standard

- For details, see [B2] (page 80)
- Set the **Relationship Type** to **Manual**.

Step 2

Configure the **MYC_BatchQualityWithCondRelease** FSM in the **Batch** FSM configuration as standard

- For details, see [B2] (page 80)
- Add the **MYC_BatchQualityWithCondRelease** FSM with the **Relationship Type** set to **Automatic**.

TIP

If your database does not contain batches yet, you can remove the relationship with the **BatchQuality** FSM and the **BatchQuality** FSM itself.

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Flexible State Model	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Configuring Flexible State Models	PSCEV2-GR010B-EN-E
A2	■ Add semantic properties to transition instance	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Configuring Flexible State Models ■ Adapting the BatchQuality FSM	PSCEV2-GR010B-EN-E
A3	■ Semantic properties	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Configuring Flexible State Models ■ Configuring PharmaSuite with Flexible State Models	PSCEV2-GR010B-EN-E
A4	■ Choice list manager	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting and Adding Field Attributes ■ Choice Lists	PSCEV2-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

The following documents provide more details relevant to the implementation of the extension use case and are distributed with the FactoryTalk ProductionCentre installation.

No.	Keyword	Document Title	Part Number
B1	■ Flexible State Model	Process Designer Online Help: ■ Flexible State Models	N/A
B2	■ Add status to FSM ■ Add transition to FSM ■ Add transition instances to status ■ Add semantic properties to transition instance ■ Add semantic properties to status	Process Designer Online Help: Flexible State Models ■ Creating FSM States, Transitions, and Transition Instances	N/A
B3	■ Overwrite value of semantic property	Process Designer Online Help: Flexible State Models ■ Configuring Additional FSM Features	N/A
B4	■ Relationship type and FSM configuration	Process Designer Online Help: ■ FSM Tab Interface	N/A

TIP

To access the "Process Designer Online Help", use the following syntax: *http://<MES-PS-HOST>:8080/PlantOperations/docs/help/pd/index.htm*, where <MES-PS-HOST> is the name of your PharmaSuite server. To view the online help, the Apache Tomcat of the FactoryTalk ProductionCentre installation must be running.

Adapting the Approval Workflow of Master Recipes

This section describes how to adapt the version graph for batch-specific master recipes of PharmaSuite. New **Edit** and **Verification** statuses will be added in order to extend the approval workflow with additional edit and review options.

TIP

For FSMs (flexible state models), structural changes (i.e. changes to states and transitions) are not allowed in order to enable a later system migration. Modifications of semantic properties can be applied to the standard FSM. They do not impact a system migration.
In the extension use case, we create a new FSM.

The standard FSM for the master recipe provides six statuses and various transitions between the statuses. There is one **Edit** and one **Verification** status.

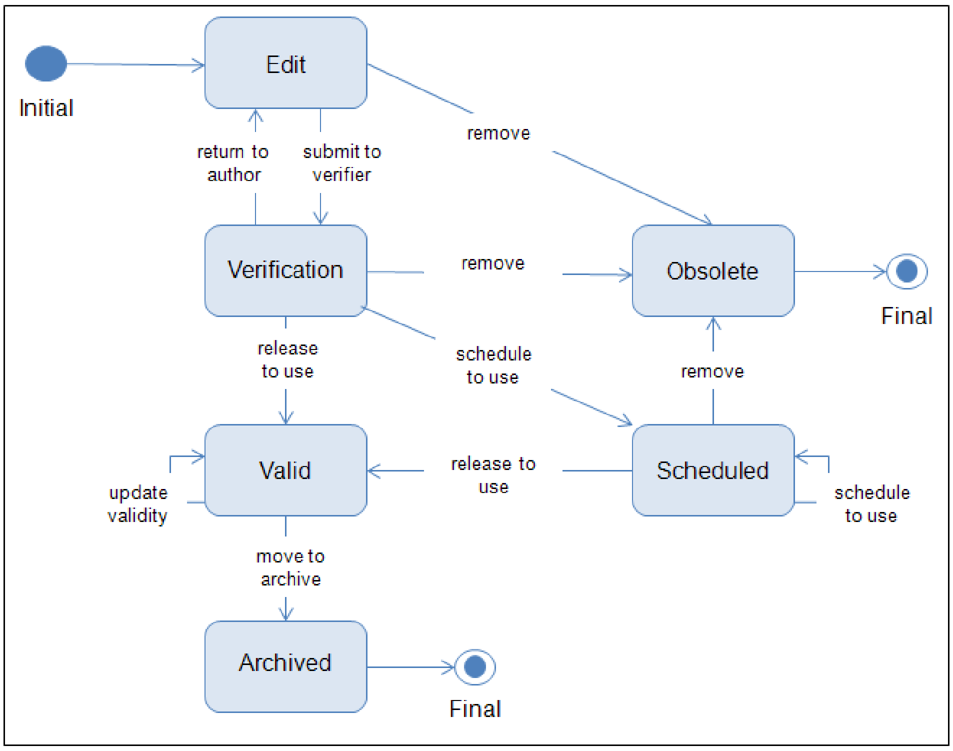


Figure 13: Simplified presentation of the master recipe FSM (MESMasterRecipeVersionGraph)

With this extension use case, the **Edit pharma** and **Edit engineering** statuses (replacing the **Edit** status), the **Ready for QA review** and **Reviewed by QA** statuses (replacing the **Verification** status), and the corresponding transitions as well as the transition instances will be added to the FSM.
Additionally, the second position of the version number will be increased when a master recipe returns from a **Verification** status and the third position of the version number will be increased when a master recipe's status is changed to **Edit engineering**.
The **Approval record** section of the Master Recipe Report will be extended by data related to the **Ready for QA review** and **Reviewed by QA** statuses.

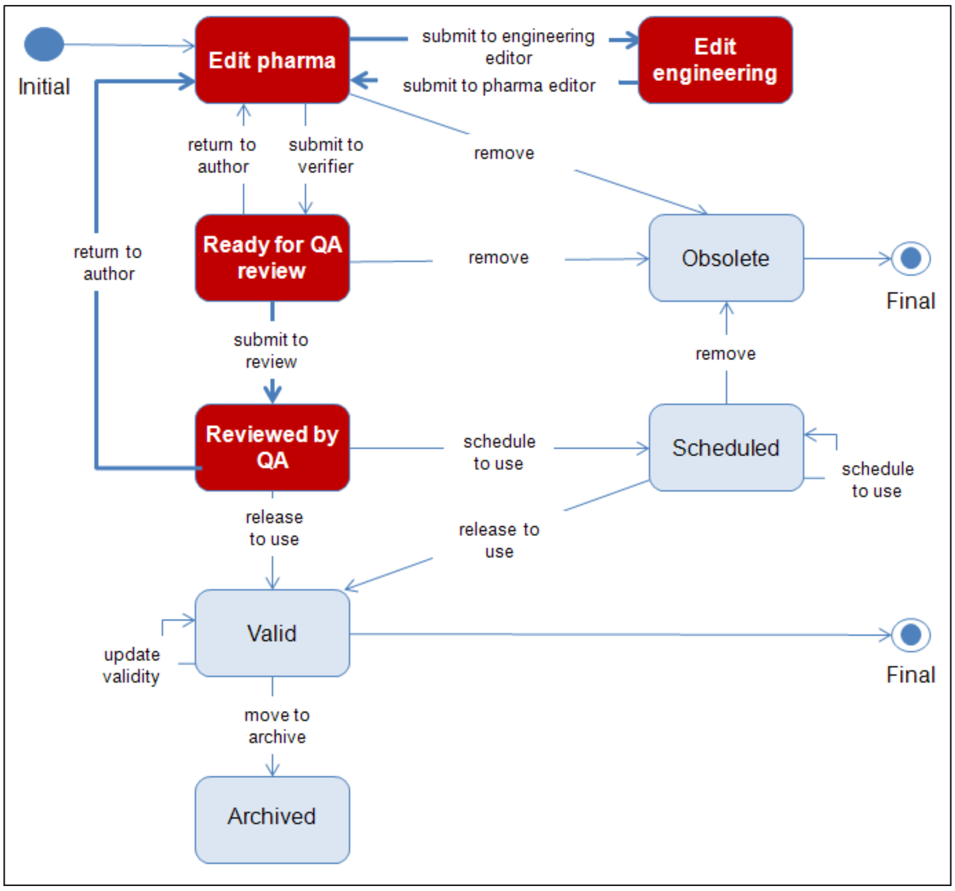


Figure 14: Simplified presentation of the adapted master recipe FSM (MYC_MESMasterRecipeVersionGraph)

The description covers the following areas:

- Create a new FSM (page 85) to support the new **Edit** and **Verification** statuses as well as the associated transitions and transition instances.
- Configure the new FSM (page 93) to be used as the standard FSM for master recipes.
- Adapt the Master Recipe Report (page 95) related to the new statuses, transitions, and transition instances.

This section assumes that you are familiar with the terms and concepts of FSMs and versioning graphs as described in "Process Designer Online Help" [B1] (page 104), "Configuring Flexible State Models" in Volume 2 of the "Technical Manual Configuration and Extension" [A1] (page 104), and "Adapting Versioning Graphs" in Volume 2 of the "Technical Manual Configuration and Extension" [A4] (page 104).

Creating an FSM with the New Edit and Verification Statuses

To create a new FSM for the **Master recipe** object, proceed as follows:

Step 1

Add messages for the new statuses (**Edit pharma**, **Edit engineering**, **Ready for QA review**, **Reviewed by QA**), transitions, and transition instances to the **MESVersioningMsgPack** message pack

- Statuses:
  - Example message ID: StateEditPharma
    - Example message: Edit pharma
  - Example message ID: StateEditEngineering
    - Example message: Edit engineering
  - Example message ID: StateVerifReadyForQAReview
    - Example message: Ready for QA review
  - Example message ID: StateVerificationReviewed
    - Example message: Reviewed by QA
- Transitions:
  - Example message ID: TransEditPharma
    - Example message: Edit pharma
  - Example message ID: TransEditEngineering
    - Example message: Edit engineering
  - Example message ID: TransVerificationReviewed
    - Example message: Ready for QA review
  - Example message ID: TransVerifReadyForQAReview
    - Example message: Reviewed by QA
- Transition instances:
  - Example message ID: TransInst_EditPharmaObsolete

    - Example message: Change to "Remove"
- Example message ID: TransInst_EditPharmaVerifReadyForReview
    - Example message: Change to "Submit to verifier"
- Example message ID: TransInst_EditPharmaEditEngineering
    - Example message: Change to "Submit to engineering editor"
- Example message ID: TransInst_EditEngineeringEditPharma
    - Example message: Change to "Submit to pharma editor"
- Example message ID: TransInst_VerifReadyForReviewEditPharma
    - Example message: Change to "Return to author"
- Example message ID: TransInst_VerifReadyForReviewObsolete
    - Example message: Change to "Remove"
- Example message ID: TransInst_VerifReadyForReviewVerificationReviewed
    - Example message: Change to "Submit to QA reviewer"
- Example message ID: TransInst_VerificationReviewedScheduled
    - Example message: Change to "Schedule to use"
- Example message ID: TransInst_VerificationReviewedValid
    - Example message: Change to "Release to use"
- Example message ID: TransInst_VerificationReviewedEditPharma
    - Example message: Change to "Return to author"

Step 2

Add messages for the new access privileges to the **MESVersioningMsgPack** message pack

- Example message ID: myc_Sig_Trans_EditPharma_Obsolete_Desc
    - Example message: Status change: Edit pharma -> Obsolete
- Example message ID: myc_Sig_Trans_EditPharma-VerifReadyForReview_Desc
    - Example message: Status change: Edit pharma -> Ready for QA review
- Example message ID: myc_Sig_Trans_EditPharma_EditEng_Desc
    - Example message: Status change: Edit pharma -> Edit engineering
- Example message ID: myc_Sig_Trans_EditEng-EditPharma_Desc
    - Example message: Status change: Edit engineering -> Edit pharma

- Example message ID: myc_Sig_Trans_VerifReadyForReview-EditPharma_Desc
    - Example message: Status change: Ready for QA review -> Edit pharma
- Example message ID: myc_Sig_Trans_VerifReadyForReview-Obsolete_Desc
    - Example message: Status change: Ready for QA review -> Obsolete
- Example message ID: myc_Sig_Trans_VerifReadyForReview-VerifReviewed_Desc
    - Example message: Status change: Ready for QA review -> Reviewed by QA
- Example message ID: myc_Sig_Trans_VerifReviewed-Scheduled_Desc
    - Example message: Status change: Reviewed by QA -> Scheduled
- Example message ID: myc_Sig_Trans_VerifReviewed-Valid_Desc
    - Example message: Status change: Reviewed by QA -> Valid
- Example message ID: myc_Sig_Trans_VerifReviewed-EditPharma_Desc
    - Example message: Status change: Reviewed by QA -> Edit pharma

Step 3

Add the access privileges for the new status transitions

- Example: myc_Vers_Trans_EditPharma-Obsolete
- Example: myc_Vers_Trans_EditPharma-VerifReadyForReview
- Example: myc_Vers_Trans_EditPharma-EditEng
- Example: myc_Vers_Trans_EditEng-EditPharma
- Example: myc_Vers_Trans_VerifReadyForReview-EditPharma
- Example: myc_Vers_Trans_VerifReadsForReview-Obsolete
- Example: myc_Vers_Trans_VerifReadsForReview-VerifReviewed
- Example: myc_Vers_Trans_VerifReviewed-Scheduled
- Example: myc_Vers_Trans_VerifReviewed-Valid
- Example: myc_Vers_Trans_VerifReviewed-EditPharma

TIP

If an access privilege is configured for a double signature, make sure to modify the **VersionHistoryEntry** Data Dictionary Class to show also the data for second signatures in the version history dialog.

Step 4

Add the new access privileges to the **versionTransitionSignature** List.

Step 5

Copy the **MESMasterRecipeVersionGraph** FSM, add your vendor code to the name, and edit the copy in the following steps.

- Example name: MYC_MESMasterRecipeVersionGraph

Step 6

Add the **Edit pharma**, **Edit engineering**, **Ready for QA review**, and **Reviewed by QA** statuses to the **MYC_MESMasterRecipeVersionGraph** FSM

- For details, see [B2] (page 104)
- Message pack: MESVersioningMsgPack
- Example properties of the new statuses:
  - Name: Edit Pharma
    - Mark with **mes.Versioning.isEditable** semantic property
    - Message ID: StateEditPharma
    - Default state
  - Name: Edit Eng.
    - Mark with **mes.Versioning.isEditable** semantic property
    - Message ID: StateEditEngineering
  - Name: Verif. ReadyForReview
    - Mark with **mes.Versioning.inVerification** semantic property
    - Message ID: StateVerifReadyForReview
  - Name: Verif. Reviewed
    - Mark with **mes.Versioning.inVerification** semantic property
    - Message ID: StateVerificationReviewed

Step 7

Add the **Edit pharma**, **Edit engineering**, **Ready for QA review**, and **Reviewed by QA** transitions to the **MYC_MESMasterRecipeVersionGraph** FSM

- For details, see [B2] (page 104)
- Message pack: MESVersioningMsgPack
- Example properties of the new transitions:
  - Name: Edit Pharma
    - Message ID: TransEditPharma
  - Name: Edit Eng.
    - Message ID: TransEditEngineering
  - Name: Verif. ReadyForReview
    - Message ID: TransVerificationReviewed
  - Name: Verification Reviewed
    - Message ID: TransVerifReadyForQAReview

Step 8

Add transition instances to and from the new statuses of the **MYC_MESMasterRecipeVersionGraph** FSM

- For details, see [B2] (page 104)
- Example transition instances:

Transition instance	From	To	Transition
1	Edit Pharma	Obsolete	Obsolete
2	Edit Pharma	Verif. ReadyForReview	Verif. ReadyForReview
3	Edit Pharma	Edit Eng.	Edit Eng.
4	Edit Eng.	Edit Pharma	Edit Pharma
5	Verif. ReadyForReview	Edit Pharma	Edit Pharma
6	Verif. ReadyForReview	Obsolete	Obsolete
7	Verif. ReadyForReview	Verif. Reviewed	Verification Reviewed
8	Verif. Reviewed	Scheduled	Scheduled
9	Verif. Reviewed	Valid	Valid
10	Verif. Reviewed	Edit Pharma	Edit Pharma

Step 9

Remove the unused transition instances from the **MYC_MESMasterRecipeVersionGraph** FSM

Transition instance	From	To	Transition
1	Edit	Obsolete	Obsolete
2	Edit	Verification	Verification
3	Verification	Edit	Edit
4	Verification	Obsolete	Obsolete
5	Verification	Scheduled	Scheduled
6	Verification	Valid	Valid

Step 10

Remove the unused statuses from the **MYC_MESMasterRecipeVersionGraph** FSM

- Edit
- Verification

Step 11

Remove the unused transitions from the **MYC_MESMasterRecipeVersionGraph** FSM

- Edit
- Verification

Step 12

Add semantic properties to the new transitions instances of the **MYC_MESMasterRecipeVersionGraph** FSM

- For details, see [A2] and [B2] (page 104)
- Example semantic properties:

Transition instance	Semantic property: mes.value.TransitionSignature	Semantic property: mes.versioning.TransitionInstanceMessageId	Additional Semantic properties
1	myc_Vers_Trans_EditPharma-Obsolete	TransInst_EditPharmaObsolete	---
2	myc_Vers_Trans_EditPharma-VerifReadyForReview	TransInst_EditPharmaVerifReadyForReview	mes.versioning.statechange.mailreceivers: MailReceiverVersionChange_MR_Edit_Verification

Transition instance	Semantic property: mes.value.TransitionSignature	Semantic property: mes.versioning.TransitionInstanceMessageId	Additional Semantic properties
3	myc_Vers_Trans_EditPharma-EditEng	TransInst_EditPharmaEditEngineering	mes.versioning.IncreaseLevel: 3
4	myc_Vers_Trans_EditEng-EditPharma	TransInst_EditEngineeringEditPharma	---
5	myc_Vers_Trans_VerifReadyForReview-EditPharma	TransInst_VerifReadyForReviewEditPharma	mes.versioning.IncreaseLevel: 2
6	myc_Vers_Trans_VerifReadyForReview-Obsolete	TransInst_VerifReadyForReviewObsolete	---
7	myc_Vers_Trans_VerifReadyForReview-VerifReviewed	TransInst_VerifReadyForReviewVerificationReviewed	---
8	myc_Vers_Trans_VerifReviewed-Scheduled	TransInst_VerificationReviewedScheduled	---
9	myc_Vers_Trans_VerifReviewed-Valid	TransInst_VerifiationReviewedValid	---
10	myc_Vers_Trans_VerifReviewed-EditPharma	TransInst_VerificationReviewedEditPharma	mes.versioning.IncreaseLevel: 2

Step 13

Add semantic properties to the new statuses of the **MYC_MESMasterRecipeVersionGraph** FSM

- For details, see [A2] and [B2] (page 104)
- Example semantic properties:
    - Status: Edit Pharma
        - sys.allow.MasterRecipe.checkin
        - sys.allow.MasterRecipe.checkout
        - sys.allow.MasterRecipe.remove
        - sys.allow.MasterRecipe.save
        - sys.allow.MasterRecipe.undoCheckout
        - sys.value.versioning.rank
        - mes.ExecutableActions.List
        - mes.WriteableAttributes.List
        - mes.versioning.Editable

- Status: Edit Eng.
  - sys.allow.MasterRecipe.checkin
  - sys.allow.MasterRecipe.checkout
  - sys.allow.MasterRecipe.remove
  - sys.allow.MasterRecipe.save
  - sys.allow.MasterRecipe.undoCheckout
  - sys.value.versioning.rank
  - mes.ExecutableActions.List
  - mes.WriteableAttributes.List
  - mes.versioning.Editable
- Status: Verif. ReadyForReview
  - sys.value.versioning.rank
  - mes.ExecutableActions.List
  - mes.WriteableAttributes.List
  - mes.versioning.InVerification
- Status: Verif. Reviewed
  - sys.value.versioning.rank
  - mes.ExecutableActions.List
  - mes.WriteableAttributes.List
  - mes.versioning.InVerification

Step 14

Override the default value of the following semantic properties of the new statuses of the **MYC_MESMasterRecipeVersionGraph** FSM

- For details, see [A2] and [B3] (page 104)
- Status: Edit Pharma, Edit Eng.
  - Semantic property: sys.value.versioning.rank
    - Example value: 10
  - Semantic property: mes.ExecutableActions.List
    - Example value: VP_Ops_Edit_MasterRecipe
  - Semantic property: mes.WriteableAttributes.List
    - Example value: VP_Atr_Edit_MasterRecipe

- Semantic property: mes.versioning.Editable
  - Example value: true
- Status: Verif. ReadyForReview, Verif. Reviewed
  - Semantic property: sys.value.versioning.rank
    - Example value: 50
  - Semantic property: mes.ExecutableActions.List
    - Example value: VP_Ops_Verification_MasterRecipe
  - Semantic property: mes.WriteableAttributes.List
    - Example value: VP_Atr_Verification_MasterRecipe

Step 15

Add the following statuses to the **StateFilter** of the **X_RefSelMasterRecipeFilterSimulation** filter

- Verif. ReadyForReview
- Verif. Reviewed

TIP

The filter is only necessary and visible if the **LibraryHolder/services-order-ifc.jar/allowNonValidMasterRecipeForOrder** configuration key is enabled.
If your database does not contain master recipes, you can remove the **Verification** status from the filter.

Configuring the New FSM As Standard FSM for Master Recipes

To configure the **MYC_MESMasterRecipeVersionGraph** FSM as the standard FSM for **Master recipe** objects, proceed as follows:

Step 1

Add the **MYC_MESMasterRecipeVersionGraph** FSM to the **MasterRecipe** FSM configuration

- For details, see [B4] (page 104)
- Add the **MYC_MESMasterRecipeVersionGraph** FSM with **vpversion** as **Relationship Name**.
- Set the **Relationship Type** to **Manual**.

Step 2

Configure the **MYC_MESMasterRecipeVersionGraph** FSM to be used as versioning FSM for batch-specific master recipes

- For details, see [A6] (page 104)
- Set the **LibraryHolder/services-s88-ifc.jar/BatchRecipeVersioningFSMName** configuration key to **MYC_MESMasterRecipeVersionGraph**.

TIP

If your database does not contain master recipes yet, you can remove the relationship with the **MESMasterRecipeVersionGraph** FSM and the **MESMasterRecipeVersionGraph** FSM itself.

Adapting the Master Recipe Report

To adapt the **Approval record** section of the Master Recipe Report to consider the version transitions for the new FSM, proceed as follows:

- For details, see [A3] (page 104)

Step 1

Create the *MYCIndividualApprovalConverter* class with *com.rockwell.mes.services.batchrecord.impl.converter.IndividualApprovalConverter* as superclass and implementing the *com.rockwell.mes.services.batchrecord.impl.converter.IIndividualApprovalConverte* standard interface.

1. Add private constants for the new status names
   - STATE_NAME_EDITPHARMA
   - STATE_NAME_VERIFREADYFORREVIEW
   - STATE_NAME_VERIFICATIONREVIEWED
2. Add public IDs for the verification transitions
   - APPROVAL_TRANSITION_VERIFREADYFORREVIEW
   - APPROVAL_TRANSITION_VERIFICATIONREVIEWED

```
package com.rockwell.custmes.services.batchrecord.impl;
import org.mesa.xml.b2MMLV0600.IndividualApprovalType;
import com.rockwell.mes.services.batchrecord.impl.converter.
       IndividualApprovalConverter;
/**
 * Customized data converter to convert a status transition history entry into
 * {@link IndividualApprovalType}.
 */
public class MYCIndividualApprovalConverter extends IndividualApprovalConverter {
  /** Name of the state edit - pharma */
  private static final String STATE_NAME_EDITPHARMA = "Edit Pharma";
  /** Name of the state verification - ready for review */
  private static final String STATE_NAME_VERIFREADYFORREVIEW = "Verif. ReadyForReview";
  /** Name of the state verification - reviewed */
  private static final String STATE_NAME_VERIFICATIONREVIEWED = "Verif. Reviewed";
  /** information type to be filled for Approval record: transition information
   *  for Action 'Submitted' */
  public static final int APPROVAL_TRANSITION_VERIFREADYFORREVIEW = 50;
  /** information type to be filled for Approval record: transition information
   *  for Action 'Reviewed by' */
  public static final int APPROVAL_TRANSITION_VERIFICATIONREVIEWED = 51;
}
```

Step 2

Modify the implementation of your *MYCIndividualApprovalConverter*

- Overwrite the *getTransitionStepValue* method in order to add a unique number to each individual approval (representing a status transition).

```
package com.rockwell.custmes.services.batchrecord.impl;
[...]
/**
 * Customized data converter to convert a status transition history entry into
 * {@link IndividualApprovalType}.
 */
public class MYCIndividualApprovalConverter extends IndividualApprovalConverter {
  [...]
  @Override
  public Integer getTransitionStepValue(IndividualApprovalType appInd, String
fromStatus,
         String toStatus) {
    final Integer returnValue;
    if (isInitialTransition(fromStatus, toStatus)) {
      returnValue = StatusTransitionStep.INITIAL_TRANSITION.getStepValue();
    } else if (isReadyForReviewTransition(fromStatus, toStatus)) {
      returnValue = APPROVAL_TRANSITION_VERIFREADYFORREVIEW;
    } else if (isReviewedTransition(fromStatus, toStatus)) {
      returnValue = APPROVAL_TRANSITION_VERIFICATIONREVIEWED;
    } else if (isValidateTransition(fromStatus, toStatus)) {
      returnValue = StatusTransitionStep.VALIDATE_TRANSITION.getStepValue();
    } else {
      returnValue = null;
    }
    return returnValue;
  }
  // overridden for approval for initial creation (changed FSM)
  @Override
  protected boolean isInitialTransition(final String fromStatus, final String toStatus)
{
    return StringUtils.isBlank(fromStatus) && STATE_NAME_EDITPHARMA.equals(toStatus);
  }
  // approval for latest state change to verification - ready for review (changed FSM)
  private boolean isReadyForReviewTransition(String fromStatus, String toStatus) {
    return STATE_NAME_EDITPHARMA.equals(fromStatus) &&
           STATE_NAME_VERIFREADYFORREVIEW.equals(toStatus);
  }
  // approval for latest state change to verification - reviewed (changed FSM)
  private boolean isReviewedTransition(String fromStatus, String toStatus) {
    return STATE_NAME_VERIFREADYFORREVIEW.equals(fromStatus) &&
           STATE_NAME_VERIFICATIONREVIEWED.equals(toStatus);
  }
  // approval for initial state change to valid (changed FSM),
  // ignore transitions from valid to valid
  @Override
  protected boolean isValidateTransition(String fromStatus, String toStatus) {
    if (!toStatus.equals(IVersioningService.STATE_NAME_VALID)) {
      return false;
    }
    return fromStatus.equals(STATE_NAME_VERIFICATIONREVIEWED) //
        || fromStatus.equals(IVersioningService.STATE_NAME_SCHEDULED);
  }
}
```

```
package com.rockwell.custmes.services.batchrecord.impl;
import org.mesa.xml.b2MMLV0600.IndividualApprovalType;
import com.rockwell.mes.services.batchrecord.impl.converter.
       IndividualApprovalConverter;
/**
 * Customized data converter to convert a status transition history entry into
 * {@link IndividualApprovalType}.
 */
public class MYCIndividualApprovalConverter extends IndividualApprovalConverter {
  /** Name of the state edit - pharma */
  private static final String STATE_NAME_EDITPHARMA = "Edit Pharma";
  /** Name of the state verification - ready for review */
  private static final String STATE_NAME_VERIFREADYFORREVIEW = "Verif. ReadyForReview";
  /** Name of the state verification - reviewed */
  private static final String STATE_NAME_VERIFICATIONREVIEWED = "Verif. Reviewed";
  /** information type to be filled for Approval record: transition information
   *  for Action 'Submitted' */
  public static final int APPROVAL_TRANSITION_VERIFREADYFORREVIEW = 50;
  /** information type to be filled for Approval record: transition information
   *  for Action 'Reviewed by' */
  public static final int APPROVAL_TRANSITION_VERIFICATIONREVIEWED = 51;
}
```

```
package com.rockwell.custmes.services.batchrecord.impl;
[...]
/**
 * Customized data converter to convert a status transition history entry into
 * {@link IndividualApprovalType}.
 */
public class MYCIndividualApprovalConverter extends IndividualApprovalConverter {
  [...]
  @Override
  public Integer getTransitionStepValue(IndividualApprovalType appInd, String
fromStatus,
         String toStatus) {
    final Integer returnValue;
    if (isInitialTransition(fromStatus, toStatus)) {
      returnValue = StatusTransitionStep.INITIAL_TRANSITION.getStepValue();
    } else if (isReadyForReviewTransition(fromStatus, toStatus)) {
      returnValue = APPROVAL_TRANSITION_VERIFREADYFORREVIEW;
    } else if (isReviewedTransition(fromStatus, toStatus)) {
      returnValue = APPROVAL_TRANSITION_VERIFICATIONREVIEWED;
    } else if (isValidateTransition(fromStatus, toStatus)) {
      returnValue = StatusTransitionStep.VALIDATE_TRANSITION.getStepValue();
    } else {
      returnValue = null;
    }
    return returnValue;
  }
  // overridden for approval for initial creation (changed FSM)
  @Override
  protected boolean isInitialTransition(final String fromStatus, final String toStatus)
{
    return StringUtils.isBlank(fromStatus) && STATE_NAME_EDITPHARMA.equals(toStatus);
  }
  // approval for latest state change to verification - ready for review (changed FSM)
  private boolean isReadyForReviewTransition(String fromStatus, String toStatus) {
    return STATE_NAME_EDITPHARMA.equals(fromStatus) &&
           STATE_NAME_VERIFREADYFORREVIEW.equals(toStatus);
  }
  // approval for latest state change to verification - reviewed (changed FSM)
  private boolean isReviewedTransition(String fromStatus, String toStatus) {
    return STATE_NAME_VERIFREADYFORREVIEW.equals(fromStatus) &&
           STATE_NAME_VERIFICATIONREVIEWED.equals(toStatus);
  }
  // approval for initial state change to valid (changed FSM),
  // ignore transitions from valid to valid
  @Override
  protected boolean isValidateTransition(String fromStatus, String toStatus) {
    if (!toStatus.equals(IVersioningService.STATE_NAME_VALID)) {
      return false;
    }
    return fromStatus.equals(STATE_NAME_VERIFICATIONREVIEWED) //
        || fromStatus.equals(IVersioningService.STATE_NAME_SCHEDULED);
  }
}
```

The idea is to add a unique step number to each individual approval. This number can be used later in the batch report to access the required approval applying this unique number.

Step 3

Optional: If you prefer to use number literals instead of JAVA constants in the report-design (50 and 51 in the example code if Step 1), you can omit this step.
Create the *MYCMasterRecipeDocumentWrapper* class with *com.rockwell.mes.services.batchreport.impl.MasterRecipeDocumentWrapper* as superclass.

- Add public IDs for the verification transitions
  - APPROVAL_TRANSITION_VERIFREADYFORREVIEW
  - APPROVAL_TRANSITION_VERIFICATIONREVIEWED

```
package com.rockwell.custmes.services.batchreport.impl;
import com.rockwell.custmes.services.batchrecord.impl.MYCIndividualApprovalConverter;
import com.rockwell.mes.services.batchreport.impl.MasterRecipeDocumentWrapper;
/**
 * Custom Wrapper for a B2MML MasterRecipeDocument. Provides the getters used by
 * Jasper reports.
 */
public class MYCMasterRecipeDocumentWrapper extends MasterRecipeDocumentWrapper {
  /**
   * information type to be filled for Approval record: transition information
   * for Action 'Submitted'
  */
  public static final int APPROVAL_TRANSITION_VERIFREADYFORREVIEW =
         MYCIndividualApprovalConverter.APPROVAL_TRANSITION_VERIFREADYFORREVIEW;
  /**
   * information type to be filled for Approval record: transition information
   * for Action 'Reviewed by'
  */
  public static final int APPROVAL_TRANSITION_VERIFICATIONREVIEWED =
         MYCIndividualApprovalConverter.APPROVAL_TRANSITION_VERIFREADYFORREVIEW;
}

package com.rockwell.custmes.services.batchrecord.impl;
import org.mesa.xml.b2MMLV0600.IndividualApprovalType;
import com.rockwell.mes.services.batchrecord.impl.converter.
       IndividualApprovalConverter;
/**
```

```
 * Customized data converter to convert a status transition history entry into
 * {@link IndividualApprovalType}.
 */
public class MYCIndividualApprovalConverter extends IndividualApprovalConverter {
  /** Name of the state edit - pharma */
  private static final String STATE_NAME_EDITPHARMA = "Edit Pharma";
  /** Name of the state verification - ready for review */
  private static final String STATE_NAME_VERIFREADYFORREVIEW = "Verif. ReadyForReview";
  /** Name of the state verification - reviewed */
  private static final String STATE_NAME_VERIFICATIONREVIEWED = "Verif. Reviewed";
  /** information type to be filled for Approval record: transition information
   *  for Action 'Submitted' */
  public static final int APPROVAL_TRANSITION_VERIFREADYFORREVIEW = 50;
  /** information type to be filled for Approval record: transition information
   *  for Action 'Reviewed by' */
  public static final int APPROVAL_TRANSITION_VERIFICATIONREVIEWED = 51;
}

package com.rockwell.custmes.services.batchreport.impl;
import com.rockwell.custmes.services.batchrecord.impl.MYCIndividualApprovalConverter;
import com.rockwell.mes.services.batchreport.impl.MasterRecipeDocumentWrapper;
/**
 * Custom Wrapper for a B2MML MasterRecipeDocument. Provides the getters used by
 * Jasper reports.
 */
public class MYCMasterRecipeDocumentWrapper extends MasterRecipeDocumentWrapper {
  /**
   * information type to be filled for Approval record: transition information
   * for Action 'Submitted'
  */
  public static final int APPROVAL_TRANSITION_VERIFREADYFORREVIEW =
         MYCIndividualApprovalConverter.APPROVAL_TRANSITION_VERIFREADYFORREVIEW;
  /**
   * information type to be filled for Approval record: transition information
   * for Action 'Reviewed by'
  */
  public static final int APPROVAL_TRANSITION_VERIFICATIONREVIEWED =
         MYCIndividualApprovalConverter.APPROVAL_TRANSITION_VERIFREADYFORREVIEW;
}
```

Step 4

Configure PharmaSuite to use your *MYCMasterRecipeDocumentWrapper* class to generate the Master Recipe Report and the *MYCIndividualApprovalConverter* class to convert the master recipe status transitions into individual approval records.

- For details, see [A5] (page 104)

1. Create a copy of *services-batchreport.xml* (e.g. *myc-batchreport.xml*) and replace the implementation class for the *MasterRecipeDocumentWrapper* class with a reference to your *MYCMasterRecipeDocumentWrapper* class.
2. Create a copy of *services-batchrecord.xml* (e.g. *myc-batchrecord.xml*) and replace the implementation class for the *IndividualApprovalConverter* class with a reference to your *MYCIndividualApprovalConverter* class.
3. Create a copy of *config.xml* (e.g. *myc-config.xml*) and replace the reference to *services-batchreport.xml* and *services-batchrecord.xml* with your copy of the service configuration XML file. Make sure that your application configuration

references your configuration file in *Libraries/commons-base-ifc.jar/ServicesConfigFile*.

Step 5

Add messages for the new actions and roles to the **PS-BatchReport** message pack

- Example message ID: SubmittedBy_Label
    - Example message: Submitted:
- Example message ID: RoleRecipeEditingResp_Label
    - Example message: Recipe Author
- Example message ID: ReviewedBy_Label
    - Example message: Reviewed by:
- Example message ID: RoleQAResp_Label
    - Example message: Responsible QA person

Step 6

Modify the Master Recipe Report with Jaspersoft Studio

- For details, see [A3] (page 104)
- Open the **PS-MRReport-Main** report in Jaspersoft Studio

1. Change band height of Detail 6 (contains the **Approval record** section) from 139 to 323
2. Add the following variables:
    - Name: approvalSubmitFirstUserInfo
        - Variable Class: java.lang.String
        - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFREADYFORREVIEW,$P{REPORT_RESOURCE_BUNDLE})
    - Name: approvalSubmitFirstValidDate
        - Variable Class: java.util.Date
        - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordDateData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFREADYFORREVIEW)

- Name: approvalSubmitSecondUserInfo
  - Variable Class: java.lang.String
  - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFREADYFORREVIEW,$P{REPORT_RESOURCE_BUNDLE}, true)
- Name: approvalSubmitSecondValidDate
  - Variable Class: java.util.Date
  - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordDateData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFREADYFORREVIEW, true)
- Name: approvalReviewFirstUserInfo
  - Variable Class: java.lang.String
  - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFICATIONREVIEWED,$P{REPORT_RESOURCE_BUNDLE})
- Name: approvalReviewFirstValidDate
  - Variable Class: java.util.Date
  - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordDateData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFICATIONREVIEWED)
- Name: approvalReviewSecondUserInfo
  - Variable Class: java.lang.String
  - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFICATIONREVIEWED,$P{REPORT_RESOURCE_BUNDLE}, true)

- Name: approvalReviewSecondValidDate
  - Variable Class: java.util.Date
  - Variable Expression: $P{MASTER_RECIPE_DOCUMENT_WRAPPER}.getApprovalRecordDateData($P{MASTER_RECIPE_DOCUMENT_WRAPPER}.APPROVAL_TRANSITION_VERIFICATIONREVIEWED, true)

3. Copy the six fields for **Approval** (starting with "$R{ApprovedBy_Label} through to $V{approvalSecondValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalSecondValidDate}")
4. Insert the copied fields twice, move them below the original fields, and move the separator line to the end.
5. Modify the first two instances (original and first copy):

From	To
$R{ApprovedBy_Label}	$R{SubmittedBy_Label}
$R{RoleManufactResp_Label}	$R{RoleRecipeEditingResp_Label}
$V{approvalFirstUserInfo}	$V{approvalSubmitFirstUserInfo}
$V{approvalFirstValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalFirstValidDate})	$V{approvalSubmitFirstValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalSubmitFirstValidDate})
$V{approvalSecondUserInfo}	$V{approvalSubmitSecondUserInfo}
$V{approvalSecondValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalSecondValidDate})	$V{approvalSubmitSecondValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalSubmitSecondValidDate})
$R{ApprovedBy_Label}	$R{ReviewedBy_Label}
$R{RoleManufactResp_Label}	$R{RoleQAResp_Label}
$V{approvalFirstUserInfo}	$V{approvalReviewFirstUserInfo}
$V{approvalFirstValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalFirstValidDate})	$V{approvalReviewFirstValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalReviewFirstValidDate})
$V{approvalSecondUserInfo}	$V{approvalReviewSecondUserInfo}
$V{approvalSecondValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalSecondValidDate})	$V{approvalReviewSecondValidDate} == null ? null : msg("{0, date, " + $P{DATE_FORMAT_FULL} + "}",$V{approvalReviewSecondValidDate})

Step 7

Add the modified **PS-MRReport-Main** report to PharmaSuite

- For details, see [A3] (page 104)
- Import the changed report by means of the **mes_PS-BatchReportManager** form

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Flexible State Model	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Configuring Flexible State Models	PSCEV2-GR010B-EN-E
A2	■ Semantic properties	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Configuring Flexible State Models ■ Configuring PharmaSuite with Flexible State Models	PSCEV2-GR010B-EN-E
A3	■ Master Recipe Report	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Master Recipe Reports in PharmaSuite ■ Adapting the Data Retrieval Related to the Main Report or a Sub-report	PSCEV2-GR010B-EN-E
A4	■ Versioning graphs	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting Versioning Graphs	PSCEV2-GR010B-EN-E
A5	■ Make use of MYCMasterRecipeDocumentWrapper class	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Managing Services ■ Configuring Service Implementations	PSCEV2-GR010B-EN-E
A6	■ Versioning FSM for master recipes	PharmaSuite Technical Manual Configuration & Extension - Volume 4: Configuration Keys of PharmaSuite ■ Configuration Keys for Recipe, Order, and Workflow Management	PSCEV4-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

The following documents provide more details relevant to the implementation of the extension use case and are distributed with the FactoryTalk ProductionCentre installation.

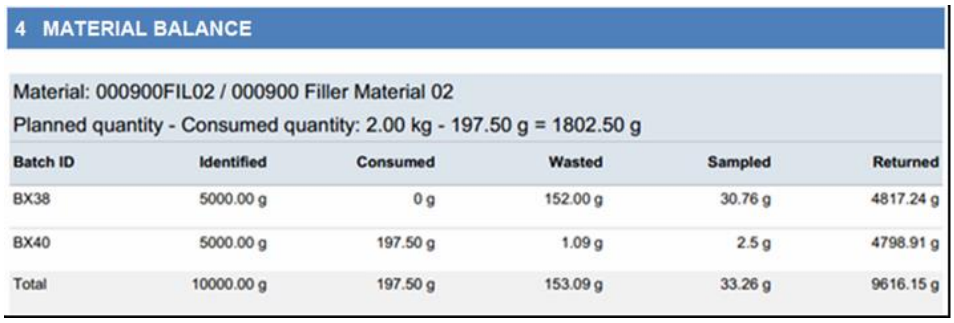
No.	Keyword	Document Title	Part Number
B1	■ Flexible State Model	Process Designer Online Help: ■ Flexible State Models	N/A
B2	■ Add status to FSM ■ Add transition to FSM ■ Add transition instances to status ■ Add semantic properties to transition instance ■ Add semantic properties to status	Process Designer Online Help: Flexible State Models ■ Creating FSM States, Transitions, and Transition Instances	N/A
B3	■ Overwrite value of semantic property	Process Designer Online Help: Flexible State Models ■ Configuring Additional FSM Features	N/A
B4	■ Relationship type and FSM configuration	Process Designer Online Help: ■ FSM Tab Interface	N/A

TIP

To access the "Process Designer Online Help", use the following syntax: *http://<MES-PS-HOST>:8080/PlantOperations/docs/help/pd/index.htm*, where <MES-PS-HOST> is the name of your PharmaSuite server. To view the online help, the Apache Tomcat of the FactoryTalk ProductionCentre installation must be running.

Adding the Material Balance Section to the Batch Report

This section describes how to add a new section to display material balance information to the batch report. The material balance is calculated from the identified materials and their accounting-related data collected through the processing of an order.



4 MATERIAL BALANCE

Material: 000900FIL02 / 000900 Filler Material 02
Planned quantity - Consumed quantity: 2.00 kg - 197.50 g = 1802.50 g

Batch ID	Identified	Consumed	Wasted	Sampled	Returned
BX38	5000.00 g	0 g	152.00 g	30.76 g	4817.24 g
BX40	5000.00 g	197.50 g	1.09 g	2.5 g	4798.91 g
Total	10000.00 g	197.50 g	153.09 g	33.26 g	9616.15 g

Figure 15: Material Balance section with one material

The extension use case requires a fully set up PharmaSuite development environment and Java know-how.

The description covers the following areas:

- Extend the batch record schema definition with new types (page 108)
- Implement a converter to convert a list of OSIs into a list of **MaterialBalanceItemType** instances (B2MML type of the data source) (page 111)
- Implement a collector to get the list of **MaterialBalanceItemType** instances (page 123)
- Extend the aggregator classes for the generation of batch reports and phase reports (page 123)
- Create a customer extension of the IBatchProductionRecordDocumentWrapperExtension (page 125) interface
- Create fast bean reader of the IFastBeanPropertyAccessor (page 125) interface
- Configure the affected services (page 132)
- Adapt the report design of the batch report to display the **Material Balance** section (page 133)

Extending the Batch Record Schema

To extend the batch record schema with new types, modify the batch production record schema file. From the *c:\Users\<USERNAME>\.FTPC\<MES-PS-HOST>\ProductionCentre\jars\ShopOps\ftps-b2mml-v0600-<PharmaSuite version>.jar* archive,in the *schemaorg_apache_xmlbeans\src* subfolder, extract the *BatchML-V0600-BatchProductionRecordExtensions.xsd* file and copy it to the *services\b2mml\src\main\xsd* folder. Then proceed as follows:

- For details, see [A1] (page 140)

TIP

The code of the following code snippets may have errors or may be deprecated. Please investigate files of the PharmaSuite SDK.

Step 1

Import all needed external types into *BatchML-V0600-BatchProductionRecordExtensions.xsd*

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.mesa.org/xml/B2MML-V0600-AllExtensions"
    targetNamespace="http://www.mesa.org/xml/ B2MML-V0600-AllExtensions"
    xmlns:b2mml="http://www.mesa.org/xml/B2MML-V0600"
    elementFormDefault="qualified"
    attributeFormDefault = "unqualified">
<!-- Import the Common schema in the 'b2mml' name space.
     This also brings in the Core Components under the 'b2mml' name space.
-->
<xsd:import namespace="http://www.mesa.org/xml/B2MML-V0600"
        schemaLocation="B2MML-V0600-Common.xsd"/>
```

Step 2

Add the new **MaterialBalance** element in the **CustomerExtendedBatchProductionRecord** group in *BatchML-V0600-BatchProductionRecordExtensions.xsd*

```
<xsd:group name = "CustomerExtendedBatchProductionRecord">
    <xsd:sequence>
        <!-- add extended elements here -->
        <xsd:element name="MaterialBalance" type="MaterialBalanceType"
            minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
</xsd:group>
```

Step 3

Add new types in *BatchML-V0600-BatchProductionRecordExtensions.xsd*

1. **MaterialBalanceType** represents the whole data of the material balance and is referenced by the **MaterialBalance** element. It contains the following elements:
   - MaterialBalanceItems
2. **MaterialBalanceItemType** represents the material balance data of one material and is referenced by the **MaterialBalanceItems** element of **MaterialBalanceType**. It contains the following elements:
   - Material
   - PlannedQuantity
   - DiffPlannedConsumedQuantity
   - MaterialBalanceItemDetails
   - MaterialAccountingTotal
3. **MaterialBalanceItemDetailsType** represents the material balance data of one batch of one material and is referenced by the **MaterialBalanceItemDetails** element of **MaterialBalanceItemType**. It contains the following elements:
   - Batch
   - MaterialAccountingDetails

4. **MaterialAccountingDetailsType** contains the different quantities of the material balance and is referenced by the **MaterialAccountingDetails** element of **MaterialBalanceItemDetailsType**. It contains the following elements:
   - IdentifiedQuantity
   - ConsumedQuantity
   - WastedQuantity
   - SampledQuantity
   - ReturnedQuantity

```
<xsd:complexType name="MaterialBalanceType">
  <xsd:sequence>
    <xsd:element name="MaterialBalanceItems" type="MaterialBalanceItemType"
                 minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MaterialBalanceItemType">
  <xsd:sequence>
    <xsd:element name="Material" type="MaterialType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="PlannedQuantity" type="b2mml:QuantityValueType" minOccurs="1"
                 maxOccurs="1"/>
    <xsd:element name="DiffPlannedConsumedQuantity" type="b2mml:QuantityValueType"
                 minOccurs="0" maxOccurs="1" nillable="true" />
    <xsd:element name="MaterialBalanceItemDetails"
                 type="MaterialBalanceItemDetailsType"
                 minOccurs="0" maxOccurs="unbounded" nillable="true" />
    <xsd:element name="MaterialAccountingTotal" type="MaterialAccountingDetailsType"
                 minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MaterialBalanceItemDetailsType">
  <xsd:sequence>
    <xsd:element name="Batch" type="BatchType" minOccurs="1" maxOccurs="1" />
    <xsd:element name="MaterialAccountingDetails"
                 type="MaterialAccountingDetailsType"
                 minOccurs="1" maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MaterialAccountingDetailsType">
  <xsd:sequence>
    <xsd:element name="IdentifiedQuantity" type="b2mml:QuantityValueType" minOccurs="0"
                 maxOccurs="1" nillable="true" />
    <xsd:element name="ConsumedQuantity" type="b2mml:QuantityValueType" minOccurs="0"
                 maxOccurs="1" nillable="true" />
    <xsd:element name="WastedQuantity" type="b2mml:QuantityValueType" minOccurs="0"
                 maxOccurs="1" nillable="true" />
    <xsd:element name="SampledQuantity" type="b2mml:QuantityValueType" minOccurs="0"
                 maxOccurs="1" nillable="true" />
    <xsd:element name="ReturnedQuantity" type="b2mml:QuantityValueType" minOccurs="0"
                 maxOccurs="1" nillable="true" />
  </xsd:sequence>
</xsd:complexType>
```

Step 4

Generate Java interfaces and bean classes from the type definitions in the XSD files

1. Extract the XSD files whose names start with *B2MML-*, *BatchML-*, and *RA* from the *ftps-b2mml-v0600.jar* library. Copy the XSD files (except *BatchML-V0600-BatchProductionRecordExtensions.xsd*) to the *services/batchrecord/config* folder.
2. Execute the *generate_b2mml_library* ant task (located under *FTPS_SDK/service/batchrecord*) to generate the new library.
   - The *ftps-b2mml-v0600-<PharmaSuite version>.jar* library is generated under *FTPS_SDK\lib\com.rockwell\*.
3. Update the *ftps-b2mml-v0600-<PharmaSuite version>.jar* library in your project.

Implementing the MaterialBalanceConverter Converter

A converter is a service used to convert PharmaSuite data into B2MML types. The *MaterialBalanceConverter* service converts:

- a list of **OrderStepInput** objects into a list of **MasterialBalanceItemType** instances and
- 5 measured values into a **MaterialAccountingDetailsType** instance

To implement the *MaterialBalanceConverter* converter, proceed as follows:

Step 1

Create a helper class containing the following methods:

```
public static MeasuredValue getIdentifiedSublotQuantity(OrderStepInput osi)

public static List<OrderStepInput> getOSIsFromBatch(ProcessOrderItem order, Batch
batch, boolean onlyInputs)

public static MeasuredValue[] getMaterialBalanceQuantitiesOfBatch(ProcessOrderItem
order, Batch batch)

private static MeasuredValue[] getAccountedQuantitiesFromOSI(OrderStepInput osi)
```

Example code

```
/**
 * A helper class providing methods for Material Balance use case
 */
public class MaterialBalanceHelper {
  /** Column name of Transaction History containing the order name */
  private static final String TH_COLUMN_NAME_ORDER = "X_order";
  /** Column name of Transaction History containing the sublot identifier */
  private static final String TH_COLUMN_NAME_SUBLOT = "X_sublotIdentifierNew";
  /** Column name of Transaction History containing the creation time */
  private static final String TH_COLUMN_NAME_TIMESTAMP = "X_timestamp";
  /** Used to compare boolean value TRUE in long */
  private static final Long LONG_TRUE = Long.valueOf(1);
  /**
   * Service used to get the master OSI from an OSI, and to check if an OSI is
   * replaced by an alternative
   */
  private static IWDOrderStepInputService osiService =
                 ServiceFactory.getService(IWDOrderStepInputService.class);
  /**
   * Service used to get identified OSI Sublots of an OSI
   */
  private static IOrderStepExecutionService osService =
                 ServiceFactory.getService(IOrderStepExecutionService.class);
  /**
   * Get the sublot quantity identified to process an order step input. If the
   * same sublot has been identified more than one time in a same order, the
   * quantity returned is the sublot quantity when it was identified the first
   * time in this order context.
   *
   * @param osi the OrderStepInput
   * @return the identified quantity of the attached sublot if exists,
   *         otherwise null
   * @throws DatasweepException if an error occurred when executing the filter
   */
  public static MeasuredValue getIdentifiedSublotQuantity(OrderStepInput osi) throws
                             DatasweepException {
    TransactionHistoryFilter filterFactory = new
                TransactionHistoryFilter(PCContext.getServerImpl(),
                TransactionHistoryObject.TABLE_NAME);
    List<TransactionHistoryObject> txObjects;
    // Create filter to get sublot identification data
    TransactionHistoryFilter filter = filterFactory
                .forTransactionTypeEqualTo(TransactionType.CHANGE_OF_SUBLOT_DATA);
    filter.forTransactionSubtypeEqualTo(TransactionSubtype.IDENTIFICATION);
    // Get sublot identification in our order
    filter.forColumnNameEqualTo(TH_COLUMN_NAME_ORDER,
           osi.getOrderStep().getControlRecipe().getProcessOrderItem().getName());
    // Get identification of our sublot
    filter.forBatchIdentifierNewEqualTo(osi.getAttachSublot().getBatchName());
    filter.forColumnNameEqualTo(TH_COLUMN_NAME_SUBLOT,
           osi.getAttachSublot().getName());
    // Order the result to get the oldest identification first
    filter.addOrderATColumnBy(TH_COLUMN_NAME_TIMESTAMP, IFilterSortOrders.ASCENDING);
    // Get only the oldest identification
    filter.setMaxRows(1);
    txObjects = TransactionHistoryFilter.getFilteredTransactionHistoryObjects(filter);
    if (txObjects.isEmpty()) {
      return null;
    } else {
      return txObjects.get(0).getQuantityNew();
    }
  }
  /**
   * Returns all OSIs (only inputs if onlyInputs is true) for
   * which a given batch has been identified.
   *
   * @param order the order for which the batch has been identified
   * @param batch the identified batch
   * @param onlyInputs if the list should contain only input OSIs (no
   *            transfer)
   * @return the list of OrderStepInput containing master and split OSIs
   */
  public static List<OrderStepInput> getOSIsFromBatch(ProcessOrderItem order,
         Batch batch, boolean onlyInputs) {
    Vector<OrderStep> osos = ((ControlRecipe) order.getRuntimeRecipe(0))
                    .getOrderSteps();
    List<OrderStepInput> osisForBatch = new ArrayList<OrderStepInput>();
    for (OrderStep os : osos) {
      Vector<OrderStepInput> osis = os.getOrderStepInputItems();
      for (OrderStepInput osi : osis) {
        // Do not return replaced by alternative and transfers (if onlyInputs is true)
        if (osiService.isReplacedByAlternative(osi)
            || (onlyInputs && osi.getProcessingType() !=
                IOrderStepInputTypes.INPUT_TYPE_INPUT)) {
            continue;
        }
        if (osi.getAttachSublot() != null && batch.equals(osi.getAttachSublot()
            .getBatch())) {
          osisForBatch.add(osi);
        }
      }
    }
    return osisForBatch;
  }
  /**
   * Indexes of quantities in the array returned by following methods
   */
  public static final int INDEX_PLANNED_QUANTITY = 0, INDEX_IDENTIFIED_QUANTITY = 1,
         INDEX_CONSUMED_QUANTITY = 2, INDEX_WASTED_QUANTITY = 3,
         INDEX_SAMPLED_QUANTITY = 4, INDEX_RETURNED_QUANTITY = 5;
  /**
   * Get the quantities of the Material Balance for a given batch.
   *
   * @param order the order for which the batch has been identified
   * @param batch the batch identified
   * @return an array of MeasuredValue containing at indexes:
   *         INDEX_PLANNED_QUANTITY: the planned quantity. Sum of planned
   *         quantities of BOM positions for which the batch is identified.
   *         INDEX_IDENTIFIED_QUANTITY: Sum of identified quantities of
   *         sublots belonging to the batch identified in the order.
   *         INDEX_CONSUMED_QUANTITY: Sum of consumed quantities
   *         of this batch in the order.
   *         INDEX_WASTED_QUANTITY: Sum of wasted quantities of
   *         this batch in the order.
   *         INDEX_SAMPLED_QUANTITY: Sum of sampled quantities of
   *         this batch in the order.
   *         INDEX_RETURNED_QUANTITY: Sum of returned quantities
   *         of this batch in the order.
   * @throws MESIncompatibleUoMException if the unit of measure of two values
   *             are incompatible
   * @throws DatasweepException if an error occurred when getting the
   *             identified quantity
```

```
    */
   public static MeasuredValue[] getMaterialBalanceQuantitiesOfBatch(
                 ProcessOrderItem order, Batch batch)
       throws MESIncompatibleUoMException, DatasweepException {
                 MeasuredValue[] accountedQuantities = new MeasuredValue[6];
     List<String> processedSublots = new ArrayList<String>();
     // Get all OSIs because split positions of a same BOM position can have different
batches
     List<OrderStepInput> osisFromBatch = getOSIsFromBatch(order, batch, true);
     for (OrderStepInput osi : osisFromBatch) {
       /*
        * Get Identified quantity from Transaction history. Sum up quantity
        * of sublots when identified the first time in this order.
        */
       if (!processedSublots.contains(osi.getAttachSublot().getUniqueName())) {
         accountedQuantities[INDEX_IDENTIFIED_QUANTITY] =
         MeasuredValueUtilities.addArgsOptional(
             accountedQuantities[INDEX_IDENTIFIED_QUANTITY],
             getIdentifiedSublotQuantity(osi), null);
         processedSublots.add(osi.getAttachSublot().getUniqueName());
       }
       MeasuredValue[] accountedQuantitiesFromOSI = getAccountedQuantitiesFromOSI(osi);
       // We only take planned quantity of master OSIs
       if (!LONG_TRUE.equals(MESNamedUDAOrderStepInput.getIsSpitCopy(osi))) {
         accountedQuantities[INDEX_PLANNED_QUANTITY] =
             MeasuredValueUtilities.addArgsOptional(
             accountedQuantities[INDEX_PLANNED_QUANTITY],
                   osi.getPlannedQuantity(), null);
       }
       accountedQuantities[INDEX_CONSUMED_QUANTITY] =MeasuredValueUtilities
           .addArgsOptional(accountedQuantities[INDEX_CONSUMED_QUANTITY],
               accountedQuantitiesFromOSI[INDEX_CONSUMED_QUANTITY],null);
       accountedQuantities[INDEX_WASTED_QUANTITY] = MeasuredValueUtilities
           .addArgsOptional(accountedQuantities[INDEX_WASTED_QUANTITY],
               accountedQuantitiesFromOSI[INDEX_WASTED_QUANTITY], null);
       accountedQuantities[INDEX_SAMPLED_QUANTITY] = MeasuredValueUtilities
           .addArgsOptional(accountedQuantities[INDEX_SAMPLED_QUANTITY],
               accountedQuantitiesFromOSI[INDEX_SAMPLED_QUANTITY],null);
     }
     if (accountedQuantities[INDEX_CONSUMED_QUANTITY] == null &&
         accountedQuantities[INDEX_WASTED_QUANTITY] == null
         && accountedQuantities[INDEX_SAMPLED_QUANTITY] == null) {
       // If nothing is accounted yet, no quantity is returned
       accountedQuantities[INDEX_RETURNED_QUANTITY] = null;
     } else {
       accountedQuantities[INDEX_RETURNED_QUANTITY] =
           MeasuredValueUtilities.subtractArgsOptional(
           accountedQuantities[INDEX_IDENTIFIED_QUANTITY],
             MeasuredValueUtilities.addArgsOptional(
               accountedQuantities[INDEX_CONSUMED_QUANTITY],
                 MeasuredValueUtilities.addArgsOptional(
                   accountedQuantities[INDEX_WASTED_QUANTITY],
                   accountedQuantities[INDEX_SAMPLED_QUANTITY], null), null), null);
     }
     return accountedQuantities;
   }
   /**
    * Recursive method which sums up accounted quantities (only consumed,
    * wasted and sampled) from a given OSI over next transferred order step
    * inputs.
    *
    * @param osi the Order Step Input the starting point from where calculate
    *             the accounted quantities
    * @return an array of MeasuredValue containing at indexes:
    *          INDEX_PLANNED_QUANTITY: null.
    *          INDEX_IDENTIFIED_QUANTITY: null.
    *          INDEX_CONSUMED_QUANTITY: Sum up of consumed
      *          quantities of the batch identified in this osi.
    *          INDEX_WASTED_QUANTITY: Sum up of wasted quantities
    *          of the batch identified in this osi.
    *          INDEX_SAMPLED_QUANTITY: Sum up of sampled quantities
    *          of the batch identified in this osi.
    *          INDEX_RETURNED_QUANTITY: null.
    * @throws MESIncompatibleUoMException if the unit of measure of two values
    *              are incompatible
    */
   private static MeasuredValue[] getAccountedQuantitiesFromOSI(
     OrderStepInput osi) throws MESIncompatibleUoMException {
       MeasuredValue[] accountedQuantities = new MeasuredValue[6];
     Vector<OrderStepOutput> osos = osi.getAssociatedOrderStepOutputs();
     for (OrderStepOutput oso : osos) {
       // Get Transfer OSIs
       Vector<OrderStepInput> succOSIs = oso.getAssociatedOrderStepInputs();
       if (osi.getProcessingType() == IOrderStepInputTypes.INPUT_TYPE_INPUT
         && succOSIs.isEmpty()) {
         /*
          * If the OSI is an input and it has no transfer, then there is
          * no accounting => no wasted and sampled. Then no more
          * recursive call.
          */
         accountedQuantities[INDEX_CONSUMED_QUANTITY] =
           MESNamedUDAOrderStepInput.getConsumedQuantity(osi);
         MeasuredValue zeroQuantity = MeasuredValueUtilities.createMV(BigDecimal.ZERO,
           MESNamedUDAOrderStepInput.getConsumedQuantity(osi).getUnitOfMeasure());
         // Wasted and Sampled quantities are 0
         accountedQuantities[INDEX_WASTED_QUANTITY] =
           accountedQuantities[INDEX_SAMPLED_QUANTITY] = zeroQuantity;
       } else {
         for (OrderStepInput succOSI : succOSIs) {
           // Recursive call to sum accounted quantities of successors
           MeasuredValue[] accountedQuantitiesSuccOsi =
             getAccountedQuantitiesFromOSI(succOSI);
           accountedQuantities[INDEX_CONSUMED_QUANTITY] =
             MeasuredValueUtilities.addArgsOptional(
               accountedQuantities[INDEX_CONSUMED_QUANTITY],
               accountedQuantitiesSuccOsi[INDEX_CONSUMED_QUANTITY], null);
           accountedQuantities[INDEX_WASTED_QUANTITY] =
             MeasuredValueUtilities.addArgsOptional(
               accountedQuantities[INDEX_WASTED_QUANTITY],
               accountedQuantitiesSuccOsi[INDEX_WASTED_QUANTITY], null);
           accountedQuantities[INDEX_SAMPLED_QUANTITY] =
             MeasuredValueUtilities.addArgsOptional(
               accountedQuantities[INDEX_SAMPLED_QUANTITY],
               accountedQuantitiesSuccOsi[INDEX_SAMPLED_QUANTITY], null);
           // Add current accounted quantities
           List<OrderStepInputSublot> identifiedQuantity =
             osService.getIdentifiedOSISublots(succOSI, false, false);
           for (OrderStepInputSublot osiSublot : identifiedQuantity) {
             if (osiSublot.getBatch().equals(osi.getAttachSublot().getBatch())) {
               accountedQuantities[INDEX_CONSUMED_QUANTITY] = MeasuredValueUtilities
                    .addArgsOptional(accountedQuantities[INDEX_CONSUMED_QUANTITY],
                       osiSublot.getConsumedQuantity(), null);
               accountedQuantities[INDEX_WASTED_QUANTITY] =
                 MeasuredValueUtilities.addArgsOptional(
                   accountedQuantities[INDEX_WASTED_QUANTITY],
                   osiSublot.getWastedQuantity(), null);
               accountedQuantities[INDEX_SAMPLED_QUANTITY] =
                 MeasuredValueUtilities.addArgsOptional(
                   accountedQuantities[INDEX_SAMPLED_QUANTITY],
                     osiSublot.getSampledQuantity(), null);
             }
           }
         }
       }
     }
     return accountedQuantities;
   }
 }
```

Step 2

Create the *IMaterialBalanceConverter* interface defining the following methods

- The *IMaterialBalanceConverter* interface must extend *IMESService*.

```
public List<MaterialBalanceItemType> convert(List<OrderStepInput> osis)
```

- Creates a list of **MaterialBalanceItemType** types for a given list of **OrderStepInput** objects

```
public MaterialAccountingDetailsType convert(MeasuredValue identifiedQuantity,
MeasuredValue consumedQuantity, MeasuredValue sampledQuantity, MeasuredValue
wastedQuantity, MeasuredValue returnedQuantity)
```

- Creates the **MaterialAccountingDetailsType** XML type and initializes all the fields

Step 3

Create the *MaterialBalanceConverter* class implementing the *IMaterialBalanceConverter* interface

1. Initialize a map containing
   - Key: Part (Material)
   - Value: list of batches identified by the order

```
private void initMaps(List<OrderStepInput> osis) {
    batchesByMaterial = new HashMap<Part, List<Batch>>();
    for (OrderStepInput osi : osis) {
      // We only process Master OSI which are INPUTS (not transfer) and not replaced
      // by alternative
      if (LONG_TRUE.equals(MESNamedUDAOrderStepInput.getIsSpitCopy(osi))
          || osiService.isReplacedByAlternative(osi)
          || osi.getProcessingType() != IOrderStepInputTypes.INPUT_TYPE_INPUT) {
        continue;
      }
      List<OrderStepInputSublot> identifiedOSISublots = ServiceFactory.getService(
          IOrderStepExecutionService.class).getIdentifiedOSISublots(osi, false, false);
      for (OrderStepInputSublot osisub : identifiedOSISublots) {
        // Map the batch with its corresponding material
        List<Batch> batchesTmp = batchesByMaterial.get(osisub.getBatch().getPart());
        if (batchesTmp == null) {
          batchesTmp = new ArrayList<Batch>();
        }
        // We do not want to have duplicated batches
        if (!batchesTmp.contains(osisub.getBatch())) {
          batchesTmp.add(osisub.getBatch());
        }
      batchesByMaterial.put(osisub.getBatch().getPart(), batchesTmp);
    }
  }
}
```

2. Create a method to create a **MaterialBalanceItem** element from a Part (Material) object
   - Get the accounting details from batches of the material
   - Calculate the total from the accounting details
   - Retrieve the accounting details from a batch by getting all OSIs where the batch is identified and perform the following steps for each OSI:
     - Sum up the identified quantity: sum of sublots' quantities as they were identified for the first time in the order (from transaction history)
     - Sum up the planned quantity: sum up the planned quantities of each position (from the position that has not been replaced by an alternative master OSI) for which the batch has been identified (from OSI's **Planned Quantity** field)

    - Sum up the consumed/wasted/sampled quantities: sum of consumed/wasted/sampled quantities stored in every subsequent transfer (in the succeeding order steps)
- Calculate the returned quantity with this formula: identified - consumed - wasted - sampled
- Sort the batch list by batch identifier

```
  /**
   * Created a material balance item of a part.
   *
   * @param part the part for which create the material balance item
   * @return the object created
   */
  private MaterialBalanceItemType createMaterialBalanceItem(Part part) {
    MaterialBalanceItemType materialBalanceConverted =
      MaterialBalanceItemType.Factory.newInstance();
    List<MaterialBalanceItemDetailsType> itemsDetails = new
      ArrayList<MaterialBalanceItemDetailsType>();

    IBatchConverter batchService = ServiceFactory.getService(IBatchConverter.class);
    IQuantityValueConverter quantityService =
      ServiceFactory.getService(IQuantityValueConverter.class);

    MeasuredValue idTotal = null, consTotal = null, wastTotal = null, samplTotal =
null,
      returnedTotal = null, plannedTotal = null, diffPlannedConsumed = null;

    try {
      for (Batch batch : batchesByMaterial.get(part)) {
        MaterialAccountingDetailsBean details = createAccountingDetails(batch);

        MaterialBalanceItemDetailsType itemConverted =
          MaterialBalanceItemDetailsType.Factory.newInstance();
        itemConverted.setBatch(batchService.convert(batch));
        itemConverted.setMaterialAccountingDetails(convert(
          details.getIdentifiedQuantity(), details.getConsumedQuantity(),
          details.getSampledQuantity(), details.getWastedQuantity(),
          details.getReturnedQuantity()));
        itemsDetails.add(itemConverted);

        idTotal = MeasuredValueUtilities.addArgsOptional(idTotal,
          details.getIdentifiedQuantity(), null);
        consTotal = MeasuredValueUtilities.addArgsOptional(consTotal,
          details.getConsumedQuantity(), null);
        wastTotal = MeasuredValueUtilities.addArgsOptional(wastTotal,
          details.getWastedQuantity(), null);
        samplTotal = MeasuredValueUtilities.addArgsOptional(samplTotal,
          details.getSampledQuantity(), null);
        returnedTotal = MeasuredValueUtilities.addArgsOptional(returnedTotal,
          details.getReturnedQuantity(), null);
        plannedTotal = MeasuredValueUtilities.addArgsOptional(plannedTotal,
          details.getPlannedQuantity(), null);

        diffPlannedConsumed = MeasuredValueUtilities.subtractArgsOptional(plannedTotal,
          consTotal, null);
      }
    } catch (MESIncompatibleUoMException e) {
      LOGGER.error(e.getMessage());
      throw new RuntimeException(e);
    }

    Collections.sort(itemsDetails, ComparatorBatchID.INSTANCE);

    materialBalanceConverted.setPlannedQuantity(quantityService.convert(plannedTotal));
    materialBalanceConverted.setDiffPlannedConsumedQuantity(
      quantityService.convert(diffPlannedConsumed));
    materialBalanceConverted.setMaterialBalanceItemDetailsArray(itemsDetails
      .toArray(new MaterialBalanceItemDetailsType[0]));
    materialBalanceConverted.setMaterialAccountingTotal(convert(idTotal, consTotal,
      samplTotal, wastTotal, returnedTotal));

    return materialBalanceConverted;
  }

  /**
   * Creates the accounting details from identified sublot information. See
   * MaterialAccountingDetailsBean for more information.
   *
   * @param batch the batch for which create the accounting information.
   * @param positionsAlreadyHandled a list of BOM positions already handled.
   *            Mandatory to calculate the planned quantity. This list can be
   *            modified.
   * @return the batch accounting details
   */
  private MaterialAccountingDetailsBean createAccountingDetails(Batch batch) {
    MaterialAccountingDetailsBean details = new MaterialAccountingDetailsBean(batch);

    try {
      MeasuredValue[] accountedQuantities = MaterialBalanceHelperMaterialBalanceHelper
        .getMaterialBalanceQuantitiesOfBatch(currentOrder, batch);

       details.setPlannedQuantity(accountedQuantities[MaterialBalanceHelper
         .INDEX_PLANNED_QUANTITY]);
       details.setIdentifiedQuantity(accountedQuantities[MaterialBalanceHelper
         .INDEX_IDENTIFIED_QUANTITY]);
       details.setConsumedQuantity(accountedQuantities[MaterialBalanceHelper
         .INDEX_CONSUMED_QUANTITY]);
      details.setWasteQuantity(accountedQuantities[MaterialBalanceHelper
         .INDEX_WASTED_QUANTITY]);
       details.setSampledQuantity(accountedQuantities[MaterialBalanceHelper
         .INDEX_SAMPLED_QUANTITY]);
       details.setReturnedQuantity(accountedQuantities[MaterialBalanceHelper
         .INDEX_RETURNED_QUANTITY]);
    } catch (MESIncompatibleUoMException e) {
      LOGGER.error(e);
      throw new MESRuntimeException(e);
    } catch (DatasweepException e) {
      LOGGER.error(e);
      throw new MESRuntimeException(e);
    }

    return details;
  }
  /**
   * A bean object containing material accounting details. This object represents
accounting
   *  information about one batch in one order.
   */
  private class MaterialAccountingDetailsBean {
    /** Accounting quantities */
    private MeasuredValue identifiedQuantity = null, consumedQuantity = null,
      wasteQuantity = null, sampledQuantity = null, returnedQuantity = null,
      plannedQuantity = null;

    private Batch batch;

    public Batch getBatch() {
      return batch;
    }

    public MaterialAccountingDetailsBean(Batch batch) {
      this.batch = batch;
    }

    public MeasuredValue getIdentifiedQuantity() {
      return identifiedQuantity;
    }

    public void setIdentifiedQuantity(MeasuredValue identifiedQuantity) {
      this.identifiedQuantity = identifiedQuantity;
    }

    public MeasuredValue getConsumedQuantity() {
      return consumedQuantity;
    }

    public void setConsumedQuantity(MeasuredValue consumedQuantity) {
      this.consumedQuantity = consumedQuantity;
    }

    public MeasuredValue getWastedQuantity() {
      return wasteQuantity;
    }

    public void setWasteQuantity(MeasuredValue wasteQuantity) {
      this.wasteQuantity = wasteQuantity;
    }

    public MeasuredValue getSampledQuantity() {
      return sampledQuantity;
    }

    public void setSampledQuantity(MeasuredValue sampledQuantity) {
      this.sampledQuantity = sampledQuantity;
    }

    public MeasuredValue getReturnedQuantity() {
      return returnedQuantity;
    }

    public void setReturnedQuantity(MeasuredValue returnedQuantity) {
      this.returnedQuantity = returnedQuantity;
    }

    public MeasuredValue getPlannedQuantity() {
      return plannedQuantity;
    }

    public void setPlannedQuantity(MeasuredValue plannedQuantity) {
      this.plannedQuantity = plannedQuantity;
    }
}
```

```
/**
 * Comparator to sort a list of MaterialBalanceItemDetailsType by their IDs.
 */
static class ComparatorBatchID implements Comparator<MaterialBalanceItemDetailsType> {

  /** Constructor */
  public ComparatorBatchID() {
  }

  /** Singleton instance */
  public static final ComparatorBatchID INSTANCE = new ComparatorBatchID();

  /**
   * @param b1 a batch
   * @param b2 a batch to compare with b1
   * @return -1 if b1 must be placed before b2.
   *         And returns 1 if b2 must be placed before b1. see
   *         java.util.Comparator#compare(java.lang.Object, java.lang.Object)
   */
    @Override
  public int compare(final MaterialBalanceItemDetailsType b1, final
    MaterialBalanceItemDetailsType b2) {
    return b1.getBatch().getIdentifier().getStringValue().compareTo(b2.getBatch()
      .getIdentifier().getStringValue());
    }
}
```

3. Implement converter methods

```
  @Override
  public List<MaterialBalanceItemType> convert(List<OrderStepInput> osis) {
    List<MaterialBalanceItemType> convertedtems = new
      ArrayList<MaterialBalanceItemType>();
    if (osis.isEmpty()) {
      return convertedtems;
    }
    currentOrder = osis.get(0).getOrderStep().getControlRecipe().getProcessOrderItem();
    initMaps(osis);
    IMaterialConverter materialService = ServiceFactory
      .getService(IMaterialConverter.class);

    for (Part part : batchesByMaterial.keySet()) {
      MaterialBalanceItemType itemsConverted = createMaterialBalanceItem(part);
      itemsConverted.setMaterial(materialService.convert(part));
      convertedtems.add(itemsConverted);
    }

    Collections.sort(convertedtems, ComparatorMaterialName.INSTANCE);
    return convertedtems;

  }

  @Override
  public MaterialAccountingDetailsType convert(MeasuredValue identifiedQuantity,
    MeasuredValue consumedQuantity, MeasuredValue sampledQuantity, MeasuredValue
    wastedQuantity, MeasuredValue returnedQuantity) {

    IQuantityValueConverter quantityService =
      ServiceFactory.getService(IQuantityValueConverter.class);

    MaterialAccountingDetailsType detailsConverted =
      MaterialAccountingDetailsType.Factory.newInstance();
    detailsConverted.setIdentifiedQuantity(quantityService.convert(
      identifiedQuantity));
    detailsConverted.setConsumedQuantity(quantityService.convert(consumedQuantity));
    detailsConverted.setSampledQuantity(quantityService.convert(sampledQuantity));
    detailsConverted.setWastedQuantity(quantityService.convert(wastedQuantity));
    detailsConverted.setReturnedQuantity(quantityService.convert(returnedQuantity));

    return detailsConverted;
  }
/**
 * Comparator to sort a list of MaterialBalanceItemType by material name.
 */
static class ComparatorMaterialName implements Comparator<MaterialBalanceItemType> {

  /** Constructor */
  public ComparatorMaterialName() {
  }

  /** Singleton instance */
  public static final ComparatorMaterialName INSTANCE = new ComparatorMaterialName();

  /**
   * @param o1 an order related BOM item
   * @param o2 an order related BOM item to compare with o1
   * @return -1 if o1 must be placed before o2.
   *         And returns 1 if o2 must be placed before o1. see
   *         java.util.Comparator#compare(java.lang.Object, java.lang.Object)
   */
  @Override
  public int compare(final MaterialBalanceItemType o1, final MaterialBalanceItemType
    o2) {
    return o1.getMaterial().getIdentifier().getStringValue()
      .compareTo(o2.getMaterial().getIdentifier().getStringValue());

  }
}
```

Implementing the MaterialBalanceCollector Collector

A collector is a class to get all the data needed for a use case within a specific context and to use a converter service to convert the data into B2MML type instance(s).
The *MaterialBalanceCollector* class retrieves a list of **MaterialBalanceItemType** instances for a given *IMESControlRecipe*.

To implement the *MaterialBalanceCollector* collector, proceed as follows:

Step 1

Create the *IMaterialBalanceCollector* interface defining the following method

- The *IMaterialBalanceCollector* interface must extend *IMESService*.

```
public List<MaterialBalanceItemType>
getMaterialBalanceOfControlRecipe(IMESControlRecipe s88ControlRecipe)
```

Step 2

Implement the *IMaterialBalanceCollector* interface above

- Get a list of **OrderStepInputItems** objects from the **OrderSteps** objects and convert it to a list of **MaterialBalanceItemType** instances

```
  @Override
  public List<MaterialBalanceItemType> getMaterialBalanceOfControlRecipe(
    IMESControlRecipe s88ControlRecipe) {
    IMaterialBalanceConverter converter =
      ServiceFactory.getService(IMaterialBalanceConverter.class);
    Vector<OrderStep> orderSteps = s88ControlRecipe.getControlRecipe().getOrderSteps();
    List<OrderStepInput> allOSIs = new ArrayList<OrderStepInput>();
    for (OrderStep orderStep : orderSteps) {
      allOSIs.addAll(orderStep.getOrderStepInputItems());
    }
    return converter.convert(allOSIs);
  }
```

Extending Report Generation-specific Aggregators

An aggregator is a class called in the context of a use case (e.g. generate a batch report, phase report). The following aggregators need to be extended:

- *OrderRecordAggregator* generates a batch report within the **Manage Orders** task in the Production Management Client
- *UnitProcedureRecordAggregator* generates a unit procedure report (= phase report) in the Print Report phase of a Dispense operation within the Production Execution Client.

To extend the aggregators, proceed as follows:

Step 1

OrderRecordAggregator

- Override the *createBatchRecord* method to add the **Material Balance** section

```
public class OrderRecordAggregatorCustom extends OrderRecordAggregator {
  /**
   * Constructor
   *
   * @param poi The ProcessOrderItem representing the order to create the
   *            Batch Record for
   */
  public OrderRecordAggregatorCustom(ProcessOrderItem poi) {
    super(poi);
  }
  @Override
  protected BatchProductionRecordDocument createBatchRecord() {
    BatchProductionRecordDocument document = super.createBatchRecord();
    IMaterialBalanceCollector materialBalanceCollector =
      ServiceFactory.getService(IMaterialBalanceCollector.class);
    document.getBatchProductionRecord().addNewMaterialBalance()
      .getMaterialBalanceItemsList().addAll(materialBalanceCollector
      .getMaterialBalanceOfControlRecipe(getControlRecipe()));
    return document;
  }
}
```

Step 2

UnitProcedureRecordAggregator

- Override the *createBatchRecord* method to add the **Material Balance** section

```
public class UnitProcedureRecordAggregatorCustom extends UnitProcedureRecordAggregator
{
  public UnitProcedureRecordAggregatorCustom(IMESUnitProcedure rtUnitProc) {
    super(rtUnitProc);
  }
  @Override
  protected BatchProductionRecordDocument createBatchRecord() {
    BatchProductionRecordDocument document = super.createBatchRecord();
    IMaterialBalanceCollector materialBalanceCollector =
      ServiceFactory.getService(IMaterialBalanceCollector.class);
    document.getBatchProductionRecord().addNewMaterialBalance()
      .getMaterialBalanceItemsList().addAll(materialBalanceCollector
      .getMaterialBalanceOfControlRecipe(getControlRecipe()));
    return document;
  }
}
```

Extending IBatchProductionRecordDocumentWrapperExtension Interface

The *IBatchProductionRecordDocumentWrapperExtension* interface wraps all the data of a batch production record including mappings and provides a helper method to the Jasper report design. To extend the class, proceed as follows:

- For details, see [A2] (page 140)

Step 1

Create an implementation of the *IBatchProductionRecordDocumentWrapperExtension* interface implementing *addCustomerExtensions* method, (see Step4 of section "Creating Fast Bean Readers" (page 125))

Creating fast bean readers

The *MESB2MMLJRDataSource* class improves the performance when values are read from the B2MML attributes. The class can also map a simple string variable to a complex access path for a B2MML attribute. We recommend registering fast bean readers for each additional attribute in the batch record schema in case the attribute is used very often. Create a fast bean accessor for each new attribute of the extended B2MML type. We recommend using an enumeration to bundle all attributes of the same type in one file.

To extend the class, proceed as follows:

Step1

Create the *EnumBatchProductionRecordDocumentWrapperCustom* class to map the string variable to the data source of all items of the *MaterialBalanceItemType* type in the batch record.

```
/**
 * Enumeration of report field descriptions for {@link
IBatchProductionRecordDocumentWrapper}
*/
enum EnumBatchProductionRecordDocumentWrapperCustom
    implements IFastBeanPropertyAccessor<IBatchProductionRecordDocumentWrapper> {


    MATERIAL_BALANCE("materialBalanceItems") {
        @Override
        public JRDataSource getPropertyValue(
            IBatchProductionRecordDocumentWrapper currentBean) {
            final List<MaterialBalanceItemType> materialBalanceItemsList =
                currentBean.getBatchRecord().getBatchProductionRecord().
                getMaterialBalance().getMaterialBalanceItemsList();
                return IMESB2MMLJRDataSource.
                    createB2MMLJRDataSource(materialBalanceItemsList);
            }
        };
    private final String accessPath;

    private EnumBatchProductionRecordDocumentWrapperCustom(String bmlAccessPath) {
        this.accessPath = bmlAccessPath;
    }

    @Override
    public String getAccessPath() {
        return accessPath;
    }

}
```

Step 2

Create the *EnumMaterialBalanceItemType* class to map the string variable to get its assigned value of the attributes of the *MaterialBalanceItemType* type.

```
/**
 * Enumeration of report field descriptions for {@link MaterialBalanceItemType}
 */
enum EnumMaterialBalanceItemType implements
IFastBeanPropertyAccessor<MaterialBalanceItemType> {

    /** material name */
    MATERIAL_NAME("materialBalanceItem.MaterialName") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return currentBean.getMaterial().getIdentifier().getStringValue();
        }
    },

    /** material description */
    MATERIAL_DESCRIPTION("materialBalanceItem.MaterialDescription") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return currentBean.getMaterial().getShortDescription().getStringValue();
        }
    },

    /** material description */
    DETAILS_LIST("materialBalanceItem.MaterialBalanceItemDetailsList") {
        @Override
        public JRDataSource getPropertyValue(MaterialBalanceItemType currentBean) {
            return IMESB2MMLJRDataSource.createB2MMLJRDataSource(
                     currentBean.getMaterialBalanceItemDetailsList());
        }
    },

    /** planned quantity */
    PLANNED_QUANTITY("materialBalanceItem.PlannedQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(currentBean.getPlannedQuantity());
        }
    },

    /** difference between planned and consumed quantity */
    DIFF_PLANNED_CONSUMED_QUANTITY("materialBalanceItem.DiffPlannedConsumedQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(
                currentBean.getDiffPlannedConsumedQuantity());
        }
    },

    /** total identified quantity */
    IDENTIFIED_QUANTITY_TOTAL("materialBalanceItem.IdentifiedQuantityTotal") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(
                currentBean.getMaterialAccountingTotal().getIdentifiedQuantity());
        }
    },

    /** total consumed quantity */
    CONSUMED_QUANTITY_TOTAL("materialBalanceItem.ConsumedQuantityTotal") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(
                currentBean.getMaterialAccountingTotal().getConsumedQuantity());
        }
    },

    /** total wasted quantity */
    WASTED_QUANTITY_TOTAL("materialBalanceItem.WastedQuantityTotal") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(
               currentBean.getMaterialAccountingTotal().getWastedQuantity());
        }
    },

    /** total sampled quantity */
    SAMPLED_QUANTITY_TOTAL("materialBalanceItem.SampledQuantityTotal") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(
                currentBean.getMaterialAccountingTotal().getSampledQuantity());
        }
    },

    /** total returned quantity */
    RETURNED_QUANTITY_TOTAL("materialBalanceItem.ReturnedQuantityTotal") {
        @Override
        public String getPropertyValue(MaterialBalanceItemType currentBean) {
            return getLocalizedMeasuredValue(
                currentBean.getMaterialAccountingTotal().getReturnedQuantity());
        }
    };

    private final String accessPath;

    private EnumMaterialBalanceItemType(String bmlAccessPath) {
        this.accessPath = bmlAccessPath;
    }
    @Override
    public String getAccessPath() {
        return accessPath;
    }

    private static String getLocalizedMeasuredValue(QuantityValueType quantity) {
        return BatchReportLocalizationHelper.getLocalizedMeasuredValue(quantity);
    }
```

Step 3

Create the *EnumMaterialBalanceItemDetailsType* class to map the string variable to get its assigned value of the attributes of the *MaterialBalanceItemDetailsType* type.

```
/**
 * Enumeration of report field descriptions for {@link MaterialBalanceItemType}
 */
    /** batch identifier  */
    BATCH_ID("materialBalanceItemDetail.BatchID") {
        @Override
        public String getPropertyValue(MaterialBalanceItemDetailsType currentBean) {
            return currentBean.getBatch().getIdentifier().getStringValue();
        }
    },

    /** total identified quantity */
    IDENTIFIED_QUANTITY("materialBalanceItemDetail.IdentifiedQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemDetailsType currentBean) {
            final MaterialAccountingDetailsType materialAccountingDetails
               = currentBean.getMaterialAccountingDetails();
            return getLocalizedMeasuredValue(
                materialAccountingDetails.getIdentifiedQuantity());
        }
    },

    /** consumed quantity */
    CONSUMED_QUANTITY("materialBalanceItemDetail.ConsumedQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemDetailsType currentBean) {
            final MaterialAccountingDetailsType materialAccountingDetails
                = currentBean.getMaterialAccountingDetails();
            return getLocalizedMeasuredValue(
                materialAccountingDetails.getConsumedQuantity());
        }
    },

    /** wasted quantity */
    WASTED_QUANTITY("materialBalanceItemDetail.WastedQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemDetailsType currentBean) {
            final MaterialAccountingDetailsType materialAccountingDetails
               = currentBean.getMaterialAccountingDetails();
            return getLocalizedMeasuredValue(
               materialAccountingDetails.getWastedQuantity());
        }
    },

    /** sampled quantity */
    SAMPLED_QUANTITY("materialBalanceItemDetail.SampledQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemDetailsType currentBean) {
            final MaterialAccountingDetailsType materialAccountingDetails
                = currentBean.getMaterialAccountingDetails();
            return getLocalizedMeasuredValue(
                materialAccountingDetails.getSampledQuantity());
        }
    },

    /** returned quantity */
    RETURNED_QUANTITY("materialBalanceItemDetail.ReturnedQuantity") {
        @Override
        public String getPropertyValue(MaterialBalanceItemDetailsType currentBean) {
            final MaterialAccountingDetailsType materialAccountingDetails
                = currentBean.getMaterialAccountingDetails();
            return getLocalizedMeasuredValue(
                materialAccountingDetails.getReturnedQuantity());
        }
    };
    private final String accessPath;

    private EnumMaterialBalanceItemDetailsType(String bmlAccessPath) {
        this.accessPath = bmlAccessPath;
    }

    @Override
    public String getAccessPath() {
        return accessPath;
    }

    private static String getLocalizedMeasuredValue(QuantityValueType quantity) {
        return BatchReportLocalizationHelper.getLocalizedMeasuredValue(quantity);
    }

}
```

Step 4

Register the instances of the fast bean accessors created in Step 1 of the section "Extending the IBatchProductionRecordDocumentWrapperExtension Interface" (page 125).

```
public void addCustomerExtensions () {
        final IMESFastBeanPropertyAccessMappings fastBeanAccessMappings = //
                IMESFastBeanPropertyAccessMappings.getSingletonInstance();
        fastBeanAccessMappings.initExpressionMappings();
        fastBeanAccessMappings.registerFastAccessors(
                IBatchProductionRecordDocumentWrapper.class, //
                EnumBatchProductionRecordDocumentWrapperCustom.values() );
        fastBeanAccessMappings.registerFastAccessors(
                MaterialBalanceItemType.class, //
                EnumMaterialBalanceItemType.values());
        fastBeanAccessMappings.registerFastAccessors(
                MaterialBalanceItemDetailsType.class, //
                EnumMaterialBalanceItemDetailsType.values());
}
```

Configuring Affected Services

Configure the new service implementation by overriding the default service configuration. For this purpose, proceed as follows:

- For details, see [A3] (page 140)

Step 1

Create the *extension-config.xml* file and add references to the new service configuration files (e.g. *myc-own-batchrecord-services.xml*, *myc-own-batchreport-services.xml*)

```
<configuration name="extension-config">
  <hierarchicalXml fileName="myc-own-batchrecord-services.xml" />
  <hierarchicalXml fileName="myc-own-batchreport-services.xml" />
</configuration>
```

Step 2

Create your service configuration files (e.g. *myc-own-batchrecord-services.xml, myc-own-batchreport-services.xml*) with the implementation classes of your service implementations

1. Batch record-related service implementations (e.g. *com.rockwell.mes.services.batchrecord.extensionuc.impl.OrderRecordAggregatorCustom*, *com.rockwell.mes.services.batchrecord.extensionuc.impl.UnitProcedureAggregatorCustom*)

```
<configuration>
  <OrderRecordAggregator>
    <implementationClass>
      com.rockwell.mes.services.batchrecord.extensionuc.impl
        .OrderRecordAggregatorCustom
    </implementationClass>
  </OrderRecordAggregator>

  <UnitProcedureAggregator>
    <implementationClass>
      com.rockwell.mes.services.batchrecord.extensionuc.impl
        .UnitProcedureAggregatorCustom
    </implementationClass>
  </UnitProcedureAggregator>
</configuration>
```

2. Batch report-related service implementations (e.g. *com.rockwell.mes.services.batchreport.extensionuc.materialbalance.impl.MESB2MMLJRDataSourceCustom*, *com.rockwell.mes.services.batchreport.extensionuc.materialbalance.impl.BatchProductionRecordDocumentWrapperCustom*)

```
<configuration>
  <BatchProductionRecordDocumentWrapperExtension>

    <implementationClass>

      com.rockwell.mes.services.batchreport.extensionuc.materialbalance.

         impl.BatchProductionRecordDocumentWrapperCustomerExtension

    </implementationClass>

  </BatchProductionRecordDocumentWrapperExtension>

</configuration>
```

Adapting the Report Design of the Batch Report

To adapt the report design, integrate the **Material Balance** section into the main report, create sub-reports describing the new section, and add them to the main report. For this purpose, proceed as follows:

- For details, see [A2] (page 140)

Step 1

Add messages for the new section to the **PS-BatchReport** message pack

- Example message ID: Heading_MaterialBalanceItems_Title (=section header)
- Example message: MATERIAL BALANCE
- Example message ID: MaterialBalanceItem_MaterialID_Label (= material and separator between material name and description)
- Example message: Material: {0} / {1}
- Example message ID: MaterialBalanceItem_DiffPlannedConsumed_Label (=formula description)
- Example message: Planned quantity - Consumed quantity: {0} - {1} = {2}
- Example message ID: MaterialBalanceItem_BatchID_Label (= column header)
- Example message: Batch ID
- Example message ID: MaterialBalanceItem_IdentifiedQuantity_Label (=column header)
- Example message: Identified

- Example message ID: MaterialBalanceItem_ConsumedQuantity_Label (= column header)
- Example message: Consumed
- Example message ID: MaterialBalanceItem_WastedQuantity_Label (= column header)
- Example message: Wasted
- Example message ID: MaterialBalanceItem_SampledQuantity_Label (= column header)
- Example message: Sampled
- Example message ID: MaterialBalanceItem_ReturnedQuantity_Label (= column header)
- Example message: Returned
- Example message ID: MaterialBalanceItem_Total_Label (= column header)
- Example message: Total

Step 2

Create the **MaterialBalanceItemDetails** sub-report containing the detail data of **MaterialBalanceItem** instances

1. In the PharmaSuite report manager, define the sub-report:
   - Example sub-report name: MaterialBalanceItemDetails
   - Example sub-report variable name: SUBREPORT_MATERIAL_BALANCE_ITEM_DETAILS
2. In Jaspersoft Studio, add fields to reference each **MaterialBalanceItemDetail** instance.
   - The description must match the identifiers defined in the fast bean reader class (see section "Creating Fast Bean Readers (page 125)"):
   - Example field name: PS_MaterialBalanceItemDetailsBatchID
   - Example field name: PS_MaterialBalanceItemDetailsIdentifiedQuantity
   - Example field name: PS_MaterialBalanceItemDetailsConsumedQuantity
   - Example field name: PS_MaterialBalanceItemDetailsWastedQuantity
   - Example field name: PS_MaterialBalanceItemDetailsSampledQuantity
   - Example field name: PS_MaterialBalanceItemDetailsReturnedQuantity

3. Add a detail band containing each item detail
   - For each column header, add a label with a **Text Field Expression** referencing the message IDs of Step 1.
     Example: $R{MaterialBalanceItem_BatchID_Label}
   - For each column value, add a label with a **Text Field Expression** referencing the fields of sub-step 2.
     If the referenced value is null, the label shall display a "N/A text".
     Example: ($F{PS_MaterialBalanceItemDetailsSampleQuantity} == null || $F{PS_MaterialBalanceItemDetailsSampleQuantity}.equals("")) ? $R(NotApplicable_Text}:$F{PS_MaterialBalanceItemDetailsSampleQuantit y}
   - Use component styles to differentiate the header from values
4. Import the report design by means of the **mes_PS-BatchReportManager** form.

Step 3

Create the **MaterialBalanceItems** sub-report containing the data of **MaterialBalanceItem** instances

1. In the PharmaSuite report manager, define the sub-report:
   - Example sub-report name: MaterialBalanceItems
   - Example sub-report variable name: SUBREPORT_MATERIAL_BALANCE_ITEMS
   - Example of report added sub-report list: PS-BatchReport-MaterialBalanceItemDetails
2. In Jaspersoft Studio, add the following parameters:
   - Name: MATERIAL_BALANCE_ITEMS_CHAPTER_NUMBER
     - Parameter class: java.lang.String
   - Name: SUBREPORT_MATERIAL_BALANCE_ITEM_DETAILS
     - Parameter class: java.lang.Object
3. Add fields to reference each **MaterialBalanceItem** instance.
   - The description must match the identifiers defined in the fast bean reader class (see section "Creating Fast Bean Readers (page 125)"):
   - Example field name: PS_MaterialBalanceItemMaterialName
   - Example field name: PS_MaterialBalanceItemMaterialDescription
   - Example field name: PS_MaterialBalanceItemPlannedQuantity
   - Example field name: PS_MaterialBalanceItemDiffPlannedConsumedQuantity

- Example field name: PS_MaterialBalanceItemDetailsIdentifiedQuantityTotal
- Example field name:
  PS_MaterialBalanceItemDetailsConsumedQuantityTotal
- Example field name: PS_MaterialBalanceItemDetailsWastedQuantityTotal
- Example field name: PS_MaterialBalanceItemDetailsSampledQuantityTotal
- Example field name: PS_MaterialBalanceItemDetailsReturnedQuantityTotal

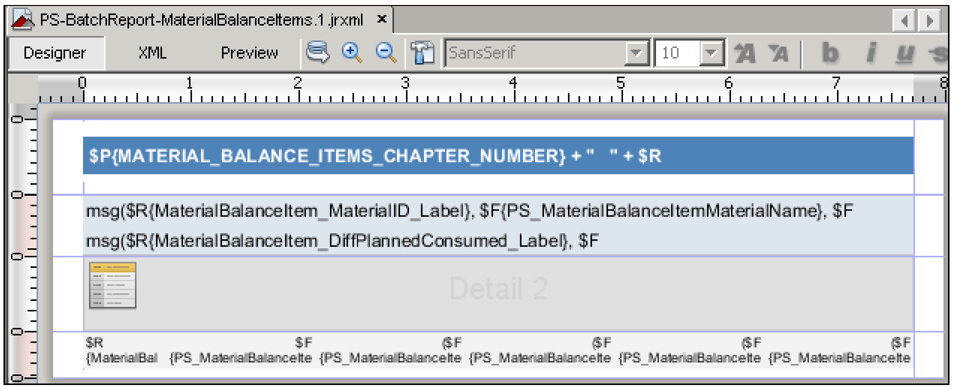
4. Configure the title
   Example: $P{MATERIAL_BALANCE_ITEMS_CHAPTER_NUMBER} + "   "
   + $R{Heading_MaterialBalanceItems_Title}
   - Add a hyperlink anchor to the title label
     Example anchor name expression:
     $R{Heading_MaterialBalanceItems_Title}
5. Add detail bands
   1. Add two labels with a **Text Field Expression** to the title band.
      Example (first label):
      msg($R{MaterialBalanceItem_MaterialID_Label},
      $F{PS_MaterialBalanceItemMaterialName},
      $F{PS_MaterialBalanceItemMaterialDescription})
      Example (second label):
      msg($R{MaterialBalanceItem_DiffPlannedConsumed_Label},
      $F{PS_MaterialBalanceItemPlannedQuantity},
      $F{PS_MaterialBalanceItemDetailsConsumedQuantityTotal},
      $F{PS_MaterialBalanceItemDiffPlannedConsumedQuantity},
      $F{PS_MaterialBalanceItemDetailList})
      - Do not display the second line if the total consumed quantity is null.
        Example (print when exception):
        $F{PS_MaterialBalanceItemDetailsConsumedQuantityTotal} != null
        &&
        !$F{PS_MaterialBalanceItemDetailsConsumedQuantityTotal}.equals(""
)

2. Add the **MaterialBalanceItemDetails** sub-report.
   Example sub-report parameter name: REPORT_DATA_SOURCE
   Example sub-report expression:
   $P{SUBREPORT_MATERIAL_BALANCE_ITEM_DETAILS}

   - Pass on the BATCH_RECORD_WRAPPER and REPORT_DATA_SOURCE parameters.
     The data source is retrieved from the $F{PS_MaterialBalanceItemDetailList} field.
     Example sub-report parameter name: BATCH_RECORD_WRAPPER
     Example value expression: $P{BATCH_RECORD_WRAPPER}
     Example sub-report parameter name: REPORT_DATA_SOURCE
     Example value expression: $F{PS_MaterialBalanceItemDetailList}

3. Add labels with a **Text Field Expression** to display the total accounting quantities.
   If the referenced value is null, the label shall display a "N/A text".
   Example: ($F{PS_MaterialBalanceItemDetailsConsumedQuantityTotal} == null ||
   $F{PS_MaterialBalanceItemDetailsConsumedQuantityTotal}.equals("")) ?
   $R(NotApplicable_Text}:$F{PS_MaterialBalanceItemDetailsConsumedQua
   ntityTotal}

6. Example of the report design



$P{MATERIAL_BALANCE_ITEMS_CHAPTER_NUMBER} + " " + $R

msg($R{MaterialBalanceItem_MaterialID_Label}, $F{PS_MaterialBalanceItemMaterialName}, $F
msg($R{MaterialBalanceItem_DiffPlannedConsumed_Label}, $F

$R {MaterialBal | $F {PS_MaterialBalanceIte | $F {PS_MaterialBalanceIte | $F {PS_MaterialBalanceIte | $F {PS_MaterialBalanceIte | $F {PS_MaterialBalanceIte

Figure 16: MaterialBalanceItems report design

7. Import the report design by means of the **mes_PS-BatchReportManager** form.

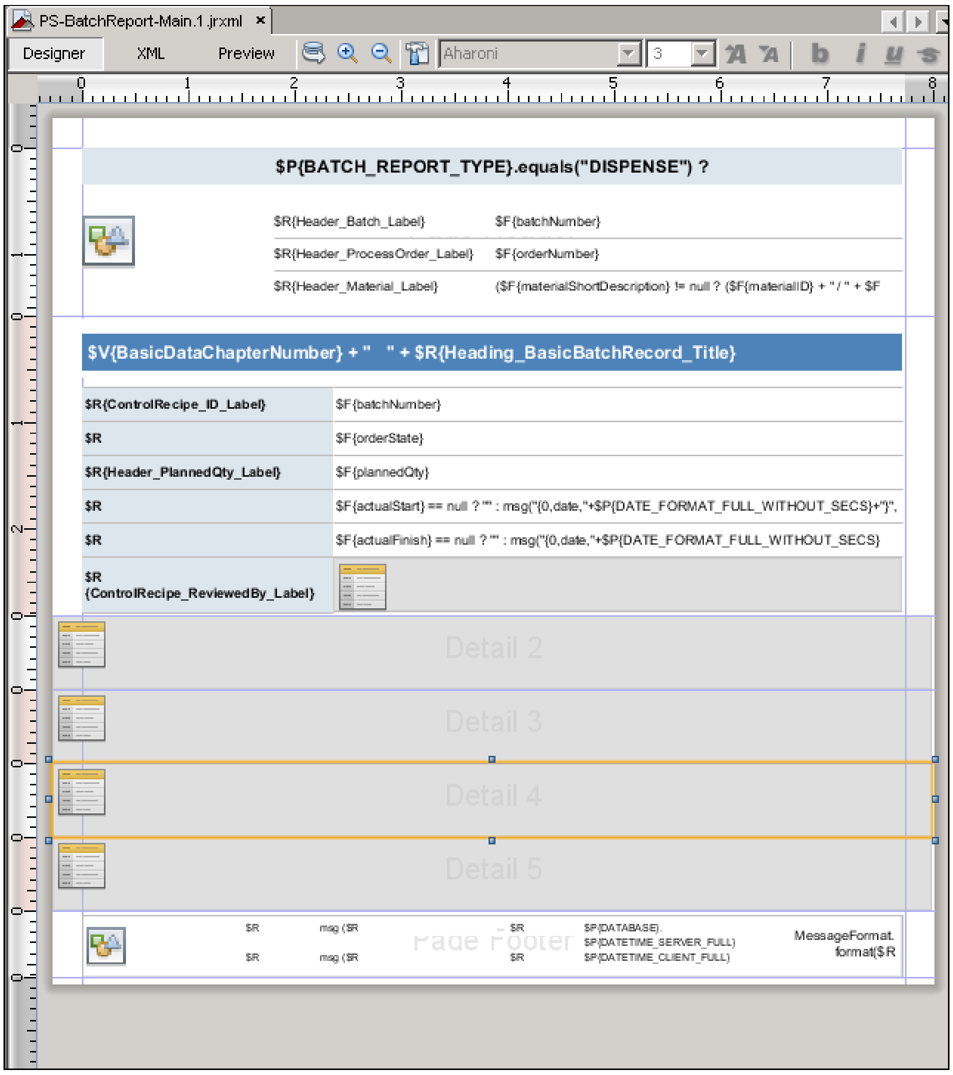
8. In the **mes_PS-BatchReportManager** form add a reference to the **PS-BatchReport-MaterialBalanceItemDetails** sub-report.

Step 4

Modify the batch report with Jaspersoft Studio

1. In the PharmaSuite report manager,create a copy of the PS-BatchReport-Main report design, you can call it e.g. PS-BatchReport-Main-MaterialBalance.
2. Assign the new main batch report to your default configuration.
3. Export the batch report to an **JRXML** file.
4. Add the **materialBalanceItems** field referencing the data source
5. Add the **MaterialBalanceItemsChapterNumber** variable containing the section number
   - Example: Initial Value Expression: `new String("4")`
6. Add a new detail band
   - Add the **MaterialBalanceItems** sub-report.
     Example: Subreport Expression:
     $$P{REPORT_PARAMETERS_MAP}.get("SUBREPORT_MATERIAL_BALANCE_ITEMS")
     Expression Class: net.sf.jasperreports.engine.JasperReport
     Parameters Map Expression: new Hash($P{REPORT_PARAMETERS_MAP})
     Connection type: Don't pass data.
   - Parameters of the sub-report (bold font indicates that the parameter is passed on):
     - REPORT_PARAMETERS_MAP: $P{REPORT_PARAMETERS_MAP}
     - **MATERIAL_BALANCE_ITEMS_CHAPTER_NUMBER**: $V{MaterialBalanceItemsChapterNumber}
     - REPORT_RESOURCE_BUNDLE: $P{REPORT_RESOURCE_BUNDLE}
     - **BATCH_RECORD_WRAPPER**: $F{batchRecordWrapper}
     - **REPORT_DATA_SOURCE**: $F{materialBalanceItems}

7. Example of the report design



$P{BATCH_REPORT_TYPE}.equals("DISPENSE") ?

$R{Header_Batch_Label}	$F{batchNumber}
$R{Header_ProcessOrder_Label}	$F{orderNumber}
$R{Header_Material_Label}	($F{materialShortDescription} != null ? ($F{materialID} + " / " + $F

$V{BasicDataChapterNumber} + "   " + $R{Heading_BasicBatchRecord_Title}

$R{ControlRecipe_ID_Label}	$F{batchNumber}
$R	$F{orderState}
$R{Header_PlannedQty_Label}	$F{plannedQty}
$R	$F{actualStart} == null ? "" : msg("{0,date,"+$P{DATE_FORMAT_FULL_WITHOUT_SECS}+"}",
$R	$F{actualFinish} == null ? "" : msg("{0,date,"+$P{DATE_FORMAT_FULL_WITHOUT_SECS}
$R {ControlRecipe_ReviewedBy_Label}	

```
$R     msg ($R     $R     $P{DATABASE}.                MessageFormat.
                          $P{DATETIME_SERVER_FULL}       format($R
$R     msg ($R     $R     $P{DATETIME_CLIENT_FULL}
```

Figure 17: PS-BatchReport-Main report design

8. Import the report design by means of the **mes_PS-BatchReportManager** form.

9. In the **mes_PS-BatchReportManager** form, add a reference to the **PS-BatchReport-MaterialBalanceItems** sub-report.

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Batch production record schema file	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Working with the Batch Record API ■ What Is the Batch Record API ■ Extending the Data Structure of the Batch Record API	PSCEV2-GR010B-EN-E
A2	■ BatchProductionRecordDocumentWrapper ■ Report Design	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Batch Reports in PharmaSuite ■ Adding Sub-reports and Phase-specific Sub-reports	PSCEV2-GR010B-EN-E
A3	■ Configuring Services	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Managing Services ■ Adapting the Behavior of Existing Services ■ Configuring Service Implementations	PSCEV2-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Chapter 10

Displaying a Retest Date Column in the Overview Panel of the Exception Dashboard

This section describes how to add a column to display the retest date of a batch in the Overview panel of the Exception Dashboard, a part of the Production Response Client.

In the default configuration, the first column of the Overview panel contains the order identifier.

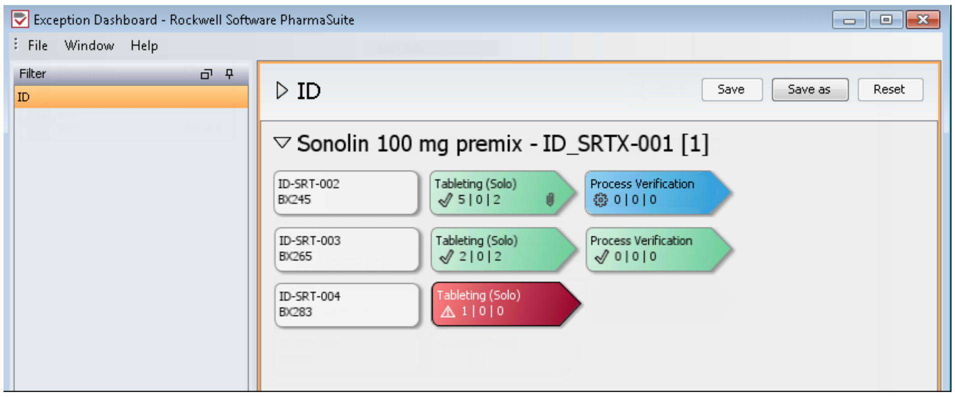


Figure 18: Default configuration of Overview panel

PharmaSuite provides the *IDashboardCustomizableColumnCreator* interface to add exactly one column to the left of the Overview panel (see "Using the IDashboardCustomizableColumnCreator Interface" (page 143)). If required, you can display several fields in one column.

The description covers two variants:

- Display the Retest date column without further configuration (page 144)

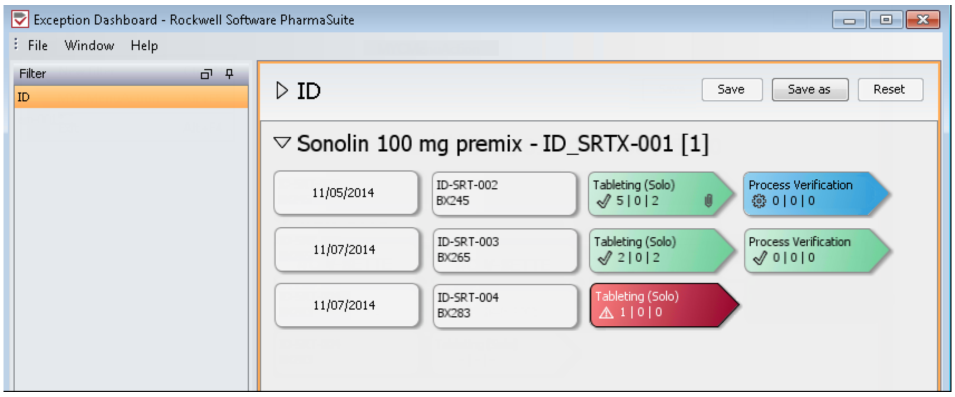

Exception Dashboard - Rockwell Software PharmaSuite
File
Window
Help
Filter
ID
ID
Save
Save as
Reset
Sonolin 100 mg premix - ID_SRTX-001 [1]
11/05/2014
ID-SRT-002 BX245
Tableting (Solo) 5 | 0 | 2
Process Verification 0 | 0 | 0
11/07/2014
ID-SRT-003 BX265
Tableting (Solo) 2 | 0 | 2
Process Verification 0 | 0 | 0
11/07/2014
ID-SRT-004 BX283
Tableting (Solo) 1 | 0 | 0

Figure 19: Additional Retest date column

- Display the Retest date column and modify the size, define the content to be displayed and an update interval for the display color, and provide a tooltip (page 145)

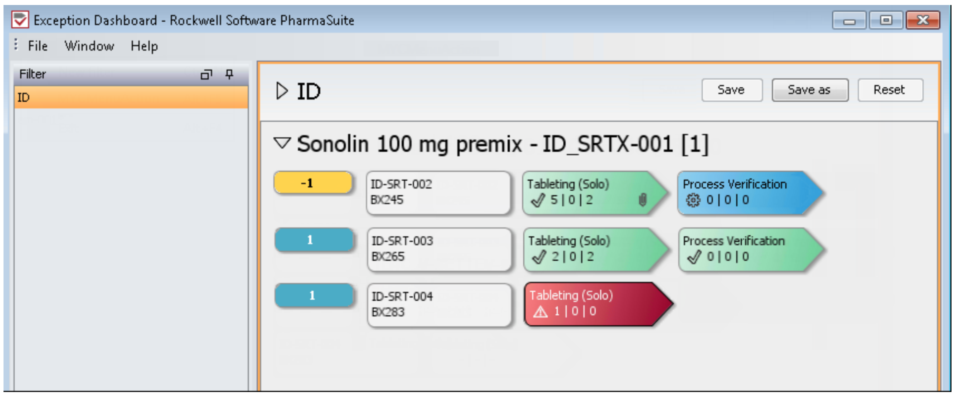

Exception Dashboard - Rockwell Software PharmaSuite
File
Window
Help
Filter
ID
ID
Save
Save as
Reset
Sonolin 100 mg premix - ID_SRTX-001 [1]
-1
ID-SRT-002 BX245
Tableting (Solo) 5 | 0 | 2
Process Verification 0 | 0 | 0
1
ID-SRT-003 BX265
Tableting (Solo) 2 | 0 | 2
Process Verification 0 | 0 | 0
1
ID-SRT-004 BX283
Tableting (Solo) 1 | 0 | 0

Figure 20: Additional Retest date column with configured display

TIP

Internationalization is not covered in the examples.

Using the IDashboardCustomizableColumnCreator Interface

The *IDashboardCustomizableColumnCreator* interface defines two methods, one to create the order-related field of the new column and a deprecated one that requires only a dummy implementation. The method delivers a *JComponent* (typically a derived class of *JButton*) to display the data. To retrieve the required information, an instance of the *IProcessOrderItemModel* object holding the process order item's data is passed to the method, which creates the order-related field.

```
package com.rockwell.mes.apps.exceptiondashboard.ifc.view.controls;
import javax.swing.JComponent;
import com.datasweep.compatibility.client.MasterRecipe;
import com.datasweep.compatibility.client.ProcessOrderItem;

public interface IDashboardCustomizableColumnCreator {
    /**
     * Creates the JComponent for the column header.
     * @param poi the process order item
     * @return the JComponent
     * @deprecated since PharmaSuite6.0 - PRC does not display table headers
     */
    JComponent createColumnHeader(MasterRecipe mr);

    /**
     * Creates the JComponent for the column field.
     *
     * @param poi the process order item
     * @return the JComponent
     */
    JComponent createColumnField(IProcessOrderItemModel poi);
}
```

Configuring the Customizable Column Creator

The implementation class for the customizable column creator can be configured by means of a configuration key in the **Application** object (Default configuration) in Process Designer. For details, see **LibraryHolder/apps-exceptiondashboard-impl.jar/CustomizableColumnCreator** configuration key in Volume 4 of the "Technical Manual Configuration and Extension" [A1] (page 153).

TIP

PharmaSuite provides the PS Administration application to administer database objects such as access privileges, lists, applications, users, or user groups, see "Implementation Guide PS Administration" [A1] (page 187).
Using Process Designer for these tasks is possible, but does not provide the full functional scope required for PharmaSuite.

Displaying the Retest Date Column

To display the Retest date column without further configuration, proceed as follows:

Step 1

Implement the *IDashboardCustomizableColumnCreator* interface

- The *DashBoardSimpleRetestColumnCreator* implementation class makes use of the *FieldButton* Exception Dashboard control to implement the interface and to achieve the same look and feel for all columns of the Overview Panel of the Exception Dashboard. The data to be displayed by the order-related field button (retest date) is extracted from the batch accessible by the process order item.

```
public class DashboardSimpleRetestColumnCreator implements
             IDashboardCustomizableColumnCreator {
  @Override
  public JComponent createColumnHeader(MasterRecipe mr) {
    return null;
  }
  @Override
  public JComponent createColumnField(IProcessOrderItemModel poi) {
    final Long batchKey = poi.getBatchKey();
    final Batch batch = (batchKey != null) ? PCContext.getFunctions()
                        .getBatchByKey(batchKey) : null;
    final Time time = (batch != null) ? MESNamedUDABatch.getRetestDate(batch) : null;
    final String dateString;
    if (time != null) {
      final String propertyName = "UDA_X_retestDate";
      // compare with MESNamedUDABatch#UDA_RETESTDATE
      dateString = MESDateUtility.getLocalizedDateTime(time, Batch.class,
propertyName);
    } else {
      dateString = "";
    }
    final FieldButton button = new FieldButton();
    button.setText(dateString);
    return button;
  }
}
```

Step 2

Set the **LibraryHolder/apps-exceptiondashboard-impl.jar/CustomizableColumnCreator** configuration key

- For details, see [A1] (page 153)

Displaying the Retest Date Column with Configured Layout

To display the Retest date column and configure the layout, proceed as follows:

Step 1

Implement the *IDashboardCustomizableColumnCreator* interface

- The *DashboardRetestColumnCreator* implementation class covers the configuration of the size, the update interval, and a tooltip.

Step 2

Modify the size

1. To modify the size (height and width of column and rows) do not directly use *FieldButton* in the *DashboardRetestColumnCreator* implementation class. Instead, instantiate a class: *RetestFieldButton* extends *FieldButton*.

```
public class DashboardRetestColumnCreator implements
             IDashboardCustomizableColumnCreator {
    @Override
    public JComponent createColumnHeader(MasterRecipe mr) {
        return null;
    }
    @Override
    public JComponent createColumnField(IProcessOrderItemModel poi) {
        final RetestFieldButton button = new RetestFieldButton(poi);
        return button;
    }
}
```

2. Define the constructor, which accepts the process order item data and defines the button size.

```
public class RetestFieldButton extends FieldButton {
    private static final Dimension BUTTON_SIZE =
            new Dimension(DEFAULT_BUTTON_SIZE.width / 2, DEFAULT_BUTTON_SIZE.height /
2);
    public RetestFieldButton(IProcessOrderItemModel processOrderItem) {
        super(BUTTON_SIZE);
        poi = processOrderItem;
    }
}
```

Step 3

Define the content and update interval for the display color

- The retest date column displays the number of day(s) from now until the retest date. The order-related field's text is dynamically updated and the color changes depending on the number of day(s). The color scheme is as follows: red, if the retest date is in the past; yellow, if the retest date is today; green, if the retest date is in the future.
  To dynamically update the content of the field's text, define a *Timer* object in the *RetestFieldButton* class, which updates an internal *State* object of the button. For this purpose, implement the *ActionListener* interface. Here, the update interval of the *Timer* object is set to 60 seconds. Adapt the settings according to your requirements. The State is later used to draw the *RetestFieldButton* in the defined color (red, yellow, green).
  Also implement the *AncestorListener* interface to be informed of SwingEvents when a *Button* control is added to or removed from the panel hierarchy of the Exception Dashboard. This is used to start and stop the *Timer* object properly.
  To enable a state-dependent drawing, use the separate *RetestFieldButtonUI* class, which uses the suitable gradient color scheme and allows to make the button and the border invisible, if the state is gray. This is done in the *drawShape()* method.

```
public class RetestFieldButton extends FieldButton implements
             ActionListener, AncestorListener {
  private static final long serialVersionUID = 1L;
  private static final Log LOGGER = LogFactory.getLog(RetestFieldButton.class);
  /** Color schema of the button */
  public enum State {
    /** gray : retest date is unknown */
    GRAY,
    /** green : retest date is in the future. */
    GREEN,
    /** yellow : retest date is today. */
    YELLOW,
    /** red : retest data is passed. */
    RED
  }
  /** State of the button */
  private State state = State.GRAY;
  /** The corresponding process order item */
  private final IProcessOrderItemModel poi;
  /**
   * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is gray.
   */
  public static final Color COLOR_GRAY = ExceptionReviewUIConfig.getBackground();
  /**
   * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is green.
   */
  public static final Color COLOR_FROM_GREEN = new Color(204, 255, 153);
  /**
   * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is green.
   */
  public static final Color COLOR_TO_GREEN = new Color(153, 255, 153);
  /**
   * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is red.
```

```
    */
  public static final Color COLOR_FROM_RED = new Color(247, 59, 59);
  /**
    * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is red.
    */
  public static final Color COLOR_TO_RED = new Color(255, 10, 10);
  /**
    * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is
yellow.
    */
  public static final Color COLOR_FROM_YELLOW = new Color(255, 255, 153);
  /**
    * Gradient <b>to</b> color to display {@link RetestFieldButton} when state is
yellow.
    */
  public static final Color COLOR_TO_YELLOW = new Color(255, 255, 92);
  /** update interval in milliseconds */
  private static final int UPDATE_INTERVAL_MILLIS = 60 * 1000;
  /** Timer for update */
  private Timer timer;
  /**
    * Size of this control
    * Same as new Dimension(ExceptionReviewUIConfig.GRID_BUTTON_SIZE.
                         width / 2,
    * ExceptionReviewUIConfig.GRID_BUTTON_SIZE.height / 2)
    */
  private static final Dimension BUTTON_SIZE = new Dimension(DEFAULT_BUTTON_SIZE.width,
                               / 2, DEFAULT_BUTTON_SIZE.height / 2);
  /**
    * Constructing the object and provide update of its state via timer
    *
    * @param processOrderItem a ProcessOrderItemModel object
    */
  public RetestFieldButton(IProcessOrderItemModel processOrderItem) {
    super(BUTTON_SIZE);
    poi = processOrderItem;
    LOGGER.info("button for batch " + (poi.getBatchKey() == null ? "null" :
                poi.getBatchName()) + " created.");
    if (poi.getBatchKey() != null) {
      LOGGER.info("timer created for batch " + poi.getBatchName());
      // provide timer for update and register this as listener
      timer = new Timer(UPDATE_INTERVAL_MILLIS, null);
      // register this as listener for changes in the swing hierarchy
      this.addAncestorListener(this);
    }
    // force correct initial state
    setButtonState();
  }
  @Override
  public void updateUI() {
    setUI(RetestFieldButtonUI.createUI(this));
  }
  /**
    * Sets the color state of the button.
    *
    * @param aState of the button
    */
  public final void setState(State aState) {
    this.state = aState;
  }
  /**
    * Gets the color state of the button.
```

```
   *
   * @return the state of the button
   */
  public final State getState() {
    return this.state;
  }
  @Override
  public void actionPerformed(ActionEvent e) {
    setButtonState();
  }
  /**
   * Sets the state of the button depending on the days until the batch retest date.
   */
  private void setButtonState() {
    if (poi.getBatchKey() != null) {
      // we refresh the POI here, because some attributes (e.g.
      // retest date) it could be changed in the meantime
      poi.refresh();
      final Batch batch = PCContext.getFunctions().getBatchByKey(poi.getBatchKey());
      final Time retestDate = MESNamedUDABatch.getRetestDate(batch);
      if (retestDate != null) {
        final Time actualTime = PCContext.getCurrentServerTime();
        final long days = retestDate.toLong() / Time.UNITS_PER_DAY -
                          actualTime.toLong() / Time.UNITS_PER_DAY;
        final String retestData = MESDateUtility.getLocalizedDateTime(retestDate,
                                  IMESDateFormatConfig.FULL_DATETIME_SECONDS);
        final StringBuffer tooltip = new StringBuffer(String.format("The retest
date(%1$s)
                                 of batch(%2$s) is ", // retestData,
poi.getBatchName()));
        final State newState;
        if (days > 0) {
          // time is in the future
          newState = State.GREEN;
          tooltip.append(String.format("in %1$s day(s).", Long.toString(days)));
        } else if (days == 0) {
          // same day
          newState = State.YELLOW;
          tooltip.append("today.");
        } else {
          // time is over
          newState = State.RED;
          tooltip.append(String.format("over since %1$s day(s).", Long.toString(-
days)));
        }
        setText(Long.toString(days));
        setToolTipText(tooltip.toString());
        setState(newState);
        LOGGER.info("State for button using batch[" + this.poi.getBatchName() + "] is "
+
                    state.toString());
      } else {
        setState(State.GRAY);
      }
    } else {
      setState(State.GRAY);
    }
  }
  /**
   * Determine the fill <b>from</b> {@link Color} based on {@link #highlight} status.
   *
   * @return The fill <b>from</b> {@link Color}
```

```
    */
   @Override
   public Color getFillFrom() {
     final Color c;
     switch (state) {
     case YELLOW:
       c = COLOR_FROM_YELLOW;
       break;
     case RED:
       c = COLOR_FROM_RED;
       break;
     case GREEN:
       c = COLOR_FROM_GREEN;
       break;
     default:
       c = COLOR_GRAY;
       break;
     }
     return c;
   }
   /**
    * Determine the fill <b>to</b> {@link Color} based on {@link #highlight} status.
    *
    * @return The fill <b>to</b> {@link Color}
    */
   @Override
   public Color getFillTo() {
     final Color c;
     switch (state) {
     case YELLOW:
       c = COLOR_TO_YELLOW;
       break;
     case RED:
       c = COLOR_TO_RED;
       break;
     case GREEN:
       c = COLOR_TO_GREEN;
       break;
     default:
       c = COLOR_GRAY;
       break;
     }
     return c;
   }
   /**
    * Use this swing callback method to start the timer correctly when the control is
 added
    * by the framework.
    * {@inheritDoc}
    */
   @Override
   public void ancestorAdded(AncestorEvent event) {
     if (poi.getBatchKey() != null) {
       timer.addActionListener(this);
       timer.start();
       LOGGER.info("timer started for batch " + poi.getBatchName());
     }
   }
   /**
    * Use this swing callback method to stop the timer correctly when the control is
 removed
    * by the framework.
    * {@inheritDoc}
    */
   @Override
   public void ancestorRemoved(AncestorEvent event) {
     if (poi.getBatchKey() != null) {
       timer.stop();
       timer.removeActionListener(this);
       LOGGER.info("timer stopped for batch " + poi.getBatchName());
     }
   }
   @Override
   public void ancestorMoved(AncestorEvent event) {
     // not used here
   }
 }
```

Step 4

Provide a tooltip

- To display a tooltip for the retest column field, extend the *AbstractExceptionReviewButtonUI* class.
  The tooltip's text is set in the private *setButtonState()* method of the *RetestFieldButton* class, which can determine the correct text depending on the field's state.

```
public class RetestFieldButtonUI extends AbstractExceptionReviewButtonUI {
  /** Singleton */
  private static final RetestFieldButtonUI UI = new RetestFieldButtonUI();
  /** Singleton */
  private RetestFieldButtonUI() {
  }
  /**
   * Create PLAF.
   *
   * @param c The hosting {@link JComponent}
   * @return The created PLAF
   */
  public static ComponentUI createUI(JComponent c) {
    return UI;
  }
  /** The border color */
  private static final Color COLOR_BORDER =
ExceptionReviewUIConfig.COLOR_OUTLINE_BRIGHT;
                 // (140, 140, 140)
  @Override
  protected void drawShape(Graphics2D g2, Rectangle b, JComponent c) {
    final RetestFieldButton btn = ((RetestFieldButton) c);

    final Shape outlineShape =
        new Area(new RoundRectangle2D.Float(b.x, b.y, b.width - 1, b.height - 1,
                 ExceptionReviewUIConfig.DEFAULT_ARC_WIDTH,
                 ExceptionReviewUIConfig.DEFAULT_ARC_WIDTH));
    final Paint paint = horizonalGradient(outlineShape, btn.getFillFrom(),
                        btn.getFillTo());
    final Color borderColor = COLOR_BORDER;
    if (btn.getState() != RetestFieldButton.State.GRAY) {
      // render shadow, highlight (if selected) and shape (including gradient and
border)
      renderBackground(g2, outlineShape, paint, btn.getFillTo(), borderColor, c);
      // renderBackground() basically replaces calls to
      // g2.setPaint(paint);
      // g2.fill(outlineShape);
      // g2.setColor(borderColor);
      // g2.draw(outlineShape);
      // (i.e. rendering filled shape and shape's outline / border), but also takes
care
      // of rendering a shadow for the shape, and its highlight.
    }
    // else: no button rendered (--> 'invisible')
  }
}
```

Step 5

Set the **LibraryHolder/apps-exceptiondashboard-impl.jar/CustomizableColumnCreator** configuration key

- For details, see [A1] (page 153)

TIP

For further classes that support the configuration (e.g. (*ExceptionReviewUIConfig*, *DefaultDashboardControlButton*), see the *com.rockwell.mes.apps.exceptiondashboard.ifc.view.controls package*.

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ LibraryHolder/apps-exceptiondashboard-impl.jar/CustomizableColumnCreator configuration key	PharmaSuite Technical Manual Configuration & Extension - Volume 4: Configuration Keys of PharmaSuite ■ Configuration Keys for the Exception Dashboard	PSCEV4-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Adding the Print FDA Report Function to PharmaSuite Clients

This section describes how to add an FDA report in addition to the existing reports of PharmaSuite.
By default, only one report design can be used to generate a batch report and another report design can be used for a workflow report. However, this behavior can be changed in order to use multiple report designs for a batch report or a workflow report, respectively.

The extension use case requires a fully set up PharmaSuite development environment and Java know-how.

The description covers the following areas:

- Create the FDA report designs (page 156).
- Optional:
  Use different report designs depending on order usage types (page 156).
- Add new actions to the Production Management Client to open and print an FDA report (page 157).
- Add new actions to the Production Response Client to open and print an FDA report (page 161).
- Include the FDA report into exported records (page 164).

Creating the FDA Report Designs

To create the FDA report designs, it is necessary to create at least a main report. Sub-reports only need to be created if they differ from the standard sub-reports.

Step 1

Create reports and sub-reports

- For details, see [A1] (page 165)
- Depending on the report, the name of the report design must begin with the following prefix:

Report	Prefix
Batch report	**PS-BatchReport**
Workflow report	**PS-BatchReport-WorkflowOrder**

- Add a suffix like **FDA** to your report and name the designs accordingly.

Using Different Report Designs Depending on Order Usage Types

This step is optional.

If you wish to use another report design for example for an order of the **Cost Center** usage type, proceed as follows:

Step 1

Add the **LibraryHolder/services-wd-impl.jar/CostCenter.FDA** configuration key and reference the specific report design

- For details, see [A2] (page 165)

Adding New Actions to the Production Management Client

To open and print an FDA report for a batch order or a workflow, you can create corresponding actions for the Production Management Client.

- For details, see [A3] (page 165)

To add printing actions for an FDA report similar to the existing **Open batch report** and **Print batch report** actions in the **(Batch) Order management** use case, proceed as follows:

Step 1

Modify the configuration of the **ucOrder** and **ucWorkflowOrder** use cases to add actions to open and print the FDA report

1. Actions for the **(Batch) Order management** use case:

```
<Action id="acShowFDABatchReport"
  actionClass="com.rockwell.custmes.apps.order.impl.action.ShowFDABatchReport"
  icon="myc_pmc_ShowFDABatchReport"
  messagePack="myc_ui_Order" labelId="ShowFDABatchReport_Menu"
  defaultLabel="Open FDA batch report" tooltipId="ShowFDABatchReport_Hint"
  privilege="PMC_AccessRight_Datahandler_ViewBatchReport" >
</Action>

<Action id="acPrintFDABatchReport"
  actionClass="com.rockwell.custmes.apps.order.impl.action.PrintFDABatchReport"
  icon="myc_pmc_PrintFDABatchReport"
  messagePack="myc_ui_Order" labelId="PrintFDABatchReport_Menu"
  defaultLabel="Print FDA batch report" tooltipId="PrintFDABatchReport_Hint"
  privilege="PMC_AccessRight_Datahandler_ViewBatchReport" >
</Action>
```

2. Add these actions to the data template:

```
<!-- Multiple order nodes belonging to the parent filter -->
  <DataTemplate id="dtViewOrdersForDefaultFilter"
      dataClass="com.rockwell.mes.apps.order.ifc.model.ProcessOrderFastBeanModel"
      explodeToMultipleNodes="true">
    <DataHandler idref="dhGetByFastBeanReader">
      <Parameter key="UseBoundClassOfDataNodeInGrid" value="true" />
      <Parameter key="UseDataHandlerOfDataNodeInGrid" value="true" />
    </DataHandler>
    <Action idref="acSaveOrder" />
    <Action idref="acDeleteBatchFromOrder" />
    <Action idref="acUndo">
      <Parameter key="0" value="getName()" />
      <!-- message parameter-->
    </Action>
    <Action idref="acApplyTransOnPOI"/>
    <Action idref="acViewHistoryOfPOI"/>
    <Action idref="acShowBatchReport" />
    <Action idref="acShowFDABatchReport" />
    <Action idref="acPrintBatchReport" />
    <Action idref="acPrintFDABatchReport" />
[...]
```

3. Actions for the **Workflow management** use case:

```
<Action id="acShowFDAWorkflowReport"
  actionClass="com.rockwell.custmes.apps.workfloworder.impl.action.
               ShowFDAWorkflowOrderReport"
  icon="myc_pmc_ShowFDAWorkflowReport" messagePack="myc_ui_WorkflowOrder"
  labelId="ShowFDAWorkflowReport_Menu" defaultLabel="Open FDA workflow report"
  tooltipId="ShowFDAWorkflowReport_Hint" privilege=
     "PMC_AccessRight_Datahandler_ViewWorkflowReport">
</Action>

<Action id="acPrintFDAWorkflowReport"
  actionClass="com.rockwell.custmes.apps.workfloworder.impl.action.
               PrintFDAWorkflowOrderReport"
  icon="myc_pmc_PrintFDAWorkflowReport" messagePack="myc_ui_WorkflowOrder"
  labelId="PrintFDAWorkflowReport_Menu" defaultLabel="Print FDA workflow report"
  tooltipId="PrintFDAWorkflowReport_Hint" privilege=
     "PMC_AccessRight_Datahandler_ViewWorkflowReport">
</Action>
```

4. Add these actions to the data template:

```
<!-- Multiple workflow order nodes belonging to the parent filter -->
  <DataTemplate id="dtViewWorkflowOrdersForDefaultFilter"
      dataClass="com.rockwell.mes.apps.order.ifc.model.WorkflowOrderFastBeanModel"
      explodeToMultipleNodes="true">
    <DataHandler idref="dhGetByFastBeanReader">
      <Parameter key="UseBoundClassOfDataNodeInGrid" value="false" />
            <!-- use bound class of filter -->
      <Parameter key="UseDataHandlerOfDataNodeInGrid" value="true" />
    </DataHandler>
    <Action idref="acSaveWorkflow" />
    <Action idref="acUndo">
      <Parameter key="0" value="getName()" />
      <!-- message parameter -->
    </Action>
    <Action idref="acApplyTransOnWorkflow" />
    <Action idref="acShowWorkflowReport" />
    <Action idref="acShowFDAWorkflowReport" />
    <Action idref="acPrintWorkflowReport" />
    <Action idref="acPrintFDAWorkflowReport" />
[...]
```

Step 2

Implement the action classes. Reporting is handled by the *AbstractBatchReportAction* and *AbstractWorkflowOrderReportAction* abstract classes.

TIP

Please note that the design suffix given to the service which creates the report must match the name of the report design you created earlier. This suffix is the name right after the **PS-BatchReport-** prefix.
Examples: If your FDA report designs are called **PS-BatchReport-FDA** or **PS-BatchReport-WorkflowOrder-FDA**, the suffixes are **FDA** or **WorkflowOrder-FDA**.

1. **Open FDA batch report** action

```
public class ShowFDABatchReport extends AbstractBatchReportAction {
  private static final long serialVersionUID = 1L;
  /**
   * Default constructor.
   */
  public ShowFDABatchReport() {
    super(FDAReportHelper.BATCH_REPORT_FDA_DESIGN_SUFFIX, false);
  }
}
```

2. **Print FDA batch report** action

```
public class PrintFDABatchReport extends AbstractBatchReportAction {
  private static final long serialVersionUID = 1L;
  /**
   * Default constructor.
   */
  public PrintFDABatchReport() {
    super(FDAReportHelper.BATCH_REPORT_FDA_DESIGN_SUFFIX, true);
  }
}
```

3. **Open FDA workflow report** action

```
public class ShowFDAWorkflowOrderReport extends AbstractWorkflowOrderReportAction {
  private static final long serialVersionUID = 1L;
  /**
   * Default constructor.
   */
  public ShowFDAWorkflowOrderReport() {
    super(FDAReportHelper.WORKFLOW_ORDER_REPORT_FDA_DESIGN_SUFFIX, false);
  }
}
```

4. **Print FDA workflow report** action

```
public class PrintFDAWorkflowOrderReport extends AbstractWorkflowOrderReport {
  private static final long serialVersionUID = 1L;
  /**
   * Default constructor.
   */
  public PrintFDAWorkflowOrderReport() {
    super(FDAReportHelper.WORKFLOW_ORDER_REPORT_FDA_DESIGN_SUFFIX, true);
  }
}
```

Step 3

Consider to process the following items:

- As there is a status management for orders, actions can be enabled/disabled depending on the status of the selected order.
  To enable the **Open...** and **Print...** actions for orders, add each to the **VP_Ops_[Status]_POI** list, where [status] is the status when the action must be enabled.
  For details, see [A5] and [B1] (page 165)
- Add icons for the six new actions.
- Create/update the message packs referenced by the actions.

Adding New Actions to the Production Response Client

To open and print an FDA report for an order or workflow, you can create corresponding actions for the Production Response Client.

- For details, see [A4] (page 165)

To add printing actions for an FDA report similar to the existing **Open report** and **Print report** actions in the Details panel of the Exception Dashboard, proceed as follows:

Step 1

Modify the UI configuration of the Production Response Client

```
<?xml version="1.0" encoding="UTF-8"?>
<ui>
  <menubar key="mainMenuBar">
  [...]
  </menubar>
  <toolbar key="detailedToolBar">
    <item key="ToolBack" action="com.rockwell.mes.apps.exceptiondashboard.impl.
          action.EDBToolBarBackAction" />
    <item key="ToolHtml" action="com.rockwell.mes.apps.exceptiondashboard.impl.
          action.EDBShowBatchReportPreviewAction" />
    <item key="ToolHtmlFDA" action="com.rockwell.custmes.apps.exceptiondashboard.impl.
          action.EDBShowFDABatchReportPreviewAction" />
    <item key="ToolPDF" action="com.rockwell.mes.apps.exceptiondashboard.impl.
          action.EDBShowBatchReportPrintPreviewAction" />
    <item key="ToolPDFFDA" action="com.rockwell.custmes.apps.exceptiondashboard.impl.
          action.EDBShowFDABatchReportPrintPreviewAction" />
    <item key="ToolOK" action="com.rockwell.mes.apps.exceptiondashboard.impl.
          action.EDBSetBatchReviewedAction" />
    <item key="ToolHelp" action="com.rockwell.mes.apps.exceptiondashboard.impl.
          action.EDBToolBarHelpAction" />
  </toolbar>
  <perspectives>
[...]
  </perspectives>
</ui>
```

Step 2

Implement the action classes. The *ExceptionReviewBatchReporter* class supports the creation of a batch report and a workflow report within the Production Response Client.

1. **Open FDA report** action

```
public class EDBShowFDABatchReportPreviewAction extends AbstractAction {
  private static final long serialVersionUID = 1L;

  /** Create instance as early as possible to preload sub-reports */
  private final ExceptionReviewBatchReporter exceptionReviewBatchReporter =
                ExceptionReviewBatchReporter.INSTANCE;

  private static final Map<S88ProcessingType, String> FDA_REPORT_DESIGN_SUFFIXES =
            ExceptionReviewBatchReporter.getReportDesignNameSuffixMapping(//
            FDAReportHelper.BATCH_REPORT_FDA_DESIGN_SUFFIX, //
            FDAReportHelper.WORKFLOW_ORDER_REPORT_FDA_DESIGN_SUFFIX);

  @Override
  public void actionPerformed(ActionEvent e) {
    final ExceptionReviewModel exceptionReviewModel =
ExceptionReviewModel.getInstance();

    exceptionReviewBatchReporter.showReportPreview(exceptionReviewModel,
                                 FDA_REPORT_DESIGN_SUFFIXES);
  }
}
```

2. **Print FDA report** action

```
public class EDBShowFDABatchReportPrintPreviewAction extends AbstractAction {
  private static final long serialVersionUID = 1L;

  /** Create instance as early as possible to preload sub-reports */
  private final ExceptionReviewBatchReporter exceptionReviewBatchReporter =
                ExceptionReviewBatchReporter.INSTANCE;

  private static final Map<S88ProcessingType, String> FDA_REPORT_DESIGN_SUFFIXES =
            ExceptionReviewBatchReporter.getReportDesignNameSuffixMapping(//
            FDAReportHelper.BATCH_REPORT_FDA_DESIGN_SUFFIX, //
            FDAReportHelper.WORKFLOW_ORDER_REPORT_FDA_DESIGN_SUFFIX);

  @Override
  public void actionPerformed(ActionEvent e) {
    final ExceptionReviewModel exceptionReviewModel =
ExceptionReviewModel.getInstance();

    exceptionReviewBatchReporter.showReportPrintPreview(exceptionReviewModel,
                                 FDA_REPORT_DESIGN_SUFFIXES);
  }
}
```

TIP

The *showReportPreview()* and *showReportPrintPreview()* methods print a batch report or a workflow report based on the currently selected object. They print a workflow report if a workflow is selected or a batch report if an order or a unit procedure is selected.

To always print the batch report, use the *showReportPreview()* and *showReportPrintPreview()* methods instead, i.e. replace the calls to

exceptionReviewBatchReporter.showReportPreview(exceptionReviewModel, FDA_REPORT_DESIGN_SUFFIXES)
and
exceptionReviewBatchReporter.showReportPrintPreview(exceptionReviewModel, FDA_REPORT_DESIGN_SUFFIXES)

with

exceptionReviewBatchReporter.showBatchReportPreview(ExceptionReviewModel.getInstance().getSelectedProcessOrderItem(), exceptionReviewModel.getSelectedRtUnitProcedure().getRtUnitProcedure(), FDAReportHelper.BATCH_REPORT_FDA_DESIGN_SUFFIX)
and
exceptionReviewBatchReporter.showBatchReportPrintPreview(ExceptionReviewModel.getInstance().getSelectedProcessOrderItem(), FDAReportHelper.BATCH_REPORT_FDA_DESIGN_SUFFIX)

, respectively.

Step 3

Consider to process the following items:

- Add icons for the two new actions.
- Update the message packs to localize the tooltips.

Including the FDA Report into Exported Records

The **Export record** actions in the Production Management Client export the standard report, along with other records. You can configure which reports will be exported with the action when a batch record or workflow record is exported.

The configuration is done in two **List** objects; one for batch records and one for workflows records. The lists contain the default report designs to be used at export. If a configuration key references a different report design for a report type (here: FDA), the default report design is not used and the one referenced in the configuration key is used instead.

To configure the **Export record** actions, proceed as follows:

Step 1

Add the FDA report to the export of the batch record

- Add the **PS-BatchReport-FDA report** design to the **BatchReportsExported** list

Step 2

Add the FDA report to the export of the workflow record

- Add the **PS-BatchReport-WorkflowOrder-FDA** design to the **WorkflowOrderReportsExported** list

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Report design	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Batch Reports in PharmaSuite ■ Adding Sub-reports	PSCEV2-GR010B-EN-E
A2	■ Report designs for different usage types	PharmaSuite Technical Manual Configuration & Extension - Volume 4: Managing Configurations Configuration Keys of PharmaSuite ■ Configuration Keys Specific to the Production Execution Client	PSCEV4-GR010B-EN-E
A3	■ Actions in the Production Management Client	PharmaSuite Technical Manual Configuration & Extension - Volume 1: Adapting the Use Cases of the Production Management Client ■ Creating Actions for the Production Management Client	PSCEV1-GR010B-EN-E
A4	■ Actions in the Production Response Client	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Configuring Menu and Toolbar of the Production Response Client	PSCEV3-GR010B-EN-E
A5	■ Semantic properties	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Configuring Flexible State Models ■ Configuring PharmaSuite with Flexible State Models	PSCEV2-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

The following documents provide more details relevant to the implementation of the extension use case and are distributed with the FactoryTalk ProductionCentre installation.

No.	Keyword	Document Title	Part Number
B1	■ Overwrite value of semantic property	Process Designer Online Help: Flexible State Models ■ Configuring Additional FSM Features	N/A

TIP

To access the "Process Designer Online Help", use the following syntax: *http://<MES-PS-HOST>:8080/PlantOperations/docs/help/pd/index.htm*, where <MES-PS-HOST> is the name of your PharmaSuite server. To view the online help, the Apache Tomcat of the FactoryTalk ProductionCentre installation must be running.

Chapter 12

Adding the Print QA Report Function to Recipe and Workflow Designer

This section describes how to add a QA report in addition to the existing reports of Recipe and Workflow Designer.
By default, only one report design can be used to generate a master recipe report and another report design can be used for a master workflow report. However, this behavior can be changed in order to use multiple report designs for a master recipe report or master workflow report, respectively.

The extension use case requires a fully set up PharmaSuite development environment and Java know-how.

The description covers the following areas:

- Create the QA report designs (page 167).
- Add new actions to Recipe and Workflow Designer to print a QA report (page 168).

Creating the QA Report Designs

To create the QA report designs, it is necessary to create at least a main report. Sub-reports only need be created if they differ from the standard sub-reports.

Step 1

Create reports and sub-reports

- For details, see [A1] (page 170)
- Depending on the report, the name of the report design must begin with the following prefix:

Report	Prefix
Master recipe report for batches	**PS-MRReport-**
Master workflow report	**PS-MWFReport-**

- Add a suffix like **Main-QA** to your report and name the designs accordingly.

Adding New Actions to Recipe and Workflow Designer

To print a QA report for a master recipe or master workflow, you can create corresponding actions for Recipe and Workflow Designer.

- For details, see [A2] (page 170)

To add printing actions for a QA report similar to the existing **Print report** actions in the **File** menu, proceed as follows:

Step 1

Implement the action classes.

1. **Print QA batch report** action

```
package com.rockwell.custmes.apps.recipeeditor.impl.frame.action;
import java.awt.event.ActionEvent;

import com.rockwell.mes.apps.recipeeditor.ifc.frame.action.RDAbstractExternalAction;
import com.rockwell.mes.apps.recipeeditor.ifc.frame.model.IS88File;
import com.rockwell.mes.apps.recipeeditor.ifc.report.RDPrintReportHelper;
import com.rockwell.mes.clientfw.docking.ifc.editor.MESAbstractEditorInstance;
import com.rockwell.mes.commons.base.ifc.utility.AutoWaitCursor;
/**
 * Customer print QA report action.
 */
public class PrintQAReportAction extends RDAbstractExternalAction {
  /** Dummy <code>serialVersionUID</code> */
  private static final long serialVersionUID = 1L;
  /** suffix of main report design */
  private static final String DESIGN_SUFFIX = "Main-QA";
    /**
     * Constructor.
     *
     * @param name action name
     * @param editor editor instance
     */
    public PrintQAReportAction(String name, MESAbstractEditorInstance<?> editor) {
        super(name, editor);
    }

    @Override
    public void update() {
        final IS88File s88File = getFocusedS88File();
        setEnabled(RDPrintReportHelper.canPrintReport(s88File));
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        final IS88File s88File = getFocusedS88File();
        if (s88File == null) {
            return;
        }
        try (AutoWaitCursor waitCursor = createAutoWaitCursor()) {
            final boolean withComparisonResult = true;
            RDPrintReportHelper.printReport(s88File, getRDEditor(), DESIGN_SUFFIX,
                                withComparisonResult);
        }
    }
```

TIP

If you do not wish to include a comparison result-specific section in your report, set the *withComparisonResult* variable to *false*.

Step 2

Modify the database to add actions to print the QA report.

1. In the **toolbar** section of the **red_buttonConfiguration** list, add a new sub-menu containing the standard printing action. Add the new action in this new sub-menu. Example configuration:

```
<menu key="printMenu">
  <item key="printReport" />
  <item key="printQAReport"
    action="com.rockwell.custmes.apps.recipeeditor.impl.frame.action.
           PrintQAReportAction" />
</menu>
```

   - For details, see [A3] (page 170)

Step 3

Consider to process the following items:

- Create/update the message packs referenced by the actions.

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Report design	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Master Recipe Reports in PharmaSuite	PSCEV2-GR010B-EN-E
A2	■ Actions in Recipe and Workflow Designer	PharmaSuite Technical Manual Configuration & Extension - Volume 3: ■ Configuring Menu and Toolbar of Recipe and Workflow Designer	PSCEV3-GR010B-EN-E
A3	■ Add action to red_buttonConfiguration list	PharmaSuite Technical Manual Configuration & Extension - Volume 3: Managing the Layout of Recipe and Workflow Designer and Data Manager ■ Removing/Adding Items from/to the Menu Bar, Toolbar, Context Menu	PSCEV3-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Chapter 13

Automatic Archiving and Purging of Specific Orders and Workflows

This section describes how to create a script to automatically archive and purge a defined set of order and workflows. The script is based on the archive- and purge-specific API of PharmaSuite.

To keep a productive system responsive and easily maintainable, it is good practice to archive and purge orders and workflows that are no longer used. The following scenarios illustrate the different use cases for an archive or purge process to be applied on a productive system:

- In the simplest scenario, a user selects an order to archive or to purge and the whole process is executed synchronously on the client machine. This approach is suitable if only a very small number of orders needs to be archived and purged.
- In a more sophisticated scenario, a user selects a set of orders to archive or purge via the user interface. The system collects all the items to process for the orders. Then, the actual export (archive) or deletion is executed in background, controlled by a scheduler. Cleanup of temporary data is done in a different background task taking a retention period into account.
- In a scenario where a big number of orders has to be archived or purged per day, it is best to perform archiving or purging automatically without any user interaction, controlled by a set of rules.

For more details related to long-term archiving, please refer to chapter "Adapting the Export for Archive Process" and "Adapting the Purge Process" in Volume 2 of the "Technical Manual Configuration and Extension" [A1] and [A2] (page 177).

With this extension use case, an automatic script can be implemented to archive specific orders that were finished before a given timestamp. The same could be done to archive workflows or to purge specific orders or workflows that were exported before a given a timestamp.

The description covers the following steps:

- Collect the orders that shall be archived or purged.
- For collected orders, call the PharmaSuite API to archive or purge each order.

The scripts can be executed periodically as a service with an event sheet.

Collecting Orders

PharmaSuite provides a service to collect order keys that fulfill the condition for archiving or purging. It is provided with the *IArchivePurgeCollectingService*. There are methods for batch-specific orders and workflows.

The specification of the product and timestamp is defined by your needs. This way it is possible to have different time periods for different products.

Example code

```
...
IArchivePurgeCollectingService service =
             ServiceFactory.getService(IArchivePurgeCollectingService.class);
keys = service.getOrderKeysForArchive(aPart, toTime);
...
```

Collecting Orders for Archiving

A timestamp must be specified before which the orders or workflows were transitioned into the status to be archived. For batch-specific orders, it is possible to define a product to collect only the orders of a given product.

Method for batch-specific orders

```
/**
* The method for batch orders of a product if given and that are:
* - "Reviewed" or "Canceled" since the given time
* More precise is that the states are defined by having the semantic property
* "archivePurge.allow.Export" for batch orders.
* @param product the product of the order
* @param since the time since the order has been reviewed or canceled
* @return a list of keys of batch orders for the given product that are reviewed
*   or canceled since the given days and that fulfills the conditions for archiving
*/
public List<Long> getOrderKeysForArchive(final Part product, final Time since);
```

Method for workflow orders

```
/**
* The method for workflow orders that are:
* - "Finished, "Production-reviewed", "Reviewed" or "Canceled" since the given time
* - All of the orders it was appended to (i.e. its parent orders) are in a terminal
status
*   (there exists no valid transition or it is in a logically terminal status)
* More precise is that the states are defined by having the semantic property
* "archivePurge.allow.Export" for workflows. The logically terminal status is marked
* by the semantic property "archivePurge.status.logicallyTerminal".
* @param since the time since the order has been reviewed or canceled
* @return a list of keys of workflow orders that are finished, (production-)reviewed or
*  canceled since the given days and that fulfills the conditions for archiving
*/
public List<Long> getWorkflowKeysForArchive (final Time since);
```

Collecting Orders for Purging

In order to collect the orders or workflows to be purged, again a timestamp must be specified. Only orders and workflows are returned that were transitioned into the status to be purged before the given timestamp. For batch-specific orders it is possible to define a product to collect only the orders of a given product.

Method for batch-specific orders

```
/**
* The method for batch orders of a product if given and that are:
* - "Annulled" before the given time
* or
* - "Reviewed" or "Cancelled" and has been exported before the given time
* More precise is that the states are defined by having the semantic property
* "archivePurge.allow.Export" for batch orders.
*
* The list might contain keys of orders that are referenced by current or previous
* equipment object's context. Furthermore it is not checked if there are identified
* sublots for the orders.
*
* @param product the product of the order
* @param since the time since the order has been reviewed or canceled
* @return a list of keys of batch orders for the given product that are annulled
* or exported since the given days and that fulfills the conditions for purging
*/
public List<Long> getOrderKeysForPurge (final Part product, final Time since);
```

Method for workflow orders

```
/**
* The method for workflow orders that are:
* - "Annulled" before the given time
* or
* - "Finished, "Production-reviewed", "Reviewed" or "Canceled" since the given time
* - All of the orders it was appended to (i.e. its parent orders) are in a terminal
status
*   (there exists not a valid transition or it is in a logically terminal status)
*
* The list might contain keys of orders that are referenced by current or previous
* equipment object's context. Furthermore it is not checked if there are identified
* sublots for the orders.
*
* @param since the time since the order has been reviewed or canceled
* @return a list of keys of workflow orders that that are annulled or exported since
the
*  given days and that fulfills the conditions for purging
*/
public List<Long> getWorkflowKeysForPurge (final Time since);
```

Archive Orders with an Automated Job

With *IArchivePurgeProcessingService*, PharmaSuite provides a service and related methods for archiving and purging. In addition to the *ProcessOrderItem*, an *ArchivePurgeProcessingContext* must be provided. The context is used for each individual archive call.

To archive all of the given orders, proceed as follows:

Step 1

Create an *ArchivePurgeProcessingContext* for the automated job.

With the *ArchivePurgeProssingContext*, you can define a unique group identifier for your job. For example, you can use the starting timestamp as a part of the group identifier.

The group identifier is set in each archive and purge-related event created within the given context. So it is possible to view all events of your job. Furthermore, the execution type is defined as **Automated** in this case.

Example code

```
...
String theJobGroupIdentifier = <define a unique group identifier for your job>;
context = new ArchivePurgeProcessingContext(ArchivePurgeExecutionType.AUTOMATED,
          theJobGroupIdentifier, null);
 ...
```

Step 2

Archive all orders.

Having the keys of the orders (according to "Collecting Orders" (page 172)) and the *ArchivePurgeProssingContext*, the archiving process can be called. You need to define the output directory of the export for the orders and the appended workflows. The interface expects to get the *ProcessOrderItem*, i.e. the *ProcessOrder* must be loaded first to get the *ProcessOrderItem*. Since an automated background process has no user interface, the parameter for the *uiListener* is null in this case.

If you wish to evaluate the result of a single *archiveOrder* call and to handle it as needed, for example, send an email in case of an error.

Example code

```
...
for (Long keyOrder : keys) {
  ProcessOrder order = PCContext.getFunctions().getProcessOrderByKey(keyOrder);
  IMESArchivePurgeEvent result = archiveOrder(order, context);
  ...
  }

private IMESArchivePurgeEvent archiveOrder(final ProcessOrder order,
                              final ArchivePurgeProcessingContext context) {
  final IArchivePurgeProcessingService archivePurgeProcessingService =
                    ServiceFactory.getService(IArchivePurgeProcessingService.class);
  final ProcessOrderItem poi = (ProcessOrderItem)
                               order.getProcessOrderItems().firstElement();
  // define exportDirectory and exportDirectoryPathForAppendedWFs
  IMESArchivePurgeEvent result = archivePurgeProcessingService.archiveOrder(poi,
               context, null, exportDirectory, exportDirectoryPathForAppendedWFs);
  return result;
 }
...
```

Purge Orders with an Automated Job

The handling of purging is similar to the handling of archiving. The *IArchivePurgeProcessingService* provides a method for purging and in addition to the *ProcessOrderItem*, an *ArchivePurgeProcessingContext* must be provided.

To purge all of the given orders, proceed as follows:

Step 1

Create an *ArchivePurgeProcessingContext* for the automated job.

Example code

```
...
String theJobGroupIdentifier = <define a unique group identifier for your job>;
context = new ArchivePurgeProcessingContext(ArchivePurgeExecutionType.AUTOMATED,
          theJobGroupIdentifier, null);
 ...
```

Step 2

Purge all orders.

Having the keys of the orders (according to "Collecting Orders" (page 172)) and the *ArchivePurgeProssingContext*, purge can be done similar to the archiving process.

Example code

```
...
for (Long keyOrder : keys) {
  ProcessOrder order = PCContext.getFunctions().getProcessOrderByKey(keyOrder);
  IMESArchivePurgeEvent result = purgeOrder(order, context);
  ...
  }
private IMESArchivePurgeEvent purgeOrder(final ProcessOrder order,
                              final ArchivePurgeProcessingContext context) {
  final IArchivePurgeProcessingService archivePurgeProcessingService =
                     ServiceFactory.getService(IArchivePurgeProcessingService.class);
  final ProcessOrderItem poi = (ProcessOrderItem)
                               order.getProcessOrderItems().firstElement();
   IMESArchivePurgeEvent result = archivePurgeProcessingService.purgeOrder(poi,
context,
                                   null);
  return result;
}
...
```

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Archive	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting the Export for Archive Process	PSCEV2-GR010B-EN-E
A2	■ Purge	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting the Purge Process	PSCEV2-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Displaying Custom Columns in PharmaSuite's Audit Trail

Custom columns are not visible in the **Manage Audit Trail** task of the Production Management Client by default. Even if audit trail is enabled for a table, newly added columns are not visible in PharmaSuite. To make all columns visible, proceed as follows:

Step 1

Add custom columns to a FactoryTalk ProductionCentre object that is under audit trail in PharmaSuite

- For details, see [A1] (page 181)
- Example column: **myc_String** of **String** type in **Part** table

Step 2

For the new column, in the PharmaSuite SDK workspace, create a Java class as a subclass of the existing abstract model class

- Example code:

```
package com.rockwell.custmes.service;
import com.rockwell.mes.services.recipe.ifc.audittrail.AbstractPartRevisionInfoModel;
public class CtPartRevisionInfoModel extends AbstractPartRevisionInfoModel {
  public CustomPartRevisionInfoModel() {
    super();
  }
}
```

Step 3

For the new column, create a getter to the subclass

- Example code:

```
package com.rockwell.custmes.service;
import com.datasweep.compatibility.client.Part;
import com.rockwell.mes.services.recipe.ifc.audittrail.AbstractPartRevisionInfoModel;
public class CtPartRevisionInfoModel extends AbstractPartRevisionInfoModel {
  public CtPartRevisionInfoModel () {
    super();
  }
  public String getMycString() {
    try {
      return (String) ((Part) coreObject).getUDA("myc_String");
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
}
```

Step 4

Register the subclass in an XML configuration file similar to a service

- For details, see [A3] (page 181)
- Example code:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <PartRevisionInfoModel>
    <implementationClass>
      com.rockwell.custmes.service.CtPartRevisionInfoModel
    </implementationClass>
   </PartRevisionInfoModel>
</configuration>
```

Replace the path formatted in red with the complete path to your model class.

Step 5

Create a custom JAR file containing your class and your XML file

- For details, see [A4] (page 181)

Step 6

In Process Designer, create a library and import your JAR file

- For details, see [B1] (page 181)

Step 7

Create a data dictionary class for your model class

- For details, see [A2] (page 181)

TIP

You must copy all elements that are not hidden of the data dictionary of the superclass and add the elements to your custom columns.

- Example:
  1. Create the *com.rockwell.custmes.service.CtPartRevisionInfoModel* data dictionary class
  2. Copy the *com.rockwell.mes.services.recipe.impl.audittrail.PartRevisionInfoModel* elements to your class
  3. Add and configure the **mycString** data dictionary element

Step 8

Start the Production Management Client, run the **Manage Audit** task, and verify whether your custom data is visible.

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Add custom columns	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting and Adding Field Attributes ■ Adding Field Attributes to Application Table Objects	PSCEV2-GR010B-EN-E
A2	■ Data dictionary class ■ Create a DD class ■ Copy DD elements	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Adapting and Adding Field Attributes ■ Working with the Data Dictionary Manager	PSCEV2-GR010B-EN-E
A3	■ Register custom classes	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Managing Services ■ Configuring Service Implementations	PSCEV2-GR010B-EN-E
A4	■ PharmaSuite SDK setup	PharmaSuite SDK Readme	N/A

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

The following documents provide more details relevant to the implementation of the extension use case and are distributed with the FactoryTalk ProductionCentre installation.

No.	Keyword	Document Title	Part Number
B1	■ Create custom JARs	Process Designer Online Help: ■ Libraries	N/A

TIP

To access the "Process Designer Online Help", use the following syntax: *http://<MES-PS-HOST>:8080/PlantOperations/docs/help/pd/index.htm*, where <MES-PS-HOST> is the name of your PharmaSuite server. To view the online help, the Apache Tomcat of the FactoryTalk ProductionCentre installation must be running.

Chapter 15

Using a Locale with Unicode Fonts

PharmaSuite supports Unicode and thereby locales with Unicode Fonts. However, the system is not configured for Unicode fonts by default. To use Unicode fonts you must change the PharmaSuite configuration.By default PharmaSuite uses only physical fonts in four groups of font definition locations:

- **Group A**: Style sheets for the user interface of every PharmaSuite application. You can find them in **ProcessDesigner** as list objects with the "**Stylesheet**" substring in their names.
- **Group B:** The style sheet for reports (master recipe, master workflow, batch order, workflow, unit procedure, operation, phase). You can find it also in **ProcessDesigner** as list objects with the name "**PS-BatchReport-JasperStylesTemplate**".
- **Group C:** The style sheet definition of each label report design. You can find it in the header of the report design file.
- **Group D:** The font definition for the PDF/A export used when archiving batch orders and their sublot labels.

The default font of PharmaSuite is "**Arial**". To configure a Unicode font instead, proceed as follows:

1. Ensure that the language of the Locale you wish to use is installed on your computer.
   Open the **Control Panel**, select **Regional and Language Options**, navigate to the **Languages** tab, and click the **Details** button. The required language must be available in the **Default input language** list. You do not need to select the language.
2. Ensure that the required Unicode fonts are installed on your computer.
   Open the **Control Panel** and select **Fonts**. The required fonts must be available in the list.

3. **For Group A:**
   Duplicate the style sheet definitions (e.g. to '**cust_pmcStyleSheet**'), replace the standard fonts with the required Unicode fonts in the copied style sheets for the PharmaSuite applications (Production Management Client, Production Execution Client, Production Response Client, Data Manager, Recipe and Workflow Designer) and use the copied lists in your customer **Application** object (e.g. in the "**Forms/ApplicationStart_ProductionManagementClient/StyleSheet**" configuration key)..

   - For details, see [A1] (page 186)
   - Example configuration:
     ```
     * {
     font-family: "Arial Unicode MS";
     label-font-family: "Arial Unicode MS";
     header-fontfamily: "Arial Unicode MS";
     grid-fontfamily: "Arial Unicode MS";
     font-size: 11.0pt;
     background-color: rgb(255,255,255);
     disabled-color: rgb(255,255,255);
     }
     ```

   TIP
   Specify each font-family separately. Lists of font-families cannot be interpreted, see example.
   Example of incorrect use: `label-font-family: "Arial Unicode MS", Verdana, Arial;`

4. If required, assign your **Application** object to specific users.

   - For details, see [A2] (page 186)

5. **For Group B:**
   If required, change the configuration of the **"PS-BatchReport-JasperStylesTemplate"** style sheet of the master-recipe report, master workflow report, batch report, workflow report, unit procedure report, operation report, and phase report.

   - For details, see [A3] (page 186)
   - Example configuration:
     ```
     <?xml version="1.0" encoding="UTF-8"?>
     <!DOCTYPE jasperTemplate PUBLIC "-//JasperReports//DTD
     Template//EN"
     "http://jasperreports.sourceforge.net/dtds/jaspertemplate.dtd">
     <jasperTemplate>
     <style name="Root" isDefault="true" mode="Opaque"
     forecolor="#000000" backcolor="#FFFFFF" isBlankWhenNull="true"
     fontName="Arial Unicode MS" fontSize="9" isBold="false"
     isItalic="false" isUnderline="false" isStrikeThrough="false">
     <pen lineWidth="1.0" lineStyle="Solid" lineColor="#BFBFBF"/>
     }
     [...]
     ```

6. **For Group C:**
   If required, duplicate the report designs for labels (e.g. **custStationLabelRD**), replace them with the copy in the customer **Application** object and change the configuration in the copied report design (e.g. in "**Stations/stationLabel.reportDesign**" configuration key).

   - For details, see [A3] (page 186)
   - Example configuration:
     ```
     <?xml version="1.0" encoding="UTF-8"?>
     <jasperReport [...] name="MESStationLabelRD.1" [...]>
       <style name="Root" isDefault="true" fontName="Arial Unicode
     MS"/>
       <style name="Label" style="Root" hTextAlign="Left"
     hImageAlign="Left" vTextAlign="Middle" vImageAlign="Middle"
     fontSize="8" isBold="false"/>
       [...]
     ```

7. **For Group D:**
   To export reports to PDF/A or to archive orders and workflows, register the Unicode font as the Jasper Report extension.

   - For details, see [A5] (page 186)
   - Example: *config\rockwell\fonts.xml* configuration file in the *services-batchreport-ifc-<version>.jar.jar* library.

   TIP
   If you decide to use "Arial Unicode MS" as Unicode font in PharmaSuite, you can skip this step. This font is already configured in *config\rockwell\fonts.xml*

8. Start your application and change the **Locale**.

   - For details, see [A4] (page 186)

Reference Documents

The following documents provide more details relevant to the implementation of the extension use case and are available from the Rockwell Automation Download Site.

No.	Keyword	Document Title	Part Number
A1	■ Style sheet	PharmaSuite Technical Manual Configuration & Extension - Volume 1: Changing the General Appearance of PharmaSuite ■ Adapting Style Sheets	PSCEV1-GR010B-EN-E
A2	■ User and Application object	PharmaSuite Technical Manual Configuration & Extension - Volume 4: Managing Configurations PharmaSuite provides the PS Administration application to administer database objects such as access privileges, lists, applications, users, or user groups, see "Implementation Guide PS Administration" [A1] (page 187). Using Process Designer for these tasks is possible, but does not provide the full functional scope required for PharmaSuite.	PSCEV4-GR010B-EN-E
A3	■ Operational reports and labels ■ Batch report	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Changing or Adding New Reports ■ Fonts ■ Modifying and Adding Styles	PSCEV2-GR010B-EN-E
A4	■ Change Locale	PharmaSuite Technical Manual Configuration & Extension - Volume 2: Providing Localization to Support Different Locales ■ Changing the Locale	PSCEV2-GR010B-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

Reference Documents

The following documents are available from the Rockwell Automation Download Site.

No.	Document Title	Part Number
A1	PharmaSuite Implementation Guide PS Administration	PSAC-IN002B-EN-E
A2	PharmaSuite Technical Manual Developing System Building Blocks	PSBB-PM010A-EN-E

TIP

To access the Rockwell Automation Download Site, you need to acquire a user account from Rockwell Automation Sales or Support.

The following documents are distributed with the PharmaSuite installation.

No.	Document Title / Section
C1	PharmaSuite-related Java Documentation: Interfaces of PharmaSuite

TIP

To access the "PharmaSuite-related Java Documentation", use the following syntax: *http://<MES-PS-HOST>:8080/PharmaSuite/javadoc/*, where <MES-PS-HOST> is the name of your PharmaSuite server.

Appendix A

Revision History

The following tables describe the history of this document.

Changes related to the document:

Object	Description	Document
---	---	---

Changes related to "Introduction" (page 1):

Object	Description	Document
---	---	---

Changes related to "Extension and Naming Conventions" (page 7):

Object	Description	Document
---	---	---

Changes related to "What Are Extension Use Cases" (page 11):

Object	Description	Document
---	---	---

Changes related to "Processing Materials with a Second Potency" (page 13):

Object	Description	Document
---	---	---

Changes related to "Adding the Second Potency Attribute to ERP BOMs" (page 61):

Object	Description	Document
---	---	---

Changes related to "Adding the Second Potency Function to the Expression Editor" (page 71):

Object	Description	Document
---	---	---

Changes related to "Managing Batches in the ConditionallyReleased Status" (page 75):

Object	Description	Document
---	---	---

Changes related to "Adapting the Approval Workflow of Master Recipes" (page 83):

Object	Description	Document
---	---	---

Changes related to "Adding the Material Balance Section to the Batch Report" (page 107):

Object	Description	Document
Adding the Material Balance Section to the Batch Report (page 107)	Updated the description section with new areas.	1.0
Extending the Batch Record Schema (page 108)	Updated JAR file location. Added a Tip section. Updated code in the Step 2.	1.0
Extending IBatchProductionRecordDocumentWrapperExtension Interface (page 125)	Updated name of the chapter. Updated Step 1 section. Updated Step 2 section and the code.	1.0
Creating Fast Bean Readers (page 125)	Updated description section. Updated Step 1 section and the code. Updated Step 2 section and the code. Updated Step 3 section and the code. Added Step 4 section with the code.	1.0
Configuring Affected Services (page 132)	Updated code section from the Step 2.	1.0
Adapting the Report Design of the Batch Report (page 133)	Updated Step 3 section. Updated Step 4 section.	1.0

Changes related to "Displaying a Retest Date Column in the Overview Panel of the Exception Dashboard" (page 141):

Object	Description	Document
---	---	---

Changes related to Adding the Print FDA Report Function to PharmaSuite Clients (page 155):

Object	Description	Document
---	---	---

Changes related to "Adding the Print QA Report Function to Recipe and Workflow Designer" (page 167):

Object	Description	Document
---	---	---

Changes related to "Automatic Archiving and Purging of Specific Orders and Workflows" (page 171):

Object	Description	Document
---	---	---

Changes related to "Displaying Custom Columns in PharmaSuite's Audit Trail" (page 179):

Object	Description	Document
---	---	---

Changes related to "Using a Locale with Unicode Fonts" (page 183):

Object	Description	Document
Using a Locale with Unicode Fonts (page 183)	Added 4 groups of font definition locations: Group A, B, C, D. Updated Step 3 with instruction for Group A. Updated Step 5 with instruction for Group B and example configuration. Updated Step 6 with instruction for Group C and example configuration. Updated Step 7 with instruction for Group , reference link and example section. Added Tip section within Step 7. Added Step8 with a reference link.	1.0

Index