Oracle® Agile Product Lifecycle Management for Process

Extensibility Guide

Extensibility Pack 3.0

E37998-01

March 2013

ORACLE®

Oracle Agile Product Lifecycle Management for Process Extensibility Guide, Extensibility Pack 3.0

E37998-01

Contents

A Developer Information

Preface

The *Agile Product Lifecycle Management for Process Extensibility Guide* provides an overview of the numerous extensibility points in the Oracle Agile Product Lifecycle Management for Process suite. Extensibility points are areas in the application suite that can be used to extend the functionality of the product, typically through custom code and/or configuration changes.

Each extensibility point and any available reference implementations are described in the following chapters, along with the location of more detailed documentation.

This Preface contains these topics:

Audience

This guide is intended for technical implementers using Oracle Agile Product Lifecycle Management for Process. It can also be used by solution architects and business analysts who are responsible for designing and managing extension solutions. Information about administering the system resides in the *Oracle Agile Product Lifecycle Management for Process Administrator User Guide*.

Variability of Installations

Descriptions and illustrations of the Agile PLM for Process user interface included in this manual may not match your installation. The user interface of Agile PLM for Process applications and the features included can vary greatly depending on such variables as:

- Which applications your organization has purchased and installed
- Configuration settings that may turn features off or on
- Customization specific to your organization
- Security settings as they apply to the system and your user account

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at http://www.consumer.att.com/relay/tty/standard2.html. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process.

Related Documents

For more information, see the following documents in the Oracle Agile Product Lifecycle Management for Process Extensibility Pack documentation set:

- *Agile Product Lifecycle Management for Process Web Services Guide*
- *Agile Product Lifecycle Management for Process Data Administration Toolkit Guide*
- *Agile Product Lifecycle Management for Process Print Extensibility Guide*
- *Agile Product Lifecycle Management for Process Custom Section Denormalization Guide*
- *Agile Product Lifecycle Management for Process Extended Attribute Denormalization Guide*
- *Agile Product Lifecycle Management for Process Custom Report Configuration Guide*
- *Agile Product Lifecycle Management for Process Navigation Configuration Guide*
- *Agile Product Lifecycle Management for Process Extended Attribute Calculation Guide*
- *Agile Product Lifecycle Management for Process Product Quality Management Extensibility Guide*
- *Agile Product Lifecycle Management for Process Extensible Column Guide*

- *Agile Product Lifecycle Management for Process Release Notes*

Notes and other documentation are posted on Oracle Technology Network (OTN) at this location:

http://www.oracle.com/technetwork/documentation/agile-085940.html#plmprocess

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introducing Extensibility Points

The Extensibility Pack contains detailed extensibility point documentation, reference example source code, and compiled reference examples. Most extensibility points are in place in the core product suite release and available without requiring the Extensibility Pack. The documentation in this guide provides an overview of each extension point, a technical introduction, and describes any available reference examples. Each extensibility point also has more detailed documentation that provides technical implementation details to assist software developers.

Several extensibility points (such as the Web Services API, Custom Portal, Custom Section & Extended Attribute Denormalization) are larger in nature and are only available in the Extensibility Pack as deployable tools, web applications, database scripts, or utility classes.

Sample Code Disclaimer

Technical Requirements

The core requirements when developing Agile PLM for Process custom extensions are as follows:

- .NET 3.5 SP1
- Visual Studio

- Proficiency in C#
- XML
- SQL (T-SQL or PL/SQL)
- Working knowledge of XSLT and XSL-FO (for printing customization)

2

Extensibility Points

This chapter describes the extensibility points found in Extensibility Pack 3.0 of Oracle Agile Product Lifecycle Management for Process. Topics in this chapter include:

BOM Calc Extensions

The formulation specification's Bill Of Material calculation process (BOM Calc) and user interface can be extended to create custom calculation rules and user interaction.

Customers can create new calculation paths to handle a formulation specification's inputs, outputs, and steps, defining which fields should be editable, which fields should be locked down, and the calculation rules that will be used. Custom tags can be created for inputs, outputs, and/or steps, which can then be assigned in the UI as needed and guide the custom calculation rules.

Possible Uses

1. Create a formulation specification where no calculations are performed.
2. Create a BOM calculation path that extends inputs with certain tags. These tags can be used to extend calculations. For example, tag an input as a "protein" and always perform a certain set of calculations on that input.

Technical Overview

A custom BOM Calc implementation requires the creation of custom classes and user interface controls that define:

- **Calculation Path**—Defines the presentation of data and processing of events from the user interface. The path determines the BOM Calculator and specifies which custom tags should display for inputs, outputs, and steps.
- **BOM Calculator**—Manages the calculation logic.

Database scripts are used to set up the calculation path for user selection.

Technical Documentation

Refer to the **BomCalcDocumentation.doc** document, located in the Extensibility Pack Code\ReferenceImplementations\BomCalc\Documentation folder for more details.

Reference Implementation

Disclaimer: Reference implementations are provided to demonstrate implementation details and are not for use in production systems.

A reference BomCalcPath can be found in the Extensibility Pack under the folder ReferenceImplementations/BomCalc. This reference implementation does not do any calculations other then some multiplication of fields based on tag settings, and the events write out messages to a log file, with a name based on the specification number of the formulation specification being edited. The deployment files can be found in the Resources folder, while the source code can be found in the SourceCode/BomCalcExample folder.

Calculation Veto Plugin

Custom rules can be evaluated to determine if GSM specification and if PQM item calculations should occur. The `IsSpecCalculationAllowed` and `IsPQMCalculationAllowed` plugins are extension points available to all GSM specifications and PQM items that allows a custom class to be accessed when the specification calculation process runs. The custom class evaluates the current item and returns a true or false value to indicate if calculation should occur.

Possible Uses

1. Turn off specification/PQM item calculation once a specification has reached Approved status.

Technical Overview

The Calculation Veto plugin extensibility point will call the PluginExtensions framework to check if a Validate plugin is configured for this extension point in the CustomPluginExtensions.xml file. If no plugin is configured, a default plugin is used that simply returns true and gives permission to run calculation. The Calculation Veto Plugins are configured using the name `IsSpecCalculationAllowed` and `IsPQMCalculationAllowed`.

Example CustomPluginExtensions.xml configuration for Spec Veto plugin:

```
<ValidatePlugins configChildKey="name">
    <Plugin name="IsSpecCalculationAllowed"
    FactoryURL="Class:ReferencePlugins.ValidatePlugins.WorkflowTagBasedSpecCalculat
    ionDisablerFactory,ReferencePlugins$4" />
</ValidatePlugins>
```

Technical Documentation

Refer to the **PluginExtensions document**, located in the Extensibility Pack Code\ReferenceImplementations\PluginExtensions\Documentation folder for more details.

Available Reference Implementations

1. WorkflowTagBasedSpecCalculationDisabler is a reference implementation of a Validate plugin that examines a specification and turns off calculation if the specification status is Approved. The Approved status is determined by checking the workflow tags on the current status - if the IsApproved workflow tag (which has a BehaviorID of 4), then calculation is disabled. The BehaviorID is entered in the configuration file, so that it can easily be changed; for instance, adding other workflow tag behaviorID.

Source code: See the ValidatePlugins in Code\ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins for details.

Cost Extensions

Extensions are available around formulation costing in the following areas:

- Input Cost—Allows you to make adjustments the cost being pulled from the cost book.
- Displayed Calculated Cost—Allows you to make adjustments the calculated cost being displayed.
- Theoretical Cost Book Entry—Allows you to make adjustments to the total cost that is added to the theoretical cost book.

Possible Uses

1. Input Cost—The cost book contains the base cost for a raw material, but because this formulation is a dairy formulation increase raw material costs by 5%.
2. Displayed Calculated Cost—Any raw material that is used in amounts greater than 50lbs increase the cost by 2%.
3. Theoretical Cost Book Entry—Adjust the total theoretical cost for the output based on the percentages added to the "Labor Cost" extended attribute.

Technical Overview

Cost extensions use Format Plugins to return an adjusted cost numeric value as a string. Each plugin is configured in the config\Extensions\CustomPluginExtensions.xml file, using the following plugin names:

Cost Extension	Plugin name
Input Cost	FormulationInputCostBookPriceOverride
Displayed Calculated Cost	FormulationOutputTheoreticialCostPriceOverride
Theoretical Cost Book Entry	FormulationOutputPersistedTheoreticialCostPriceOverride

Technical Documentation

Refer to the PluginExtensions document, located in Code\ReferenceImplementations\PluginExtensions\Documentation for more details.

Custom Data Denormalization

Custom Data Denormalization is available via two denormalization techniques: Custom Section Denormalization and Extended Attribute Denormalization. The Extensibility Pack provides database scripts that are used to create new denormalized database tables and populate those tables with the denormalized data. Custom Section denormalization is configured in the Data Admin application of the PLM for Process suite, and allows for specifying how each custom section should be denormalized. Extended Attribute denormalization is not configured in the user interface; instead, all relevant extended attributes are automatically included in the process.

Custom sections and extended attributes can be denormalized in (near) real time, triggered by the Save events of business objects such as GSM specifications. See the *Agile Product Lifecycle Management for Process Custom Section Denormalization Guide* and the ***Agile Product Lifecycle Management for Process Extended Attribute Denormalization Guide*** for details.

Custom Sections

Custom Section Denormalization (CS Denorm) is a feature that provides the ability to convert the internal data storage of a custom section into data structures that are easier to understand and report against while providing improved query performance.

The CS Denorm process allows clients to select which custom sections (and which rows and columns) to denormalize and indicate how the target database tables should be set up. The CS Denorm process then reads this information, pulls the relevant Custom Section data from specifications (or other business objects), and populates that data into a single, simplified database table created solely for that custom section.

This approach provides customers with the following benefits:

1. Improve Performance—The denormalized data will be accessible via far fewer joins.
    a. Without Denormalization, querying for custom section data can involve over 20 database tables just for the custom section data.
    b. Using CS Denorm, simply querying the single new table provides most of that same data needed.
2. Lower Cost and Improve Delivery Time:—Since the denormalized data for a custom section is stored in a single table, the SQL needed is very easy to write. This will improve the time it takes to access the data and make the solution easier to maintain.

Extended Attributes

Extended Attribute Denormalization (EA Denorm) is a feature that provides the ability to convert the internal data storage of extended attributes into data structures that are easier to understand and report against while providing improved query performance.

The EA Denorm process pulls data for all activated (Active, Archive, and Inactive) extended attributes from specifications (or other business objects, such as sourcing approvals, NPD projects, etc.), and populates that data into specific denormalization tables. Extended attributes from custom sections are also included if they are marked as IsDistinct. The denormalization tables include additional information such as attribute IDs, custom section IDs, etc., that make the data easier to query against for reporting purposes.

Possible Uses

1. Reporting
2. Analytics

Technical Documentation

Detailed documentation explaining custom data denormalization can be found in the following guides:

Agile Product Lifecycle Management for Process Custom Section Denormalization Guide

Agile Product Lifecycle Management for Process Extended Attribute Denormalization Guide

Custom Portal

The Custom Portal is an extension of the Agile PLM for Process (PLMP) application suite. It allows customers to implement various integration solutions that leverage the PLMP data and capabilities without using the core application. Its primary usage is to provide a framework for searching, filtering, and displaying PLMP data, and gives solution implementers the ability to customize each of those aspects.

Custom Portal pages can be built to give users (who would not typically access PLMP) very specific access to certain data. Views of that data can be tailored to meet specific business needs, such as providing business partners with custom views into their specifications.

Possible Uses

1. Grant read only access to your individual plants. Plant users are a very different audience compared to the average GSM specification user. Plant users need to see a read only view of the entire finished good specification. This could be a combined view of data spanning attributes from the trade, nutrient profile, formulation, and raw materials.
2. Grant read only access to internal departments in a format they are used to seeing the data. For example, you can grant the Marketing department access to Product Fact Sheet reports for only approved finished goods. This would allow them to see nutritional fact panels and label claims pertaining to a particular finished good without granting them access to the entire nutrient profile and trade specification.

Technical Overview

Custom Portal is a web application that must be installed in an existing Agile PLM for Process environment. It contains portal management screens, page layout, security, and a pluggable framework that is used to develop custom search, filter, and display functionality. It relies on the Interfaces located in the CustomPortalInterfaces assembly, which define the class structure required when using the Search, Render, and Filter Plugins.

Client implementations that use the Agile PLM for Process Web Services API will require that the Web Services API is installed in an accessible environment.

Custom Portal may also host the client's own web application or assembly in which most of the customized plugins and other implementation code should be located.

Technical Documentation

Detailed documentation explaining the Custom Portal framework, including the administration of portal pages and views, the technical implementation requirements for extending the portal, and the existing reference implementation, can be found in the following location:

Code\Web\CustomPortal\Documentation\Custom Portal Implementation Guide.doc.

Available Reference Implementation

Disclaimer: Reference implementations are provided to demonstrate implementation details and are not for use in production systems.

MockCustomPortalPlugins is a reference implementation of a CustomPortal solution. It demonstrates the use of various search criteria and Plugins, and uses various web service calls and direct database queries to populate data that is then rendered as a PDF.

Source code: See Code\ReferenceImplementations\MockCustomPortalPlugins\.

eSignature Validate Plugin

If using the eSignature feature, and not using the out-of-the-box Passphrase based eSignature feature, this plugin can be called to perform custom eSignature authentication. The plugin receives the token passphrase (a string value) entered for eSignature authentication. The current user account is also available via the User property.

Technical Overview

The eSignature Validate plugin extensibility point will call the PluginExtensions framework to check if a Validate plugin is configured for this extension point in the CustomPluginExtensions.xml file. If no plugin is configured, a default plugin is used that simply returns true and gives permission to the eSignature entry.

The eSignature Validate plugin is configured using the name eSignatureValidatePlugin.

Example CustomPluginExtensions.xml configuration:

```
<ValidatePlugins configChildKey="name">
    <Plugin name="eSignatureValidatePlugin"
    FactoryURL="Class:Xeno.Prodika.PluginExtensions.Plugins.DefaultPlugins.DefaultV
    alidateTruePluginExtensionFactory,PluginExtensions" />
</ValidatePlugins>
```

Technical Documentation

Refer to the P**luginExtensions document**, located in Code\ReferenceImplementations\PluginExtensions\Documentation for more details.

Available Reference Implementations

While there are no specific reference implementations, any other validate plugin reference implementation can be reviewed for general guidelines.

Event Model

As specific events occur in GSM, SCRM, and PQM, their details are captured and recorded in a database table. Clients can watch for events added to this table to trigger some custom actions.

Each event captured may include the following information:

Event Type—The type of event that occurred (1: Create, 2: Save, 3: Workflow, 4: Copy)

Event Source—What caused the event (New issue of a specification, workflow transition, etc.)

Actor—User who performed the event

Time—Date and time stamp of when the event happened

Affected Object—Specification or object that was acted upon (Specification that was saved, specification that was copied, etc.)

Related Object—Related object when appropriate (Workflow step, smart issue request, specification ID, etc.)

Reason—Reason the action occurred when appropriate (Workflow comments, global succession reason for change, smart issue request that caused the change, etc.)

Possible Uses

The events captured can be the catalyst to a third party system action (email notifications, data comparisons, etc.).

1. Every time an ingredient specification is created, the Ingredient Manager is notified by email.
2. Every time a new specification issue is created a comparison will be performed. If any compliance data has changed, the facilities producing the product will be notified when opening the specification in custom portal.

Technical Overview

A feature configuration will determine if events will be logged.

GSM & SCRM—`Common.Auditing.LifecycleEvents.Enabled`

PQM—`PQM.Auditing.LifecycleEvents.Enabled`

The following tables are used to capture these events:

GSM & SCRM—`commonLifecycleEventLog`

PQM—`pqmLifecycleEventLog`

The table schemas are the same for both tables:

```
commonLifecycleEventLog (
pkid char(40) not null unique,
eventType int not null,                 -- create, save, workflow, etc.
eventSource varchar(50),                -- cause of the event
timestamp DateTime not null,            -- time of change
fkActor char(40) not null,              -- user making the changefkAffectedObject
char(40) not null,                      -- changed data object
reason nvarchar(256),                   -- user comments
fkRelatedObject char(40)                -- optional participant
)
```

The event type codes include:

Create—1

Save—2

Workflow—3

Copy—4

Extended Attribute Calculations

Calculated Extended Attributes allow you to create a read-only extended attribute that displays results of a calculation to the user. There are three types of calculated attributes: Numeric, Boolean and Text. The calculation, entered in the Data Admin user interface for Extended Attributes, must be written in JScript, and can access many predefined PLM for Process functions and properties that give access to specific data. Custom warning messages may be added during the calculation process for display to the user.

Clients wishing to have more control over calculations, consolidate their calculation logic, or access other data not directly available through JScript (and the predefined functions), may call out to custom classes from their scripts. The custom classes get executed and return a result back to the script. They may optionally receive parameter data from the script.

Technical Overview

Custom calculation classes, written in c#, are identified in the CustomerSettings.config file using a unique key for each class. This key is then referenced in the extended attribute's JScript calculation which calls out to the class and optionally passes data from the script to it.

Technical Documentation

Refer to the *Agile Product Lifecycle Management for Process Extended Attribute Calculation Guide*, located in the main documentation folder, for more details.

Available Reference Implementations

An example custom calculation class, Other Carbohydrates Calculator, demonstrates how a custom class can be used in calculations.

See the reference implementation in Code\ReferenceImplementations\CalculationExtensions\SourceCode for implementation details.

Extensible Columns

A few sections in PLM for Process allow you to add additional columns. These columns can display custom read only content. The following locations are available:

Trade > Packaging

Trade > Alternate Packaging

Trade > Material

Trade > Next Lower Level Items

Trade > Parent Items

Trade > Sourcing Approvals

Formulation > Inputs Grid

Formulation > Outputs Grid

Packaging > Packaging Sub Components

Packaging > Sourcing Approvals

Packaging > Printed Packaging

Printed Packaging > Packaging

Printed Packaging > Sourcing Approvals

Material > Sourcing Approvals

Equipment > Sourcing Approvals

Menu > Menu Item Build

Product > Sourcing Approvals

GSM > Cross References Grid

SCRM > Cross References Grid

PQM > Cross Reference Grid

Possible Uses

1. Add Qty Volume column to GSM formulation input BOM so that when the weight Qty column is adjusted the calculated volume is automatically shown.
2. Add theoretical nutrient "Sodium" or Extended Attribute like "% Meat" to the formulation output grid so that a specific theoretical target can be monitored when editing quantities.
3. Display the number of open quality issues found around the packaging specification included in the trade packaging BOM.

Technical Overview

For more information, refer to the *Agile Product Lifecycle Management for Process Extensible Column Guide*.

Formulation Percent Breakdown Classification Override Plugin

This extension point allows for the programmatic override of the percent breakdown classification on the formulation output popup. Out of the box the classification override can be declared by the formulator on the formulation output. This plugin allows you to calculate the classification override.

Technical Overview

The Formulation Percent Breakdown Classification Override plugin extensibility point will call the PluginExtensions framework to check if a Format plugin is configured for this extension point in the CustomPluginExtensions.xml file. If a custom plugin is configured, it must return a list of comma separated Formulation Classification PKIDs, which will then be listed by their names in the UI. If no plugin is configured, the overrides must be done manually in the UI.

The Formulation Percent Breakdown Classification Override plugin is configured using the name FormulationPercentBreakdownClassificationOverride.

Example CustomPluginExtensions.xml configuration:

```
< FormatPlugins configChildKey="name">
    <Plugin name="FormulationPercentBreakdownClassificationOverride"
    ignoreInheritFromPluginName="true"
    FactoryURL="Class:AcmePLM.FormatPlugins.CustomFormulationPercentBreakdownClassi
    ficationOverrideFactory,AcmePlugins" />
</ FormatPlugins>
```

Technical Documentation

Refer to the **PluginExtensions document**, located in Code\ReferenceImplementations\PluginExtensions\Documentation for more details.

Formulation Push Percent Breakdown Plugin

This extension point allows for the conditional enabling/disabling of the formulation output push of percent breakdown information to the material specification.

Technical Overview

The Formulation Push Percent Breakdown plugin extensibility point will call the PluginExtensions framework to check if a Validate plugin is configured for this extension point in the CustomPluginExtensions.xml file. If no plugin is configured, a default plugin is used that simply returns true and gives permission to push the percent breakdown.

The Formulation Push Percent Breakdown plugin is configured using the name FormulationPushPercentBreakdown.

Example CustomPluginExtensions.xml configuration:

```
<ValidatePlugins configChildKey="name">
    <Plugin name="FormulationPushPercentBreakdown"
    FactoryURL="Class:Xeno.Prodika.GSMLib.Security.Plugins.DefaultPushOutputBreakdo
    wnValidatePluginFactory,GSMLib" />
</ValidatePlugins>
```

Technical Documentation

Refer to the **PluginExtensions document**, located in Code\ReferenceImplementations\PluginExtensions\Documentation for more details.

Available Reference Implementations

While there are no specific reference implementations, any other validate plugin reference implementation can be reviewed for general guidelines.

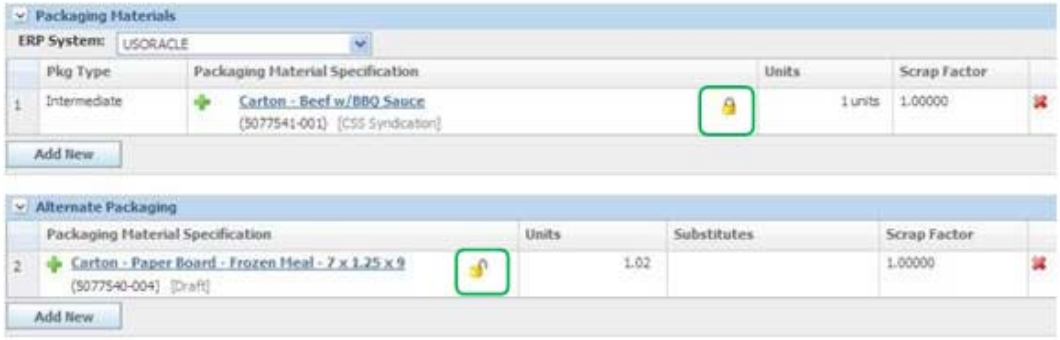
Get Latest Revision Extensibility

Get Latest Revision (GLR) is a feature that allows Agile PLM for Process specifications links to be automatically updated with newer Approved revisions. In the user interface, a lock icon next to a linked specification controls the GLR status for that item.

- When the icon is marked as locked ( ), the specification is tied to an exact specification/issue combination.
- When the icon is marked as unlocked ( ), however, the specification will be replaced with the latest revision/issue of that specification, based on defined behavior.

See the *Agile Product Lifecycle Management for Process Configuration Guide*, Table A-20 Custom Revisions for a list of locations where get latest revision is available. Out of the box the system will find the latest issue in a status that contains the isApproved tag. This extension allows you to change this behavior.

Figure 2–1 Locked and unlocked icons



The default behavior for retrieving the latest issue of a spec is to retrieve the latest Approved issue of that specification. If there are no newer approved issues of that specification, no updates are made.

> **Note:** The GLR feature actually identifies a specification as Approved if its current workflow status contains the IsApproved workflow tag.

This extension point allows for customizing the default behavior of Get Latest Revision, either by modifying the retrieval behavior to include specs in other workflow statuses, or using custom retrieval logic by implementing custom classes.

Possible Uses

1. The relationship will remain unlocked until the specification is in an archived state. This will allow a historical record of the final relationship that was active.
2. The relationship will only be updated if Allergens have not changed from the previous version of the specification.

Technical Overview

Get Latest Revision works in the UI by examining a spec link as it is loaded and, if unlocked, retrieving any newer revisions. A separate process runs on the Remoting Container on a regular basis to find any newer revisions and update the relevant spec links behind the scenes. To customize the GLR behavior, you will have to modify both functional areas.

Modifying the behavior to allow for different workflow statuses can generally be accomplished with no code changes, while implementing more complex customization will require a more involved implementation.

Technical Documentation

Refer to the **GetLatestRevision document**, located in Code\ReferenceImplementations\GetLatestRevision\Documentation for more details.

Available Reference Implementations

A reference implementation demonstrates how to prevent a Trade spec link from being updated if the parent Trade specification is in a status with one of the given workflow statuses (using workflow tag behaviorIDs).

See the reference implementations in Code\ReferenceImplementations\GetLatestRevision\SourceCode for implementation examples.

Label Claims Extensibility

Label Claim determination rules can be created and customized by using the Data Administration Toolkit. Label Claim formula calculation rules must be written in JScript, and return a boolean result indicating if the label claim is met. The formula rule calculation script can access various nutritional and reference data from the current business object via predefined properties.

Clients wishing to have more control over label claim determination rules, consolidate their calculation logic, or access other data not directly available through JScript (and the predefined functions), may call out to custom classes from their scripts. The custom classes get executed and return a result back to the script.

Technical Overview

A custom calculation class is identified in the CustomerSettings.config file with a unique key. This key is then referenced in the extended attribute's JScript calculation which calls out to the class and optionally passes data from the script to it.

Technical Documentation

Refer to the Label Claims Calculation document, located in Code\ReferenceImplementations\CalculationExtensions\Documentation for more details.

Available Reference Implementations

An example label claims calculation class, AlternateNutrientPer100gValueDynamicScriptMethod, demonstrates how a custom class can be used to return an alternate nutrient value.

See the reference implementation in Code\ReferenceImplementations\CalculationExtensions\SourceCode for implementation details.

Material Identity Plugins

Throughout the application suite there are many grids that display related specifications. For example, Formulation Input BOM and Trade Packaging BOM. The material identity extension allows for additional information to be displayed along with the standard specification information. Out-of-the-box, this extension shows the specification status in the majority of locations, however this plugin can be replaced with your own custom plugin displaying other information.

There are 32 unique specification identity extension points that can be leveraged to display additional specification related information. Each extension point is uniquely identified; for instance, the specifications listed in the trade specification's Next Lower Level Items grid are configured using the plugin name "TrdNextLowerLevelItemsIdentityPlugin". Each plugin can implement its own behavior, or it can call a common plugin. These plugins also return data for display in the print results, and can have the output returned for printing be different than the output returned for the user interface.

The material identity plugins are available in the following UI locations:

Table 2–1 Material identity plugin locations

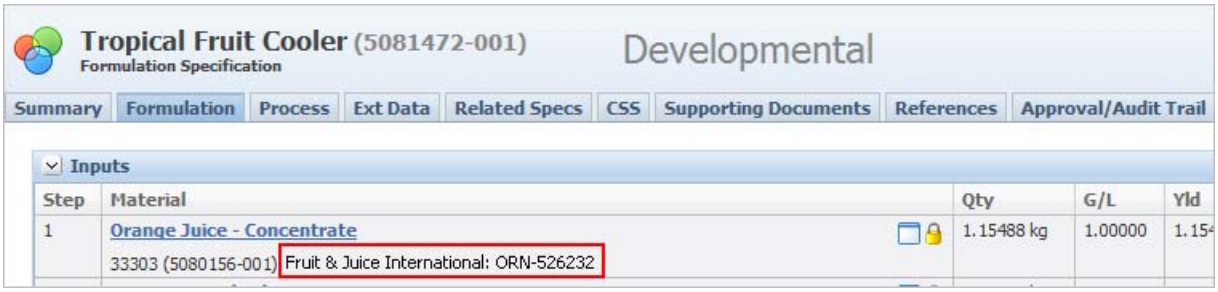
GSM Specifications	UI Area	Plugin Name
All Specifications	Associated Specification	AssociatedSpecsIdentityPlugin
	Master Specification	MasterSpecsIdentityPlugin
	Related Labeling	PackingRelatedLabelingIdentityPlugin
Equipment Specifications	Related Packaging	EquipmentRelatedPackagingIdentityPlugin
Formulation Specifications	Formulation Tab: Input Row	BOMInputItemPlugin
	Process Tab: Input Row	BOMInputItemPlugin
	Output PopUp: Composition Grid	BOMInputItemPlugin
	Alt Material	FrmAltMaterialIdentityPlugin
	Output Material	FrmOutputMaterialIdentityPlugin
Labeling Specifications	Related Packing	LabelingRelatedPackingIdentityPlugin
Material Specifications	Related Formulations	MaterialRelatedFormulationsIdentityPlugin
	Trade Specification Association	MaterialSpecTrdSpecAssociationIdentityPlugin
	Trade Specification Context Association	MaterialSpecTrdSpecContextAssociationIdentityPlugin
	Packaging Configuration	PackagingConfigIdentityPlugin
	Substitute Material	SubStituteMaterialIdentityPlugin

Table 2–1 Material identity plugin locations

GSM Specifications	UI Area	Plugin Name
Menu Item Specification	Alt Global Standard	AltGlobalStandardIdentityPlugin
	Global Standard	GlobalStandardIdentityPlugin
	Item Alternate	MenuItemAltIdentityPlugin
	Item Product	MenuItemProductIdentityPlugin
	Alternate Packaging	MenuItemRelatedAltPackagingIdentityPlugin
	Related Packaging	MenuItemRelatedPackagingIdentityPlugin
	Nutrient Profile	NutrientProfileIdentityPlugin
Nutrient Profiles	Related Specification	NutrientProfileRelatedSpecIdentityPlugin
Packaging Specifications	Packaging Configuration	PackagingConfigIdentityPlugin
	Printed Packaging Material	PackagingPrintedPkgMaterialIdentityPlugin
	Related Equipment	PackagingRelatedEquipmentIdentityPlugin
	Sub Component	PackagingSubComponentIdentityPlugin
	Substitute Material	SubStituteMaterialIdentityPlugin
Packing Configuration Specifications	Relates Specs	DeliveredMaterialPackingIdentityPlugin
Printed Packaging Specifications	Parent Packaging Material	PrintedPkgRelatedParentPkgIdentityPlugin
	Substitute Material	SubStituteMaterialIdentityPlugin
Product Specifications	Alternate Global Standard	AltGlobalStandardIdentityPlugin
	Global Standard	GlobalStandardIdentityPlugin
	Packaging Configuration	PackagingConfigIdentityPlugin
Trade Specifications	Nutrient Profile	NutrientProfileIdentityPlugin
	Alternate Packaging	TrdAlternatePackagingIdentityPlugin
	Material Specification Association	TrdMaterialSpecAssociationIdentityPlugin
	Material Specification Context Association	TrdMaterialSpecContextAssociationIdentityPlugin
	Next Lower Level Items	TrdNextLowerLevelItemsIdentityPlugin
	Packaging Material	TrdPackagingMaterialIdentityPlugin
	Parent Items	TrdParentItemsIdentityPlugin

Figure 2–2 Standard input material display

Figure 2–3 Extended input material display



Possible Uses

1. Display all cross references versus just the user's cross reference preference.
2. Display the Supplier Item #s from all sourcing approvals attached to the material specification.

Technical Overview

Each Material Identity extensibility point will call the Plugin Extensions framework to check if a format plugin is configured. Each plugin is identified by a specific unique name, which is then referenced in the CustomPluginExtensions.xml configuration file.

If a plugin is found for the given extensibility point name, the class specified in the configuration is loaded, passed the relevant data item (e.g., the related specification). The result of the plugin is then returned to the user interface.

If no plugin is found, it will use the out-of-the-box specification status implementation. To return a blank instead, use the EmptyIdentityPlugin (inheritFromPluginName="EmptyIdentityPlugin")Example CustomPluginExtensions.xml configuration for the Material Identity plugin:

```
<FormatPlugins configChildKey="name">
    <Plugin name="BOMInputItemPlugin"
    FactoryURL="Class:ReferencePlugins.FormatPlugins.BOMInputSupplierItemPluginFact
    ory,ReferencePlugins"  MaxSizeUI="40" MaxSizePrinting="100" />
</FormatPlugins>
```

The Material Identity plugins are implemented using a FormatPlugin, which provides for several capabilities:

- **MaxSizeUI**—Configuration setting tells the plugin what the maximum length for display should be.

- **MaxSizePrinting**—Configuration setting tells the plugin what the maximum length for printing display should be.
- **UseTextURL**—A boolean setting in the plugin to determine if the display should be replaced by some custom Javascript code.
- **GetTextURL**—A string value that is returned if UseTextURL returns true. This can contain html content, such as an anchor tag with a javascript pop-up code, for instance. A predefined pop-up is also available for use (and is demonstrated using the reference implementation below) to display content longer than the MaxSizeUI value.

See the **BOMInputSupplierItemPlugin** reference implementation and the code comments for details.

Technical Documentation

Refer to the **PluginExtensions document**, located in the Extensibility Pack Code\ReferenceImplementations\PluginExtensions\Documentation folder for more details.

Available Reference Implementations

1. BOMInputSupplierItemPlugin—Returns a list of the facility name and the supplier item number for each sourcing approval.
2. GSMSpecNumberFormatPluginExtension—Displays the specification number and the effective date.

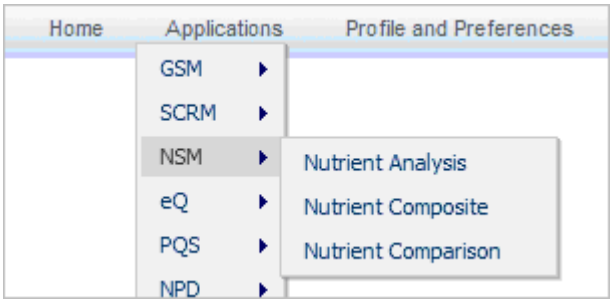
Source Code: See Code\ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins\FormatPlugins

Navigation Extensibility

You can extend the navigation panels throughout the application suite. There are three primary navigation areas:

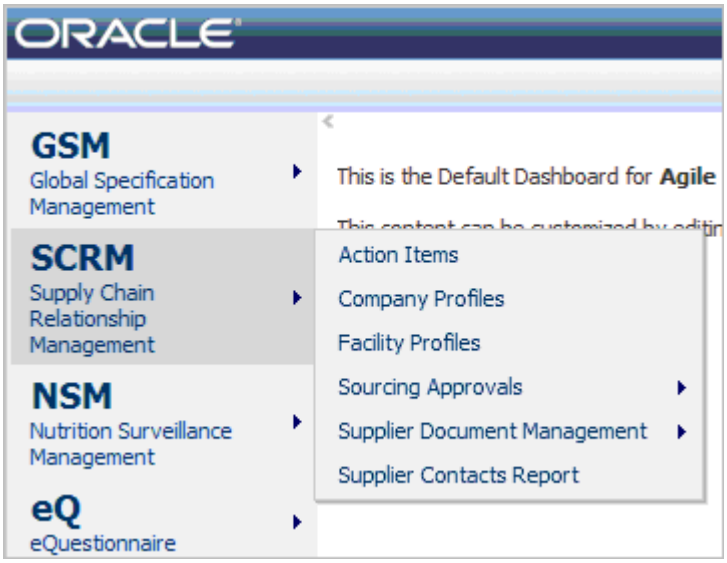
1. Platform Navigation—The navigation menu available in the top right of the browser window inside the suite header. This menu can be adjusted in the following ways:
   a. Add items
   b. Remove items
   c. Re-arrange items
   d. Apply visibility and security controls

Figure 2–4 Platform navigation



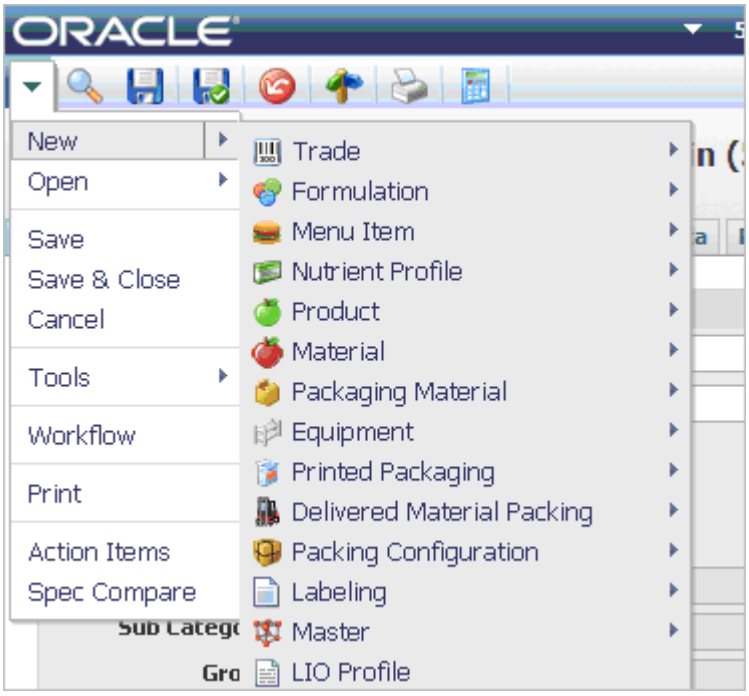
2. Portal Navigation—Available on the portal homepage listing in the left navigation panel. This menu can be adjusted in the following ways:
   a. Add items
   b. Remove items
   c. Re-arrange items
   d. Apply visibility and security controls

Figure 2–5 Portal navigation



3. Action Navigation—Available in the top left corner of all objects. This navigation also includes the quick access icons. This menu can be adjusted in the following ways:
   a. Add menu items
   b. Remove menu items
   c. Add quick access icons
   d. Remove quick access icons
   e. Adjust hot keys
   f. Re-arrange items
   g. Apply visibility and security controls

Figure 2–6 Action navigation



Possible Uses

1. Only users in the UGM user group of "Nutrition" are able to see the Nutrient Profiles link in GSM.
2. Add a quick access icon for a commonly used core action.
3. Add a link to an external system sending certain specification information to that system to direct the user's view.

Technical Overview

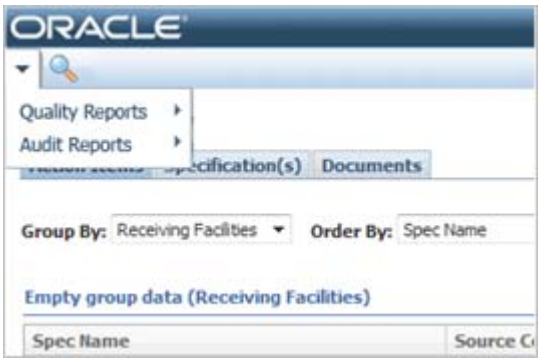
For more information, refer to the following the *Agile Product Lifecycle Management for Process Navigation Configuration Guide*. This guide can be found in: /Extensibility Pack 3.0 Documentation/ Oracle Agile Product Lifecycle Management for Process Navigation Configuration Guide.pdf

Navigation Extensibility: Supplier Portal

You can add navigation panels to Supplier Portal. There are two types of navigation you can add.

1. Primary Navigation—Primary navigation is available in the top left corner of all objects. This navigation also includes the quick access icons. This menu can be adjusted in the following ways:
   a. Add menu items
   b. Add quick access icons
   c. Apply visibility and security controls

Figure 2–7 Primary navigation



2. Object Navigation—Object navigation is available inline next to each specification. This menu can be adjusted in the following ways:
   a. Add menu items
   b. Apply visibility and security controls

Figure 2–8 Object navigation

Possible Uses

1. Provide links to other sites or portals that you offer or participate in with your suppliers.
2. Provide a link to a supplier performance report.
3. Provide a link to show audit results of your supplier's facilities over time.
4. Provide a link to a specific specification report from the specification & documents listing.

Technical Overview

For more information, refer to the following the *Agile Product Lifecycle Management for Process Navigation Configuration Guide*.

Notification Panel

A Notification Panel is available to display custom notification messages in GSM, SCRM and PQM when the user opens the object. Its content is populated through one or more Notification Plugins and configured in the NotificationPlugins node of the CustomPluginExtensions.xml file.

Notification Plugins are extension points used to return a list of messages. Multiple notification plugins can be configured and are chained together; each notification plugin is executed in the order found in the configuration file. Each notification plugin returns a list of strings, which is displayed to the user.

Figure 2–9 Sample notification panel

Possible Uses

1. Notify users when a specification contains specific allergens
2. Notify users when they are reading a specification that is not the approved issue
3. Notify the user when there are quality issues around a supplier or specification

Technical Overview

The NotificationPlugins extensibility point will call the PluginExtensions framework to check if any NotificationPlugins are configured for this extension point in the CustomPluginExtensions.xml file, and executes each notification plugin listed.

Example CustomPluginExtensions.xml configuration for the Material Identity plugin:

```
<NotificationPlugins configChildKey="name">
    <Plugin name="CustomNotificationsReaderPlugin"
    FactoryURL="Class:ReferencePlugins.NotificationPlugins.CustomNotificationsReade
    rPluginFactory,ReferencePlugins" UsedIn="PQMItem"/>
    <Plugin name="AllergenNotifierPlugin"
    FactoryURL="Class:ReferencePlugins.NotificationPlugins.AllergenNotifierPluginFa
    ctory,ReferencePlugins" UsedIn="GSMSpec"/>
    <Plugin name="SupplierQualityAlertPlugin"
    FactoryURL="Class:ReferencePlugins.NotificationPlugins.SupplierQualityAlertPlug
    inFa
    ctory,ReferencePlugins" UsedIn="SCRM"/>
</NotificationPlugins>
```

The notification plugins are called for each rendering of the page, regardless of the tab selected, or the edit/read mode. Creation of alternate display behavior, such as only showing the notifications while in Read mode, is the responsibility of the individual plugin. If no results are returned by any of the configured notification plugins, the notification panel is not displayed.

Custom Notification Table

A database table, CustomNotification, is available to store custom messages and then display them using a notification plugin. Entries in this table are not populated by any actions in Agile PLM for Process (PLMP); rather, the table is a storage location for other integration needs to store specific messages for an Agile PLMP object such as an ingredient specification.

These records can then be read by a notification plugin and displayed to the user as needed. A sample implementation (CustomNotificationsReaderPlugin) is included in the ReferencePlugins project.

CustomNotifications Table schema:

```
[customNotifications]
(
    [pkid] [char](40) NOT NULL,
    [fkOwner] [char](40) NOT NULL,
    [message] [nvarchar](2048) NOT NULL,
    [created] [datetime] NULL,
    [starts] [datetime] NULL,
    [expires] [datetime] NULL,
    [NotificationContext] [nvarchar](1024) NULL
)
```

- **pkid**—4 digit typeID + 36 character GUID: [Ex: '1149' + newId()]
- **fkOwner**—Represents the PKID of the relevant object, such as the PKID of the ingredient spec that the message is for
- **Message**—The message notification text
- **NotificationContext**—Unused

Technical Documentation

Refer to the **PluginExtensions** document, located in the Extensibility Pack Code\ReferenceImplementations\PluginExtensions\Documentation folder for more details.

Available Reference Implementations

1. AllergenNotifierPlugin—If the current object is a trade or material specification, a list of contained allergens is returned.
2. FormulationOutputsNotifierPlugin—If the current object is a formulation spec, displays a list of inputs and outputs that are not in a given status, such as approved.
3. CustomNotificationsReaderPlugin—Displays any entries for the current object in the CustomNotificaton database table.

Print Extensibility

Printing from GSM and Supplier Portal may be customized to meet various client needs. Clients may limit access to specific print templates, use custom data and field translations in the existing print templates, create their own print templates, configure what is pre-selected for users in the print dialog UI and use other printing engines (Oracle's BI Publisher, for instance) to render the results.

Possible Uses

1. Reformat the trade specification print out to use a different font or different spacing guidelines.
2. Remove certain sections from appearing in the material specification printout.
3. Create a Fact Panel report that is accessed from the trade specification. This report will include the fact panel data from the active nutrient profile and the potential label claims stored on the trade specification.
4. Every time a user prints a trade specification the packaging specifications, the custom sections and the active nutrient profile is included.
5. Provide a customized print view for your ingredient suppliers versus your packaging suppliers.

Technical Overview

See the *Agile Product Lifecycle Management for Process Print Extensibility Guide* for more information.

Quick Links

Methods are available to quickly launch a specification or object by using the objects system defined number. For example, to access a GSM object, the URL would be http://LOCALSITEURL/gsm/getSpecByNum.aspx?SpecNum=5084567-001 (5084567-001 would be the GSM specification number and issue number).

These methods are available for the following objects:

Table 2–2 Quick Links listing by application

Object	URL
GSM	http://LOCALSITEURL/gsm/getSpecByNum.aspx?SpecNum=xxxxxxx-xxx
SCRM Company	http://LOCALSITEURL/scrm/BaseForms/frmCompany.aspx?EntityID=xxxxxxx
SCRM Facility	http://LOCALSITEURL/scrm/BaseForms/frmFacility.aspx?EntityID=xxxxxxx
Sourcing Approval	http://LOCALSITEURL/scrm/BaseForms/frmSAC.aspx?EntityID=xxxxxxx
Sourcing Approval (Non-Spec)	http://LOCALSITEURL/scrm/BaseForms/frmNonSpecSAC.aspx?EntityID=xxxxxxx
NPD Projects	http://LOCALSITEURL/npd/MainPage/NPD.aspx?ContentKey=ProjectEditor&Load=xxxxxxx
NPD Strategic Briefs	http://LOCALSITEURL/npd/MainPage/NPD.aspx?ContentKey=StrategicBriefEditor&Load=xxxxxxx
PQM Issue, Action, Audit	http://LOCALSITEURL/pqm/getPQMByNumber.aspx?PQMItemNumber=xxxxxx
NSM Analysis	http://LOCALSITEURL/reg/NutritionSurveillance/NSM.aspx?ContentKey=NutrientAnalysis&Load=xxxxxxx
NSM Composite	http://LOCALSITEURL/reg/NutritionSurveillance/NSM.aspx?ContentKey=NutrientComposite&Load=xxxxxxx
Smart Issue	http://LOCALSITEURL/gsm/gsmextensions/SmartIssue/SmartIssue.aspx?ContentKey=SmartIssueRequest&Load=xxxxxxx
Global Succession	http://LOCALSITEURL/reg/MainPage/GlobalSuccession.aspx?ContentKey=SuccessionRequest&Load=xxxxxxx
DRL Document	http://LOCALSITEURL/drl/DRL.aspx?ContentKey=DrlDocument&DocumentId=xxxxxxx-xxx
LIO Profile	http://LOCALSITEURL/gsm/baseforms/frmLIOProfile.aspx?id=xxxxx
Component Catalog Term	http://LOCALSITEURL/reg/FIC/GetTermByNumber.aspx?TermNumber=xxxxxxx
eQuestionnaire	http://LOCALSITEURL/eq/MainPage/eq.aspx?ContentKey=ctlGetEntity&id=xxxxxxx

Section Level Editing

Custom validation rules can be created to control edit access of GSM sections. For example, a rule can be written to turn off editing of specific sections based on UGM user group and specification category, regardless of workflow status. When a section is read only, all editing methods will be hidden, for example, New buttons, Edit icons (pencils, deletes, etc.).

Refer to the **GSM Section IDs document**, located in the Extensibility Pack Code\ReferenceImplementations\PluginExtensions\Documentation folder for a list of secured section IDs.

Possible Uses

1. Only users in the UGM group "Nutrition" can edit the Fact Panel custom section on a nutrient profile.
2. When a specification is in an Approved state, only the Approved for Use in section is editable.
3. Only users in the UGM Group "Packaging" can edit the packaging sections of the trade specification.

Technical Overview

Section Level Editing rules are declared in the config\Extensions\SLESecurityExtension.config file. Security Handler classes are created that have access to the specification and the user information, and are used to determine if a particular GSM section can be edited.

Technical Documentation

Refer to the SLE Reference Implementation document, located in the Extensibility Pack Code\ReferenceImplementations\SectionLevelEditing\Documentation folder for more details.

Available Reference Implementations

Disclaimer: Reference implementations are provided to demonstrate implementation details and are not for use in production systems.

1. User Group and Specification Status

   The included reference implementation evaluates the specification's workflow status and user's UGM group membership.

   a. The LockByWorkflowTagUnlessInGroupSecurityHandler example security handler will lock down a section if a specification is in a certain workflow status (as indicated by the workflow tag on the status), unless the user is in a certain User Group.

      – If the spec editor is in the UGM group of "Spec Admin" then all sections on the specification can be edited.

      Source Code:

      ```
      \ReferenceImplementations\SectionLevelEditing\SourceCode\ReferenceSectionLe
      velEditingExtensions\ReferenceSLEHandlers\LockByWorkflowTagUnlessInGroupSec
      ```

```
urityHandlerFactory.cs
```

2. Configurable Handler

   This example demonstrates how to parse configurable information to the handler from the SLESecurityExtension.config.

   Source Code:

   ```
   \ReferenceImplementations\SectionLevelEditing\SourceCode\ReferenceSectionLevelE
   ditingExtensions\ReferenceSLEHandlers\ConfigurableSLESecurityHandler.cs
   ```

Specification Veto Plugin

Custom security rules can be evaluated when determining GSM specification read permissions. The Specification Veto Plugin is an extension point available to all GSM specifications that allows a custom class to be accessed when the user opens a specification. The custom class evaluates the current specification and returns a true or false value giving read access to the specification or not.

Possible Uses

1. If the user does not have read access to every specification in the trade's hierarchy, the user is not allowed to read the trade specification.
2. If the user does not have read access to all inputs used on the formulation specification, the user is not allowed to read the formulation.

Technical Overview

The Specification Veto plugin extensibility point will call the PluginExtensions framework to check if a Validate plugin is configured for this extension point in the CustomPluginExtensions.xml file. If no plugin is configured, a default plugin is used that simply returns true and gives read access.

The Specification Veto Plugin is configured using the name HasSpecVisibilityPlugin.

Example CustomPluginExtensions.xml configuration for Spec Veto plugin:

```
<ValidatePlugins configChildKey="name">
    <Plugin name="HasSpecVisibilityPlugin"
    FactoryURL="Class:ReferencePlugins.ValidatePlugins.ValidateTradeAccessPluginFac
    tory,ReferencePlugins" />
</ValidatePlugins>
```

If Business Unit (BU) security is enabled, the user's business unit permissions are evaluated prior to calling the HasSpecVisibility plugin. If BU security is not enabled, the HasSpecVisibility plugin is called immediately and its results determine read permission to that specification. The specification and the current user data objects are passed to the plugin.

Technical Documentation

Refer to the **PluginExtensions document**, located in the Extensibility Pack Code\ReferenceImplementations\PluginExtensions\Documentation folder for more details.

Available Reference Implementations

Disclaimer: Reference implementations are provided to demonstrate implementation details and are not for use in production systems.

1. ValidateTradeAccessPlugin is a reference implementation of a Validate Plugin that examines trade specifications and only allows access if the user has read permission to each lower level trade specification.

Source code: See the ValidatePlugins in Code\ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins for details.

PQM Veto Plugins

Basic PQM read, write, and workflow permissions for issues, actions, and Aaudits are based on the workflow templates set up in Workflow Administration. PQM adds two useful extensibility points to further customize Read and Write permissions on a PQM item.

Custom Read Permission

A Validate Plugin class can be created to extend the Read permission logic of a PQM item, if desired.

To customize the Read permission checks for PQM, create a new Validate Plugin and add an entry into the CustomPluginExtensions.xml file in config\Extensions, in the ValidatePlugins node, using the plugin name **"HasPQMReadPermissionPlugin"**, like so:

```
<Plugin name="HasPQMReadPermissionPlugin"
ignoreInheritFromPluginName="true"
FactoryURL="{Your custom class using ObjectLoaderURL syntax}" />
```

Custom Write Permission

A Validate Plugin class can be created to extend the Write permission logic of a PQM item, if desired.

To customize the Write permission checks for PQM, create a new Validate Plugin and add an entry into the CustomPluginExtensions.xml file in config\Extensions, in the ValidatePlugins node, using the plugin name **"HasPQMWritePermissionPlugin"**, like so:

```
<Plugin name="HasPQMWritePermissionPlugin"
ignoreInheritFromPluginName="true"
FactoryURL="{Your custom class using ObjectLoaderURL syntax}" />
```

Technical Documentation

To learn more about Validate Plugins, see the PluginExtensions document in the \ReferenceImplementations\PluginExtensions\Documentation folder. Various reference implementations of Validate Plugins can be found in the \ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins\ValidatePlugins folder.

Validation Framework

The validation framework allows you to configure custom validation rules to specific UI events in the system. For example, when a user selects the Save action button on a specification, code can be put in place to make sure specific required fields are properly filled out. If any required fields are left blank, an error message can be displayed preventing the user from saving the specification until all of the data is provided.

The following objects are tied to the validation framework:

- GSM specifications and templates
- Smart issue requests
- Testing protocols
- SCRM companies, facilities, and sourcing approvals
- eQuestionnaires
- Custom section templates
- NPD projects, activities, innovation sales pipeline, and strategic briefs
- PQM issues, actions, and audits

To see a detailed listing of events, type IDs, validation target objects and context objects refer to ReferenceImplementations/Validation/Documentation/Validation Objects.xls.

Possible Uses

1. Make sure all data has been added to the specification or object before it is saved or transitioned to a new workflow state. This includes custom data. For example, a nutrient profile cannot be approved until the custom section: NLEA Fact Panel has been added.
2. A trade specification cannot be approved until all packaging specifications attached to the trade specification are in an approved state.
3. A sourcing approval cannot be approved until the specification it is tied to is in an approved state.
4. A user cannot transition an issue of a specification to an approved state if a previous issue of that specification is in an approved state.
5. A user cannot create an issue of a specification that is in a non-approved state.

Technical Overview

Validation logic is declared in a configuration file (Config\Extensions\ValidationSettings.xml) and specified by using predefined validation classes or creating custom validation classes.

Validators are classes that can examine the current object and execute validation rules against it. The result of a validator is true for a successful validation and false for a failed validation check. Error messages may be added based on the validation, which are then displayed to the user.

Example rule in ValidationSettings.xml:

```
<ValidationRules>
    <!-- Example Ingredient Spec save validation requires Cross References (aka
```

```
    Legacy profiles) -->
    <rule type="1004">
        <condition event="save">
            <if type="ReflectiveRequiredValidator" property="LegacyProfiles" />
        </condition>
    </rule>
</ValidationRules>
```

Technical Documentation

Detailed technical training of the Validation Framework is available in the Extensibility Pack in Code\ReferenceImplementations\Validation\Documentation\.

Available Reference Implementations

Disclaimer: Reference implementations are provided to demonstrate implementation details and are not for use in production systems.

Several reference implementations are available in the Code\ReferenceImplementations\Validation\SourceCode\ReferenceValidation project, including:

1. TIP Value Validator—Evaluates a given property of a TIP in a given namespace.
2. CustomDataValidator—Performs validation on the existence of custom sections, their rows or columns, and on the existence of values on extended attributes. This validator is a good reference implementation of using the CustomDataFacade utility class to perform specific validations on custom sections and extended attributes.

Other reference implementations and examples of all predefined validators are found in the Validation Framework Training presentation and source code, located in the Extensibility Pack in Code\ReferenceImplementations\Validation\Documentation\.

Workflow Actions and Guard Conditions

A workflow action is an extension point that triggers the execution of custom classes when a workflow transition occurs. A guard condition is an extensibility point that helps determine if a workflow transition can occur.

Workflow actions and workflow guard conditions are assignable to workflow transitions in WFA. Different workflow actions and guard conditions are available in WFA for GSM, SCRM, PQM, and CSS workflows.

Possible Uses

1. Every time a sourcing approval reaches the approved state, specific data from the sourcing approval can be sent to a third party system.

Technical Overview

Workflow actions and workflow guard conditions are created as custom classes, packaged into a DLL, and added to the relevant web applications (web\gsm\bin, web\scrm\bin, and web\ugm\bin).

They must be configured in the config\Extensions\CustomWFAExtensionsConfig.xml file to be made available for assignment in WFA.

Workflow actions can perform custom activities, such as sending an email, logging information, etc., and have access to the item being workflowed.

Guard conditions can evaluate the item being workflowed, determine if the workflow transition should occur, and return a true or false result accordingly. Additionally, they can add error messages which will be displayed to the user in the workflow pop-up.

Technical Documentation

See the \Code\ReferenceImplementations\WorkflowActions\Documentation folder located in the Extensibility Pack for more details.

Available Reference Implementations

Disclaimer: Reference implementations are provided to demonstrate implementation details and are not for use in production systems.

Several reference implementations are available in the Code\ReferenceImplementations\WorkflowActions\SourceCode\ReferenceWorkflows and Code\ReferenceImplementations\GuardConditions\SourceCode\RefGuardConditions projects, including:

1. SpecStatusChangeLogger—Logs workflow status changes, along with specification identifier information, to a file.

Workflow Email Extensions

PLM for Process provides various automated email notifications when certain business objects move from one workflow status to another. Emails are sent to owners of a certain object informing them that the item is now in their action items listing, sent to users asking them to sign off on the item, or as a simple notification that the item moved from one status to another. The email recipients are specified in the WFA application, using the Owners, Notifications, and Signature Request grids.

These emails can be customized for GSM, SCRM, and PQM workflows. There are two ways emails can be customized. You can change the single translation that is offered out of the box. Or you can go one step further by creating your own format plugin and actually send different email messages based on certain conditions. For example, when a packaging specification is going from Draft to Review send "Message A", when it's going from Review to Approved send "Message B".

You can even go a step further and base which email is sent by which tag a status contains. This allows you to create email templates and the WFA business administrator can select when to send them. For example, send "Message A" when entering a step containing the "Development" tag. Send "Message B" when entering a step containing the "ManagementReview" tag.

Technical Overview

Clients wishing greater control over the email contents can choose to use FormatPlugin classes to replace the email subject and/or body contents, rather than just using the translations. A format plugin can be specified for each email type and for the email subject and email body. This allows clients to keep certain email behaviors as is, and just customize what is needed.

Technical Documentation

See the ReferenceImplementations\EmailExtensions\Documentation\Email Extensions.doc file for details.

A

Developer Information

PLM4PExtensionUtils Developer Utility Library

PLM4PExtensionUtils is a library that provides classes to assist external developers with Agile PLM for Process extensibility development. Custom Validators, Workflow Actions and Workflow Guard Conditions, Plugins, Calculation Extensions, and other extensibility points can leverage these utility classes by referencing the PLM4PExtensionUtils.dll.

The following utility classes are available:

- SpecPermissionEvaluator—Provides specification related security permission methods
- SpecWorkflowTagEvaluator—Provides workflow status related methods for GSM specifications
- SCRMWorkflowTagEvaluator—Provides workflow status related methods for SCRM sourcing approvals
- FormulationStepsRetriever—Retrieves a sorted list of formulation steps for a given formulation specification
- CustomDataFacade—A class that provides simplified access to extended attributes and custom sections.

Detailed documentation and the PLM4PExtensionUtils.dll are available in the Extension Utilities document in the Utilities\PLM4PExtensionUtils folder.

Several reference implementations, such as the ValidateTradeAccessPlugin in ReferencePlugins, already leverage the various classes available in this dll.

Object Loader URLs

Object Loader URLs are classpaths that are used to dynamically load objects. They are used to declare the protocol to use when loading the class, the class path, and optionally any parameters to pass to the class.

Format

```
[Protocol] : [Path] $ [{parameter1} | { parameter2}|…]
```

- Protocol—Examples are "Class" and “Singleton"
- Path—The fully qualified class name, including the package name. For example `"Xeno.Prodika.SecurityModel.Contextual.UserRoleBasedSecurityPluginFactory,ProdikaLib"` where ProdikaLib is the name of the package (.dll file).
- Parameters—If the class implements the ITakesParameters interface, the parameter list, separated by pipes (|), is available to the class. See Passing Parameters in the ObjectLoaderURL below.

When loading an object, the loader first inspects the Protocol and using lazy loading, determines an appropriate protocol handler based on this protocol’s name. The "Class" protocol may refer to a class that accepts parameters during instantiation which are defined after a "$" and delimited by "|"s (pipes).

Common Usage

The most common usage of this class is in configuration files. Often a factory class is supplied in a configuration and the Object Loader bootstraps the factory, which in turn facilitates the use of the rest of the implementation. These implementations are easily swapped by simply providing a different factory in the configuration.

Example

```
Class:Xeno.Prodika.Portal.WebUI.Util.Security.UserPropertyBasedSecurityPluginFactory,ProdikaLib$NPD
```

"Class" is the protocol, "NPD" is a parameter, and the rest of the string between the ":" and the "$" is the path as defined by the protocol. In this case, it is the class path of the object that is to be instantiated.

Passing Parameters in the ObjectLoaderURL

Implementing the `Xeno.Prodika.Common.ITakesParameters` interface (from ProdikaCommon.dll) by the Factory class allows the passing in of parameters in the ObjectLoaderURL. Its method setParams is called, with the StringSplitter input parameter containing the arguments in the ObjectLoaderURL. This allows the same factory class to be used for multiple situations, such as passing in the desired workflow statuses as a parameter.

For an example of a class that implements the ITakesParameters interface, see the WorkflowTagBasedSpecCalculationDisablerFactory in ReferencePlugins

```
public class WorkflowTagBasedSpecCalculationDisablerFactory:
IValidatePluginExtensionFactory, ITakesParameters
{
    private IList<int> _behaviorIDs ;
    public IValidatePlugin Create()
    {
        return new WorkflowTagBasedSpecCalculationDisablerPlugin(_behaviorIDs);
    }
    public void setParams(StringSplitter splitter)
    {
        _behaviorIDs = new List<int>();
        Assert.True(splitter.hasMoreTokens(),
    "WorkflowTagBasedSpecCalculationDisablerFactory must pass a comma delimited
    list of workflow behavior IDs assigned to Workflow Steps that should not have
    Calculation occur." );
        string[] tags = splitter.nextToken().Split(',');
        foreach(string tag in tags)
        {
            _behaviorIDs.Add(int.Parse(tag));
        }
    }
}
```

Object and Data Schema Documentation

When writing custom reports or SQL queries against the PLM4P database, or writing various extensibility points such as Validators, Workflow Actions, and more, developers must be able to navigate and understand the internal data and object structures they will be interacting with. The Object and Database Schema document (available via the index.html file in the DatabaseAndObjectSchema folder) is a catalog of the Agile PLM for Process database tables and data object classes. The tool allows SQL developers and .NET developers to inspect the internal Agile PLM for Process database and data object hierarchies using HTML files. It provides a listing of all database tables and their corresponding data object classes, categorized by the application and the high level business objects (e.g., GSM -> Packaging Specification).

Database Tables

Each database table listed describes its database columns and its various relationships to and from other tables. Clicking on a relationship link will display the related table and maintain a breadcrumb trail of the relationship. A “Show SQL” link can be used to show SQL code that can be used to join the tables defined in the breadcrumb trail.

For instance, to get the trade type name of a trade specification, (starting in All Applications), click **GSM**, then **Trade Specification**, then **gsmTradeType**, then **gsmTradeTypeMML**, where the Name column can be found. The breadcrumb trail shows the following: Applications > GSM > gsmBaseTradeSpec > gsmTradeType > gsmTradeTypeMML.

Clicking **Show SQL** displays the following results:

```
SELECT * FROM gsmBaseTradeSpec t1
INNER JOIN gsmTradeType t2 ON t1.fkTradeType = t2.pkid
INNER JOIN gsmTradeTypeMML t3 ON t2.pkid = t3.fkTradeType
```

Additionally, since each database table is related to a specific .NET class, a link to its corresponding data object is available.

Data Objects

Each data object listed describes its implemented interfaces, simple/primitive properties, object properties, and collection properties. The PLM4P internal data objects, however, can only be accessed by their immediate interface.

For instance, the AdditiveContainedDO data object can be accessed by the IAdditiveContainedDO interface. Since Additives may be found on multiple specification types, the AdditiveContainedDO data object has a property named Parent, which is of type IBaseSpec, the common interface of all specification types.

To access the trade type information for the data object, (starting in All Applications), click **GSM**, then **Trade Specification**, then the **Data-Object IGSMTradeSpecDO** link, then **ITradeType**, then **ITradeTypeMML**, where the Name property can be found. If trying to access this data in code, the property can be accessed like so: string tradeName = ((IGSMTradeSpecDO) baseSpec).TradeType.TradeTypeMML.Name;

Each data object also links back to its related database table.

Other Available Data

The topmost navigation provides several other useful listings:

- **All Applications**—The front page of this document set and provides an alphabetized list of all application groupings of the highest level business objects. You can navigate from any of the listed objects to all of their constituent tables via their relationships.
- **All Tables**—An alphabetical listing of all of the documented tables.
- **All Columns**—An alphabetical index of all of the Agile PLM4P fields (columns and join-tables) with their descriptions. This index can be especially useful when searching for a table when all that is known is a keyword/concept. Columns are listed in the form of "Columnname.Tablename: Description" (or "JoinTableName.MasterTableName: Description" for join-relationships). The hyperlink navigates to the table where that relationship is defined, and down to the specific section where that column is listed.
- **All Data-Objects**—An alphabetical listing of all of the documented data-object/classes.
- **All Data-Object Properties**—An alphabetical listing of all of the documented data-object properties with their descriptions. This index can be especially useful when searching for a data-object when all that is known is a keyword/concept. Properties are listed in the form of "Classname.Property: Description". The hyperlink navigates to the data-object where that relationship is defined, and down to the specific section where that property is listed.
- **All Views**—An alphabetical listing of all of the Agile PLM4P views.

Additional Details

Agile PLM for Process uses a custom Object Relational Mapping layer, which defines how the data objects used in the application are tied to the database tables. Each class relates to a database table. Each row in the table represents a single object instance. The OR Mapping relationships are stored in the database. This provides a way to understand the database table relationships by examining the OR Mapping tables.

PKIDs—Primary Key Identifiers

All tables entries have a uniquely typed PKID by prefixing a 4 digit type id onto the front of a 36 character GUID (or 6 character GUID in some cases).

PKID = 4 Digit Type ID + GUID (Globally Unique Identifier)

The TypeID can help navigate the database structure to locate where an identifier can be found. For example, the SpecSummary table maintains a SpecID column, which could point to one of many different specification tables. Extracting the typeID value from the SpecID foreign key will tell us which table.

OR Metadata Tables

The ORClassMetaInfo table tells us which database table (and therefore which class) the TypeID represents:

```
SELECT * FROM orclassmetainfo WHERE type=1004 OR type = 2147;
```

Tablename	Classname	Type
MaterialSpec	IngredientSpecification	1004
gsmBaseTradeSpec	GSMTradeSpecDO	2147

We can now see that a PKID starting with:

TypeID 1004 is a material specification, the table is MaterialSpec, and the class is IngredientSpecification

TypeID 2147 is a trade specification, the table is gsmBaseTradeSpec, and the class is GSMTradeSpecDO

- **ORClassMetaInfo**—Tells which database table the TypeID represents.
- **ORObjectPropertyMetaInfo** —Tells the related objects for a table, for single and multi-value secondary object references. To find related tables based on a specific table look at:

  ```
  SELECT * FROM orpropertymetainfo WHERE fkORClassMetaInfo = (SELECT pkid FROM
  orclassmetainfo WHERE tablename = '<yourtablename>')
  ```

- **ORPropertyMetaInfo** —Simple and foreign-key fields.

Language Aware Tables

To support multiple languages, all translatable text is stored in language aware tables. These tables will always contain the column, langID, which is a reference to a predefined language in the SupportedLanguages table. Many of the language aware tables also contain "ML" as part of the table name. For example, gsmShortNameML contains the text for the specification's short name. The default value for langID is 0 (English). There should always be a value in the language aware tables with langID=0. It is important to specify the langID when writing direct SQL or you may end up with more results than desired. For example:

```
Select
    spec.SpecNumber,
    specname.name,
    shortname.name shortname
From specSummary spec
    inner join SpecSummaryName specname on specname.fkSpecsummary = spec.PKID and
    specname.langid = 0
    inner join gsmShortNameML shortname on shortname.fkSpecSummary = spec.PKID and
    shortname.langid = 0
where
    specname.name like '%test%';
```