Oracle® Agile Product Lifecycle Management for Process

Extensibility Overview Guide

Release 6.2.3.x

E97597-02

December 2018

ORACLE

Oracle Agile Product Lifecycle Management for Process Extensibility Guide, Release 6.2.3.x

E97597-02

Contents

Preface ix

Audience ix
Variability of Installations ix
Documentation Accessibility x
Software Availability x
Related Documents x
Conventions xi

1 Introducing Extensibility Points

Sample Code Disclaimer 1-1
Technical Requirements 1-1

2 Extensibility Points

BOM Calc Extensions 2-3
Possible Uses 2-3
Technical Overview 2-3
Technical Documentation 2-3
Reference Implementation 2-3
Calculation Veto Plugin 2-4
Possible Uses 2-4
Technical Overview 2-4
Technical Documentation 2-4
Available Reference Implementations 2-4
Clone Extensibility 2-5
Clone Event Types 2-5
Possible Uses 2-5
Technical Documentation 2-5
Cost Extensions 2-6
Possible Uses 2-6
Technical Overview 2-6
Technical Documentation 2-7
Available Reference Implementations 2-7
Custom Data Denormalization 2-8
Custom Sections 2-8
Extended Attributes 2-8

Possible Uses 2-9
Technical Documentation 2-9
Custom Portal 2-10
Possible Uses 2-10
Technical Overview 2-10
Technical Documentation 2-10
Available Reference Implementation 2-10
Email Extensions 2-12
Technical Overview 2-12
Technical Documentation 2-12
eSignature Validate Plugin 2-13
Technical Overview 2-13
Technical Documentation 2-13
Available Reference Implementations 2-13
Event Framework 2-14
Technical Overview 2-14
Possible Uses 2-14
Technical Documentation 2-14
Available Reference Implementations 2-14
Legacy Event Model Tables 2-15
Tips: 2-15
Data Captured 2-15
Technical Overview 2-15
Table of Logged Events 2-16
Available Event Subscribers 2-17
Available Event Subscriber Filters 2-17
Available Reference Implementations 2-17
Extended Attribute Calculations 2-18
Technical Overview 2-18
Technical Documentation 2-18
Available Reference Implementations 2-18
Extensible Columns 2-19
Possible Uses 2-19
Technical Overview 2-19
Available Reference Implementations 2-19
FlexSync Foundation 2-21
Overview 2-21
Technical Documentation 2-21
Formulation Output Naming Plugins 2-22
Possible Uses 2-22
Technical Documentation 2-22
Available Reference Implementations 2-23
Formulation Percent Breakdown Classification Override Plugin 2-24
Technical Overview 2-24
Technical Documentation 2-24
Formulation Push Percent Breakdown Plugin 2-25
Technical Overview 2-25

Technical Documentation 2-25
Available Reference Implementations 2-25
Get Latest Revision Extensibility 2-26
Possible Uses 2-26
Technical Overview 2-27
Technical Documentation 2-27
Available Reference Implementations 2-27
Hierarchy Denormalization Extensibility 2-28
Overview 2-28
Possible Uses 2-28
Technical Documentation 2-28
Hierarchy Navigator Extensibility 2-29
Possible Uses 2-29
Technical Overview 2-29
Display Options (Identity) 2-29
Creating an Identity 2-30
Sort By Options 2-31
Creating a SortBy 2-31
Filters 2-32
Creating a Filter 2-33
Context Menu 2-34
<MenuItem> Attributes 2-34
Creating a Context Menu Item 2-34
Creating a Label 2-34
Workflow Actions 2-35
Identity Plugins 2-36
Object Identity Plugins 2-36
GSM Identity Plugins 2-37
PQM Identity Plugins 2-39
Possible Uses 2-39
Technical Overview 2-40
Technical Documentation 2-40
Available Reference Implementations 2-40
Label Claims Extensibility 2-42
Technical Overview 2-42
Technical Documentation 2-42
Available Reference Implementations 2-42
Navigation Extensibility 2-43
Possible Uses 2-45
Technical Overview 2-45
Navigation Extensibility: Supplier Portal 2-46
Possible Uses 2-47
Technical Overview 2-47
Notification Panel 2-48
Possible Uses 2-48
Technical Overview 2-48
Custom Notification Table 2-49

Technical Documentation 2-49
Available Reference Implementations 2-49
Print Extensibility 2-50
Possible Uses 2-50
Technical Overview 2-50
Product Portfolio Management Integration 2-51
Use Cases 2-51
Supported Versions 2-53
Technical Documentation 2-53
Quick Links 2-54
Refresh Hierarchy Warning Plugin 2-55
Technical Overview 2-55
Rich Text Extensibility 2-56
Possible Uses 2-56
Technical Overview 2-56
Available Reference Implementations 2-56
Javascript Wrapper Example for the CkEditor 2-56
Provider Class Example for the CkEditor 2-58
Search Extensibility 2-60
Possible Uses 2-60
Technical Documentation 2-60
Section Level Editing 2-61
Possible Uses 2-61
Technical Overview 2-61
Technical Documentation 2-61
Available Reference Implementations 2-61
Side Bar 2-63
Specification Veto Plugin 2-64
Possible Uses 2-64
Technical Overview 2-64
Technical Documentation 2-64
Available Reference Implementations 2-64
PQM Veto Plugins 2-65
Custom Read Permission 2-65
Custom Write Permission 2-65
Technical Documentation 2-65
Supporting Document Extensions 2-66
External URL Sample 2-66
Technical Documentation 2-66
Configuration Changes 2-66
External URL Page Changes 2-69
User Interface Extensions 2-70
Technical Overview 2-70
Technical Documentation 2-70
Validation Framework 2-71
Possible Uses 2-71
Technical Overview 2-71

Default Language 2-72
Technical Documentation 2-73
Workflow Actions and Guard Conditions 2-74
Possible Uses 2-74
Technical Overview 2-74
Technical Documentation 2-74
Available Reference Implementations 2-74
Workflow Actions—Automatic Workflow 2-75
Possible Uses 2-76
Technical Overview 2-76
Technical Documentation 2-77
Workflow UI Extensions 2-78
Possible Uses 2-78
Technical Overview 2-78
Creating a Workflow Action 2-78
Notes: 2-78

A Developer Information

PLM4PExtensionUtils Developer Utility Library A-1
Object Loader URLs A-2
Format A-2
Common Usage A-2
Example A-2
Passing Parameters in the ObjectLoaderURL A-2
Object and Data Schema Documentation A-3
Database Tables A-3
Data Objects A-4
Other Available Data A-4
Additional Details A-5
PKIDs—Primary Key Identifiers A-5
OR Metadata Tables A-5
Language Aware Tables A-6

Preface

The *Agile Product Lifecycle Management for Process Extensibility Overview 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
- Variability of Installations
- Documentation Accessibility
- Software Availability
- Related Documents
- Conventions

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

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Software Availability

Oracle Software Delivery Cloud (OSDC) provides the latest copy of the core software. Note the core software does not include all patches and hot fixes. Access OSDC at:

http://edelivery.oracle.com

Related Documents

For more information, see the following documents in the Oracle Agile Product Lifecycle Management for Process 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 FlexSync Foundation 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 Reporting 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 Columns Guide*
- *Agile Product Lifecycle Management for Process Hierarchy Denormalization 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

This guide contains detailed information on the extensibility points included in the core product suite, reference example source code, and compiled reference examples.

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 as deployable tools, web applications, database scripts, or utility classes.

Sample Code Disclaimer

The files provided as reference implementations, which have been provided by Oracle Corporation as part of an Oracle® product for use ONLY by licensed users of the product, include CONFIDENTIAL and PROPRIETARY information of Oracle Corporation.

USE OF THIS SOFTWARE IS GOVERNED BY THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT AND LIMITED WARRANTY FURNISHED WITH THE PRODUCT.

IN PARTICULAR, YOU WILL INDEMNIFY AND HOLD ORACLE CORPORATION, ITS RELATED COMPANIES AND ITS SUPPLIERS, **HARMLESS** FROM AND AGAINST ANY CLAIMS OR LIABILITIES ARISING OUT OF THE USE, REPRODUCTION, OR DISTRIBUTION OF YOUR PROGRAMS, INCLUDING ANY CLAIMS OR LIABILITIES ARISING OUT OF OR RESULTING FROM THE USE, MODIFICATION, OR DISTRIBUTION OF PROGRAMS OR FILES CREATED FROM, BASED ON, AND/OR DERIVED FROM THESE SAMPLE SOURCE CODE FILES.

Technical Requirements

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

- .NET 4.5
- Visual Studio
- Proficiency in C#

- XML
- SQL (T-SQL or PL/SQL)
- Working knowledge of XSLT and XSL-FO (for printing customization)
- JavaScript (UI Extensibility requires basic JavaScript knowledge)

2

Extensibility Points

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

- BOM Calc Extensions
- Calculation Veto Plugin
- Clone Extensibility
- Cost Extensions
- Custom Data Denormalization
- Custom Portal
- Email Extensions
- eSignature Validate Plugin
- Event Framework
- Extended Attribute Calculations
- Extensible Columns
- FlexSync Foundation
- Formulation Output Naming Plugins
- Formulation Percent Breakdown Classification Override Plugin
- Formulation Push Percent Breakdown Plugin
- Get Latest Revision Extensibility
- Hierarchy Denormalization Extensibility
- Hierarchy Navigator Extensibility
- Identity Plugins
- Label Claims Extensibility
- Navigation Extensibility
- Navigation Extensibility: Supplier Portal
- Notification Panel
- Print Extensibility
- Product Portfolio Management Integration
- Quick Links

- Refresh Hierarchy Warning Plugin
- Rich Text Extensibility
- Search Extensibility
- Section Level Editing
- Side Bar
- Specification Veto Plugin
- Supporting Document Extensions
- User Interface Extensions
- Validation Framework
- Workflow Actions and Guard Conditions
- Workflow Actions—Automatic Workflow
- Workflow UI Extensions

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 [ProdikaHome]\Installer\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 Installer under the folder ReferenceImplementations/BomCalc. This reference implementation does not do any calculations other than 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 [ProdikaHome]\Installer\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 ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins for details.

Clone Extensibility

Clone extensibility allows you to control which fields are included when a specification or object is copied. This extensibility point allows you to control clone rules in the following applications: GSM, SCRM, NPD, PQM, UGM, EQ, DRL and NSM. Clone extensibility could be as simple as a quick xml configuration update to copy the "Available UOMs" section when a specification is cloned to more complex conditional logic like "if the specification category is Dairy then copy Short Name, else leave blank."

Clone Event Types

There are four clone event types supported by clone extensibility:

1. Copy
2. New Issue
3. Create From Template
4. Target Revision

Possible Uses

1. Always include Available UOMs section when copying a specification.
2. Provide a UI that allows users to select which elements are copied when a packaging specification is issued.

Technical Documentation

To learn more about clone extensibility and review some out of the box reference implementations refer to the **Clone Extensibility** document, located in the [ProdikaHome]\Installer\ReferenceImplementations\CloneExtensibility\Documentation folder for more details.

Cost Extensions

Extensions are available around costing for trade, packaging material, and formulation specifications in the following areas:

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

Possible Uses

1. Formulation 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. Trade Displayed Calculated Cost—Any raw material that is used in amounts greater than 50lbs increase the cost by 2%.
3. Packaging Material Theoretical Cost Book Entry—Adjust the total theoretical cost for the parent Packaging Material based on the Per Unit "Labor Cost" extended attribute due to required assembly.

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 names
Input Cost	TradeAssociatedMaterialCostPriceOverride
	TradeLowerLevelTradeCostPriceOverride
	TradePackagingSubcomponentCostPriceOverride
	PackagingSubComponentCostPriceOverride
	FormulationInputCostBookPriceOverride
Formulation Displayed Calculated Cost	TradeAssociatedMaterialTheoreticalCostPriceOverride
	TradeLowerLevelTradeTheoreticalCostPriceOverride
	TradePackagingSubcomponentTheoreticalCostPriceOverride
	PackagingSubComponentTheoreticalCostPriceOverride
	FormulationOutputTheoreticialCostPriceOverride
Formulation Theoretical Cost Book Entry	TradePersistedTheoreticalCostPriceOverride
	PackagingPersistedTheoreticalCostPriceOverride
	FormulationOutputPersistedTheoreticialCostPriceOverride

Technical Documentation

Refer to the **PluginExtensions** document, located in [ProdikaHome]\Installer\ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins\FormatPlugins\CostExtensions for more details.

Available Reference Implementations

1. Highest, Lowest, and Average cost plugins—Classes that return the highest, lowest, or average cost for a formulation input when there are several possible valid costs.

Note that while this capability is now a core feature of the product (available through CustomerSettings.config in the GSMSettings node using the entry `<add key="GSM.Cost.MultipleFound" value="Highest/Lowest/Average"></add>`), the Reference Implementation code is still available to demonstrate building a costing extension.

Source Code: See ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins\FormatPlugins\CostExtensions for details.

Custom Data Denormalization

Custom Data Denormalization is available via two denormalization techniques: Custom Section Denormalization and Extended Attribute Denormalization. PLM for Process 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:

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: [ProdikaHome]\Installer\Extensions\MockCustomPortalPlugins\.

Email Extensions

PLM for Process provides various automated email notifications when certain business objects move from one status to another. Emails can be sent:

- To owners of a certain object informing them that the item is now in their action items listing
- To users asking them to sign off on the item
- To users as a simple notification that the item moved from one status to another.

The email recipients for GSM, SCRM and PQM are specified in the WFA application, using the Owners, Signature Request, and Notifications grids.

Emails for GSM, SCRM, PQM, UGM, Supplier Portal, and NPD may be customized in the following ways:

1. You can change the email subject and body contents by modifying the translations that are offered out of the box, using placeholder variables that are replaced with data from the business object. Customizing the content of workflow emails in GSM and SCRM without custom code is now easier with the addition of a new set of placeholder variables, conditional variables, and other capabilities. These variables are listed in Email Extensions.docx file.

   Some examples of the new functionality include:

   - Displaying new data in the emails, such as Cross Reference numbers, Category/Sub-category/Group, a trade specification's GTIN, a material specification's Ingredient Statement, and more.
   - Providing meaningful related specification information in Signature Document emails.
   - Conditionally displaying data, such as only including a material specification's classification if it is an Approved status.

2. You can create 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 "Management Review" 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. See the documentation for more details.

Technical Documentation

See the [ProdikaHome]\Installer\ReferenceImplementations\EmailExtensions\Documentation\Email Extensions.docx file for details.

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 PluginExtensions document, located in [ProdikaHome]\Installer\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 Framework

Technical Overview

The Event Framework provides a consistent, extensible, and flexible way to handle raised events throughout the application suite. System events are raised for actions such as Save, Workflow, Read, Copy, Print, Login (captured for all applications), and more, and are currently active in GSM, SCRM, PQM, and NPD (projects and activities. Customers can subscribe to and act on core application events by leveraging out-of-the-box event subscribers or creating their own. Event Subscribers are classes that get called to handle specific events, and are passed event argument data, such as the object being acted on (eg., the current specification).

A flexible configuration allows for simple ways to specify which events a subscriber should be called to handle and when. For example, a subscriber can be configured to get called for any GSM specification Copy event if the specification is not in Approved Status, or for Material Spec Read events when the specification is Approved and the user is in a specific Group. Additionally, the configuration itself is extensible so that customers can create their own filters to help determine if a subscriber should be called.

Possible Uses

The events can be the catalyst to a third party system action (email notifications, data comparisons, etc.). They can also be used to create audit tables. Here are a few examples:

1. Capture every time a user reads a spec and store it in a custom Audit table.
2. Every time a material specification is created, the Material Manager is notified by email.
3. 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 by email.

Technical Documentation

Event subscribers are C# classes that receive event related information and are specified in the config\Extensions\CustomEventing.xml file. The subscriber configuration can specify events such as GSM.Item.5816.Create for formulation specification creation events, or GSM.Item.*.Copy for any specification Copy events.

For more details, see the *Agile Product Lifecycle Management for Process System Events Extensibility Guide*.

Available Reference Implementations

A reference implementation is available that demonstrates how to notify a user if the specification allergens have been changed. This can be found in the following location:

[ProdikaHome]\Installer\ReferenceImplementations\EventFrameworkExtensions

Legacy Event Model Tables

The previous eventing solution was a table driven approach that populated two tables, CommonLifecycleEventLog and PQMLifecycleEventLog, based on system events. These tables will not be supported long term. Customers wishing to take advantage of events should create a subscriber for the events they are interested in, using the new Event Framework. Customers already using these tables should plan on creating a subscriber that populates similar tables to the ones above, resulting in minimal rework of their solution.

Tips: As part of the feature replacement plan, the legacy table-based logging feature "Common.Auditing.LifecycleEvents.Enabled" has been refigured as a prerequisite of the built-in Hierarchy Denormalization feature. So it has been renamed to "HierarchyDenorm.Auditing.LifecycleEvents.Enabled" while the legacy logging behavior doesn't change at all. If a customer wishes to stay on their old, existing extensions which are dependant on the legacy table "CommonLifecycleEventLog" for a short while, just simply adjust the feature name in CustomerSettings.config to be "HierarchyDenorm.Auditing.LifecycleEvents.Enabled". That provides a buffer time to further code migration. Note that there is no change at the "PQM.Auditing.LifecycleEvents.Enabled" feature in this release, which should be moved out in the next several major releases.

Data Captured

We offer a few out of the box database tables that do not get purged regularly. These tables store key events in GSM, SCRM and PQM.

Each event captured may include the following information:

Event Type—The type of event that occurred 1: Create, 2: Save, 3: Workflow, 4: Copy, 5: Get Latest Revision, 6: Add Material to Substitutes

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.)

Technical Overview

A feature configuration will determine if events will be logged.

GSM & SCRM—`HierarchyDenorm.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
)
```

Table of Logged Events

Table 2–1 Table of Logged Events

Event Source	Event Type	Affected Object	Related
GSM.Editor	Save	Specification saved (pkid)	
GSM.Clone	Create	New specification created (pkid)	
GSM.Clone	Copy	Specification copied (pkid)	New specification created (pkid)
GSM.NewIssue	Create	New specification Issued (pkid)	
GSM.NewIssue	Copy	Specification copied (pkid)	New specification created (pkid)
GSM.Workflow.Transition	Workflow	Specification transitioned (pkid)	Workflow Step (pkid)
GSM.Workflow.Resolve	Workflow	Resolved specification (pkid)	Workflow Step (pkid)
GSM.Workflow.Resolve	Save	Resolved specification (pkid)	
GSM.SmartIssue	Create	Specification created (pkid)	Smart Issue Request (pkid)
GSM.SmartIssue	Copy	Specification copied (pkid)	New specification created (pkid)
GSM.GlobalSuccession	Save	Host specification (pkid)	Specification that was replaced (pkid)
GSM.Revision	Revision	Revision Object (pkid)	
GSM.Substitute	Substitute	Substitute Material Spec (pkid)	
SCRM.Editor	Save	Specification object (pkid)	
SCRM.Clone	Create	Sourcing Approval copied (pkid)	
SCRM.Clone	Copy	Sourcing Approval copied (pkid)	Sourcing Approval created (pkid)
SCRM.Workflow.Transition	Workflow	Sourcing Approval transitioned (pkid)	Workflow Step (pkid)
SCRM.Workflow.Resolve	Workflow	Resolved specification (pkid)	Workflow Step (pkid)
SCRM.Workflow.Resolve	Save	Resolved specification (pkid)	
PQM.Editor	Create	New PQM object (pkid)	
PQM.Editor	Edit	PQM object Save (pkid)	

Available Event Subscribers

The following out of the box Event Subscribers are available for any event type. These Event Subscribers simply log event information, including the application name, event name, primary object id, secondary objectid, object status info, userID, and more to a database table or a file.

Database Logger—An out of the box subscriber is available that can log events to a database table. By default, this logs to the CommonEventingLog, but it can be changed to log to other tables created by customers

File Logger—An out of the box subscriber is available that can log events to a configurable system file.

See the Event Framework document for more details.

Available Event Subscriber Filters

Event Subscribers can be controlled by the event type, the object types, the object status, and more, through simple configuration options. Additionally, filters can be used to provide further filtering options.

The following out of the box Event Subscriber Filters are available for any event type.

User Group Filter—Allows for filtering an event by a list of included and/or excluded User Groups

User Role Filter—Allows for filtering an event by a list of included and/or excluded User Roles

Segments Filter—Allows for filtering an event by a list of included and/or excluded Segments for the current object

Reflective Property Filter—An out of the box event subscriber filter allows for filtering an event by a list of included and/or excluded values to compare against a specific property on the current object. For instance, this filter can be used to restrict the event to specific trade spec types (eg, TU co-pack)

Custom Filters can be added for further control.

See the Event Framework document for more details.

Available Reference Implementations

An Event Subscriber reference implementation is available that demonstrates how to notify a user if the specification allergens have been changed.

An Event Subscriber Filter reference implementation is available which uses a Material Spec Classification as a filter, allowing filtering by a list of classifications for inclusion and a list for exclusion.

These can be found in the following location:

[ProdikaHome]\Installer\ReferenceImplementations\EventFrameworkExtensions

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* for more details.

Available Reference Implementations

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

Two cost calculation classes, Formulation Total Costs Calculator and Trade Total Costs Calculator, demonstrate how to get the total costs of a specification.

See the reference implementation in [ProdikaHome]\Installer\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 (Deprecated)

Printed Packaging (Deprecated) > Packaging

Printed Packaging (Deprecated) > 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*.

Available Reference Implementations

1. Formulation Input and Output Quantity as Volume—These plugins convert and display the formulation inputs or outputs to a volume measurement.

- See the InputVolumeNoStorageExtensibleColumnPlugin and OutputVolumeNoStorageExtensibleColumnPlugin classes for formulation extensible columns examples that do not store the volume in the database.
- See the InputFloatWithOptionalUOMColumnPlugin (along with the InputNumericWithOptionalUOMColumnPluginFactory) and the OutputStoredFloatWithOptionalUOMColumnPlugin (along with the OutputNumericWithOptionalUOMColumnPluginFactory) for formulation extensible columns that do store the volume in the database, as well as the related calculation methods.

2. Preferred UOM Input Quantity and Yield and Output Quantity and Yield column plugins—These plugins will display new input or output columns with quantity or yield values using the user's preferred Units of Measure. For instance, if the formulation is using pounds for the Quantity values, and the user's preferred UOM is KG, the new column would display the value in KG.

Source Code: See [ProdikaHome]\Installer\ReferenceImplementations\ExtensibleColumns\SourceCode\FormulationExtensionsSample\ for many examples.

FlexSync Foundation

Overview

FlexSync Foundation provides the platform necessary to create, manage, and orchestrate reports, providing you with the ability to export relevant data to a customized Excel user interface with macros, manipulate the data based on business needs, and then import the data back into PLM4P to maintain your single source of truth.

Features include:

- Chains of data handlers allow data to be pulled/pushed via a variety of mechanisms allowing for unlimited extensibility.
- Oracle-provided general purpose handlers provide out of the box functionality.
- Reusability of out of the box templates. Templates can be modified to provide additional functionality without the need for a developer to create new handler chains.
- Table based report configuration allows for easy report creation/maintenance.

FlexSync foundation is the base for the FlexSync Formulation feature. To enable FlexSync formulation:

1. Set configuration: `<add key="GSM.FlexSyncFormulation.Enabled" value="true" configDescription="Enables the flexSync formulation feature when configure is true. When it's enabled, Users assigned with role [FRM_FLEXSYNC] can access the FlexSync output tool for formulation specifications. "/>` in the CustomerSettings.config file.
2. Restart IIS.

Technical Documentation

For more information, see the *Agile Product Lifecycle Management for Process FlexSync Foundation Guide* and the *Agile Product Lifecycle Management for Process FlexSync Formulation Guide*.

Formulation Output Naming Plugins

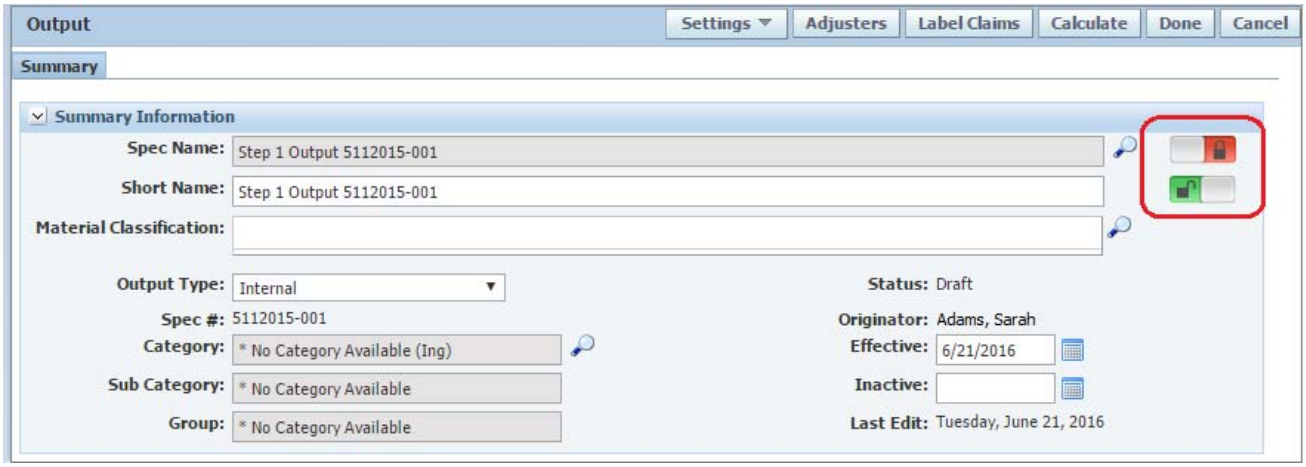
An output naming plugin is available to set formulation output name as user preferred automatically. When the name field is locked, it will refresh automatically when it meets the configuration criteria.

The out of box implementation is turned off.

As explained below, clients can use the default internal/external output naming factory with parameters and create their own internal/external output naming factory.

Note the lock/unlock for output Spec Name and Short Name in the Output dialog. If turned on, users can unlock for the name if they don't want to use it. The lock feature is are controlled by configurations described in the *Agile Product Lifecycle Management for Process Configuration Guide*.

Figure 2–1 Example of locked and unlocked fields



Possible Uses

1. Set First External output name same as formulation name. Currently, all the external output has a non-descriptive name like 'Step 1 Output xxxxx-001', which is annoying when a user tries to search this specification, or has to manually change the name to a more meaningful one before saving. Now with this new feature, the external output name will be refreshed to same as formulation name automatically, saving the user a lot of time.
2. Set different output name rule based on different output type. For example, when output type is 'External-Product', set the name same as formulation name; when output type is 'External - Waste', set the name same as 'formulation name + waste'. You can set up as many rules as you prefer.

Technical Documentation

Refer to the **PluginExtensions** document, located in the [ProdikaHome]\Installer\ReferenceImplementations\PluginExtensions\Documentation folder for more details.

Available Reference Implementations

All output types are supported and are controlled by the following plugins.

- Internal Output Spec Name

Internal Output Spec Name Format Plug-in: FrmInternalOutputMaterialNamePlugin

Default Internal Output Spec Name Factory:
FormulationOutputMaterialNamePluginFactory

- External Output Spec Name

External Output Spec Name Format Plug-in: FrmExternalOutputMaterialNamePlugin

Default External Output Spec Name Factory:
FormulationOutputMaterialNamePluginFactory

- Internal Output Short Name

Internal Output Short Name Format Plug-in:
FrmInternalOutputMaterialShortNamePlugin

Default Internal Output Short Name Factory:
FormulationOutputMaterialShortNamePluginFactory

- External Output Short Name

External Output Short Name Format Plug-in:
FrmExternalOutputMaterialShortNamePlugin

Default External Output Short Name Factory:
FormulationOutputMaterialShortNamePluginFactory

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 [ProdikaHome]\Installer\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 [ProdikaHome]\Installer\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.

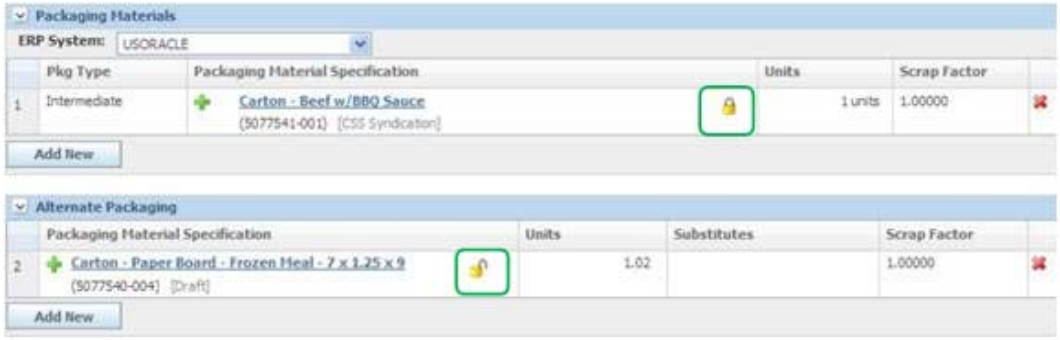
Get Latest Revision Extensibility

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 2–2 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 [ProdikaHome]\Installer\ReferenceImplementations\GetLatestRevision\Documentation for more details.

Available Reference Implementations

A reference implementation demonstrates how to prevent a trade specification 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 ReferenceImplementations\GetLatestRevision\SourceCode for implementation examples.

Hierarchy Denormalization Extensibility

Overview

Agile PLM for Process stores objects, such as specifications, along with the relationships to each other, in a normalized database schema, making inserts, updates and deletes highly efficient while minimizing its size. The challenges with having a normalized schema are that it can make custom SQL queries complex and possibly not optimal for bulk data retrieval. For example, to construct a report that returns the entire hierarchy of a trade specification, would require a deep understanding of many relationship tables and would be extremely difficult to do in SQL alone, due to the varying number of possible layers in the hierarchy. A hierarchy of a specification is defined as that specification plus all descendant specifications as well as other related objects. For example, these objects would be considered part of a trade hierarchy:

- The main trade specification
- All lower level trade specifications
- The material specification directly associated to the trade specification
- The formula to create the above material
- All inputs and outputs to the above formula
- All formulas that create the above inputs
- All inputs and outputs to the above formulas

By continuing to drill down into the formula and intermediate formulas that comprise a trade specification, you will have what we are referring to as the Trade Hierarchy. This hierarchy is not limited to the relationships defined above but covers many of the relationships that are defined in PLM for Process.

Hierarchy Denormalization was designed as a near real-time backend feature in RemotingContainer. By adjusting the corresponding configuration nodes, the denormalizers can reflect a little faster or slower but cannot reach the absolute real-time. It provides a solution to this data access problem by storing the object relationship information in a single table, allowing for simple and performant hierarchy retrieval.

Many solutions can use this table to provide functionality such as hierarchical navigation and reporting.

Possible Uses

1. Reporting
2. Analytics
3. Hierarchy Navigator

Technical Documentation

For more information, see the *Agile Product Lifecycle Management for Process Hierarchy Denormalization Guide*.

Hierarchy Navigator Extensibility

The following areas of Hierarchy Navigator are extensible:

- Display Options (Identities)
- Sort By Options
- Filters
- Node Contextual Menus
- Workflow Actions

This section outlines important files used to enable the feature. You can modify it based on your implementation.

Possible Uses

1. Show the Segment of all items in Hierarchy Navigator panel so user can make sure they are all from the same segment type.
2. Add a new filter to show items only owned by current user.

Technical Overview

Display Options (Identity)

Identities are used to display object information in a tree node. There are several default identities for display view under the "HierarchyNavigatorCustomExtensions\IdentityExtensions" section. You can add new entries or override an existing one to change the display item.

Example:

```
<IdentityExtensions>
    <Identity Key="ObjectNameIdentity" SortOrder="10" DisplayText="Object Name"
    Default="true" >
        <Retriever ObjectTypes="1004,1005,1006"
        Factory="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtension
        s.ObjectNameIdentity.GSMObjectNameIdentityExtension, WebCommon" />
    </Identity>
</IdentityExtensions>
```

Table 2–2 <Identity> Attributes

Attribute	Description	Notes
Key	Identify the entry	Used for translation as well
SortOrder	Specify an order number	
DisplayText	Specify text note for the entry	This will be used as the caption of the item if the node has no translation
Default	Specify if the item will be default displayed on the hierarchy node	true/false

Table 2–3 <Identity/Retriever> Attributes

Attribute	Description	Notes
ObjectTypes	Specify the object type which the retriever applied for	Can be set with multiple object types
Factory	Specify the retriever implementation factory	

Creating an Identity Complete the following steps for creating a new identity. The user can select the new identity from display view after configured. It will display the information on the tree node when new identity is checked on.

1. Create a new identity retriever extension based on the interface. For example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Xeno.Prodika.Common;

namespace
Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtensions.IdentityExtensions
{
    public class IdentityExtensionFactoryExample1 : IFactory
    {
        public Object Create()
        {
            return new IdentityExample1();
        }
    }

    public class IdentityExample1 : IIdentityExtension
    {
        public string GetFormatedIdentity(IHierarchyNavigatorNode obj)
        {
            return "IdentityExample1";
        }

        public virtual string GetIdentity(IHierarchyNavigatorNode obj)
        {
            return "IdentityExample1";
        }
    }
}
```

2. Build and put the compiled DLL into application directory. Let's say "MyExtensions.dll".

3. Add an entry to "HierarchyNavigatorCustomExtensions\IdentityExtensions".

```
<Identity Key="TestIdentity" SortOrder="60" DisplayText="TestIdentity"
Default="true" >
    <Retriever ObjectTypes="<YourObjectTypes>"
    Factory="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtensions.Id
    entityExtensions.IdentityExtensionFactoryExample1, MyExtensions" />
</Identity>
```

4. Restart IIS.

Sort By Options

There are several default entries for sort view under "HierarchyNavigatorCustomExtensions\SortByExtensions" section. User can add new entry or override existing one to change the sort item.

Example:

```
<SortByExtensions>
    <SortBy Key="ObjectNameSort" DisplayText="Object Name" SortOrder="10" >
        <Retriever ObjectTypes="1004,1005,1006"
        Factory="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtension
        s.ObjectNameSort.GeneralObjectNameSort, WebCommon"/>
    </SortBy>
</SortByExtensions>
```

Table 2–4 <SortBy> Attributes

Attribute	Description	Notes
Key	Identify the entry	Used for translation as well
SortOrder	Specify an order number	
DisplayText	Specify text note for the entry	This will be used as the caption of the item if the node has no translation
Default	Specify if the item will be default displayed on the hierarchy node	true/false

Table 2–5 <SortBy/Retriever> Attributes

Attribute	Description	Notes
ObjectTypes	Specify the object type which the retriever applied for	Can be set with multiple object types
Factory	Specify the retriever implementation factory	

Creating a SortBy Complete the following steps for creating a new SortBy. The user can sort hierarchy content by the new SortBy from sorting view after configured.

1. Create a new SortBy retriever extension based on the interface. For example:

```
using Xeno.Data;

namespace
Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtensions.SortByExtensions

{
  public class SortByFactoryExample1 : IFactory
  {
    public Object Create()
    {
      return new SortByExample1();
    }
  }
```

```
  public class SortByExample1 : ISortExtension
  {
    public int Compare(object x, object y)
    {
      return 0;
    }
  }
}
```

2. Build and put the compiled DLL into application directory. Let's say "MyExtensions.dll".
3. Add an entry to "HierarchyNavigatorCustomExtensions\SortByExtensions".

```
<SortBy Key="TestSortBy" DisplayText="Test SortBy" SortOrder="60" >
    <Retriever ObjectTypes="<YourObjectTypes>"
    Factory="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtensions.So
    rtByExtensions.SortByFactoryExample1, MyExtensions"/>
</SortBy>
```

4. Restart IIS.

Filters

There are several default entries for filters under "HierarchyNavigatorCustomExtensions\FilterExtensions" section. User can add new entry or override existing one to change the filter item.

Example:

```
<FilterExtensions>
    <Filter Key="All" DisplayText="Select All" SortOrder="10" >
        <Filter Key="ObjectTypeFilter" DisplayText="Object Type" SortOrder="10"
        DynamicGenerator="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.Custom
        Extensions.ObjectTypeFilter.ObjectTypeFilterGenerator,WebCommon">
        </Filter>
        <Filter Key="AlternateFilter" DisplayText="Alternate" SortOrder="30" >
            <Retriever ObjectTypes="1004,1005,1006"
            Factory="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExten
            sions.AlternateFilter.AlternateFilter, WebCommon"/>
        </Filter>
    </Filter>
</FilterExtensions>
```

Table 2–6 <Filter> Attributes

Attribute	Description	Notes
Key	Identify the entry	Used for translation
SortOrder	Specify an order number	
DisplayText	Specify text note for the entry	This will be used as the caption of the item if the node has no translation

Table 2–6 <Filter> Attributes

Attribute	Description	Notes
DynamicGenerator	Specify a generator for generating a list based on the hierarchy content	No need to add retriever if you specify this attribute

Table 2–7 <Filter/Retriever> Attributes

Attribute	Description	Notes
ObjectTypes	Specify the object type which the retriever applied for	Can be set with multiple object types
Factory	Specify the retriever implementation factory	

Creating a Filter Complete the following steps for creating a new filter. The user can filter hierarchy content by the new filter from filter view after configured. Nested filter is supported.

1. Create a new filter retriever extension based on the interface. For example:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace
Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtensions.FilterExtensions
{
   public class FilterExtensionFactoryExample1 : IFactory
    {
        public Object Create()
        {
            return new FilterExample1();
        }
    }

    public class FilterExample1 : IFilter
    {
        public bool IsItemIncluded(IHierarchyNavigatorNode node)
        {
            return (int)node.DenormResult["fkAncestorRelationshipContext"] >= 0;
        }
    }
}
```

2. Build and put the compiled DLL into application directory. Let's say "MyExtensions.dll".

3. Add an entry to "HierarchyNavigatorCustomExtensions\FilterExtensions".

```
<Filter Key="FilterExample1" DisplayText="FilterExample1" SortOrder="30" >
    <Retriever ObjectTypes="1004,1005,1006"
    Factory="Class:Oracle.PLM4P.SolutionPack.HierarchyNavigator.CustomExtensions.Fi
    lterExtensions.FilterExtensionFactoryExample1, MyExtensions"/>
</Filter>
```

4. Restart IIS.

Context Menu

The navigator allows you to add a contextual menu per node. This menu is accessed by right clicking on the node. For example, the user could launch a report from a node within the tree. User can configure context menu when right click on one node from hierarchy navigator. The menu configuration follows the same structure and concepts as the action navigation extension. (See the Agile Product Lifecycle Management for Process Navigation Configuration Guide for more details.)

The "HierarchyNavigatorTreeviewContextMenu" node located in Sitemap-extensions.xml is used for the hierarchy navigator context menu.

Example:

```
<MenuItem ID="HierarchyNavigatorTreeviewContextMenu">
    <MenuItem ID="lblTest"  ClientSideCommand="alert(NavigatorNode.nodeIdentifier)"
    />
</MenuItem>
```

<MenuItem> Attributes

See the *Agile Product Lifecycle Management for Process Navigation Configuration Guide*.

Table 2–8 Variables

Variable	Value	Notes
NavigatorNode.nodeIdentifier	The active node PKID when right clicking	
NavigatorNode.level	The level of current node	
NavigatorNode.url	The object URL for current node	
NavigatorNode.childItems	Child item of current node	An array of node items
NavigatorNode.hasChildren	Indicate a node has children or not	true/false

Creating a Context Menu Item Complete the following steps for adding a new context menu item. The user can right click on a node to get the menu item configured.

1. Add an entry to "HierarchyNavigatorTreeviewContextMenu".

```
<MenuItem ID="HierarchyNavigatorTreeviewContextMenu">
    <MenuItem ID="lblTest"  ClientSideCommand="alert(NavigatorNode.nodeIdentifier)"
    />
</MenuItem>
```

2. Restart IIS.

Creating a Label Labels for the context menu are created the same way as outlined in the Agile Product Lifecycle Management for Process Navigation Configuration Guide.

A translatable or non-translatable label can be created.

Non-translatable: Omit the ID attribute from the MenuItem node and add the DisplayText attribute with the label value. Example:

```
<MenuItem ID="HierarchyNavigatorTreeviewContextMenu">
    <MenuItem  DisplayText="TestLabel"
    ClientSideCommand="alert(NavigatorNode.nodeIdentifier)" />
</MenuItem>
```

Translatable: Add the ID attribute to the MenuItem node and add the proper translations to the commonXLAExtensionCacheItem table. Example:

```
<MenuItem ID="HierarchyNavigatorTreeviewContextMenu">
    <MenuItem  ID='HierarchyNavigatorLabelTest'
    ClientSideCommand="alert(NavigatorNode.nodeIdentifier)" />
</MenuItem>
```

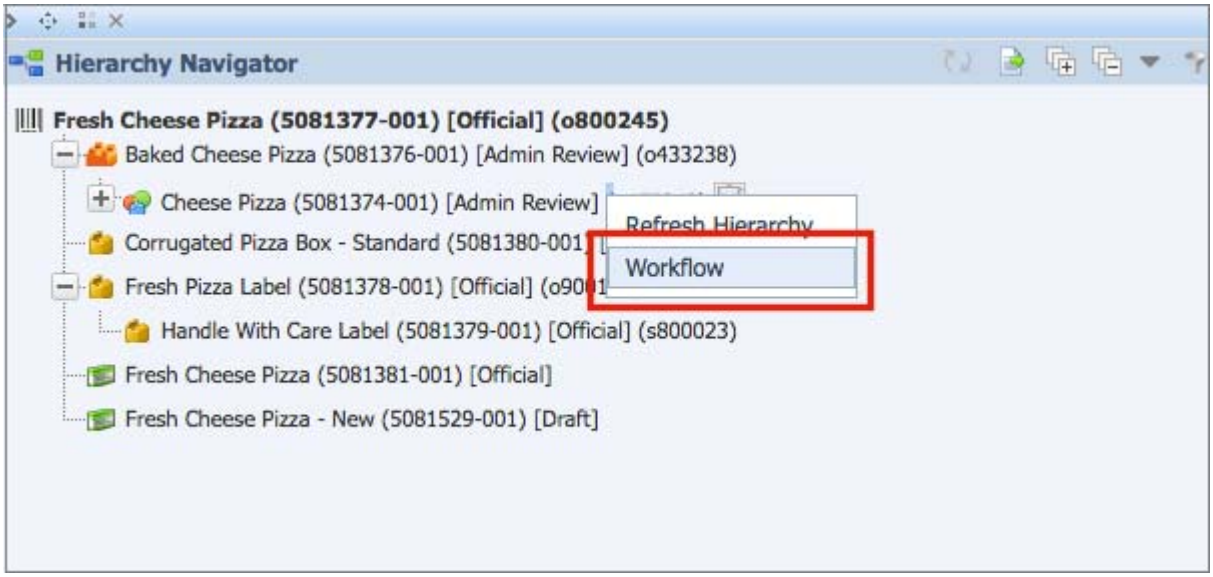
To add the translation you will need to use the 'NavigationMenu' translation cache found in the commonXLAExtensionCache database table.

Workflow Actions

You can workflow a specification from multiple locations throughout the GSM application. You no longer have to be on the specification to initiate a workflow. Workflow is available from Hierarchy Navigator as well as specification search result listings.

Calling the workflow dialog is also possible from other locations throughout GSM as an extension. This setup allows you to place the workflow action in other locations throughout GSM using UI extensibility or navigation extensions. You can reference "Workflow UI Extensions" in this guide to learn how to call the workflow dialog panel.

Figure 2–3 Right-click a specification within the hierarchy and select the Workflow action



Identity Plugins

The application suite includes three types of identity plugins which are described below.

Object Identity Plugins

Throughout the application suite, all objects are displayed with an associated suite header, object header, and Most Recently Used (MRU) menu. All of these locations are extensible and can be adjusted. For example, if the Short Name is an important identifier for your GSM specifications you can display it next to the specification name or even replace the specification name entirely. All configurations can be found in PluginExtensions.xml and follow specific naming conventions. The following areas are extensible:

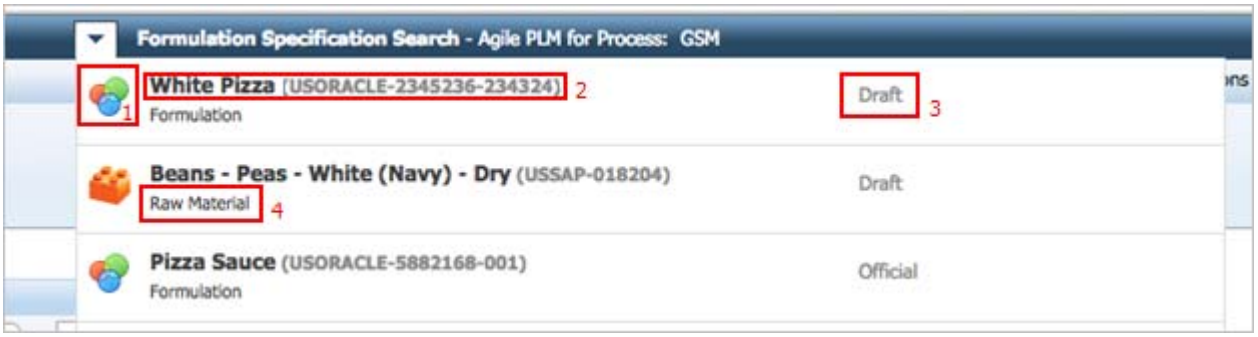
Figure 2–4 Object Header

The Suite and Object Identity Header can be broken apart into multiple extensible components, which are highlighted in Figure 2–4 above.

Table 2–9 Suite and Object Identity Header, extensible components

Object Identity Component	Plugin Naming Convention
1. Suite Header Page Name	SuiteHeader.PageName.APPNAME.OBJECT
2. Object Icon	ObjectHeader.ObjectIcon.APPNAME.OBJECT
3. Object Page Name	ObjectHeader.ObjectCaption.APPNAME.OBJECT
4. Object Type	ObjectHeader.ObjectType.APPNAME.OBJECT
5. Object Status	ObjectHeader.ObjectStatus.APPNAME.OBJECT

Figure 2–5 Most Recently Used



Most Recently Used (MRU) can be broken apart into multiple extensible components, which are highlighted in Figure 2–5:

Table 2–10 Most Recently Used, extensible components

Object Identity Component	Plugin Naming Convention
1. Object Icon	MRU.ObjectIcon.APPNAME.OBJECT
2. Object Page Name	MRU.ObjectCaption.APPNAME.OBJECT
4. Object Status	MRU.ObjectStatus.APPNAME.OBJECT
5. Object Type	MRU.ObjectTypeNameOverride.APPNAME.OBJECT

The Object Identity plugins include the following capabilities:

- NameMaxLength—Configuration setting that tells the plugin the maximum length the name can be.
- NumberMaxLength—Configuration setting that tells the plugin the maximum length the number can be.

GSM Identity Plugins

Throughout the application suite there are many grids and fields that display related specifications. For example, Formulation Input BOM and Trade Packaging BOM. The 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 34 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 GSM identity plugins are available in the following UI locations:

Table 2–11 GSM identity plugin locations

GSM Specifications	UI Area	Plugin Name
All Specifications	Associated Specification	AssociatedSpecsIdentityPlugin
	Master Specification	MasterSpecsIdentityPlugin
	Related Labeling	PackingRelatedLabelingIdentityPlugin
	Originator field	OriginatorIdentityPlugin
	Supercedes field	SupercedesIdentityPlugin
Equipment Specifications	Related Packaging	EquipmentRelatedPackagingIdentityPlugin

Table 2–11 GSM identity plugin locations

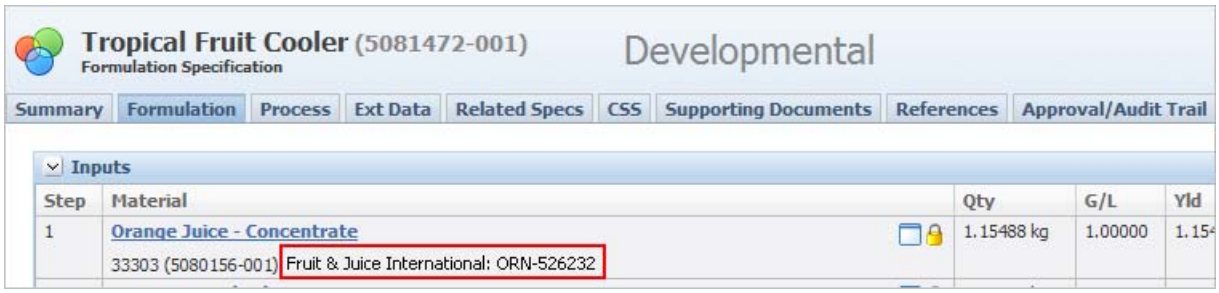
GSM Specifications	UI Area	Plugin Name
Formulation Specifications	Formulation Tab: Input Row	BOMInputItemPlugin
	Process Tab: Input Row	BOMInputItemPlugin
	Output PopUp: Composition Grid	BOMInputItemPlugin
	Alternate Output - Original Material	OriginalMaterialIdentityPlugin
	Alternate Input - Original Material	OriginalMaterialIdentityPlugin
	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
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 (Deprecated)	PackagingPrintedPkgMaterialIdentityPlugin
	Related Equipment	PackagingRelatedEquipmentIdentityPlugin
	Sub Component	PackagingSubComponentIdentityPlugin
	Substitute Material	SubStituteMaterialIdentityPlugin
Packing Configuration Specifications	Relates Specs	DeliveredMaterialPackingIdentityPlugin
Printed Packaging Specifications (Deprecated)	Parent Packaging Material	PrintedPkgRelatedParentPkgIdentityPlugin
	Substitute Material	SubStituteMaterialIdentityPlugin
Product Specifications	Alternate Global Standard	AltGlobalStandardIdentityPlugin
	Global Standard	GlobalStandardIdentityPlugin

Table 2–11 GSM identity plugin locations

GSM Specifications	UI Area	Plugin Name
	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 2–6 Standard input material display

Figure 2–7 Extended input material display



PQM Identity Plugins

For more information, see the *Agile Product Lifecycle Management for Process Product Quality Management Extensibility Guide*.

Possible Uses

1. Change the Object Icon for an NPD project.
2. Display the specification short name instead of the specification name.
3. Display all cross references versus just the user's cross reference preference.

4. Display the Supplier Item #s from all sourcing approvals attached to the material specification.

Technical Overview

Each 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>
```

All 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 [ProdikaHome]\Installer\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 ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins\Format Plugins

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 [ProdikaHome]\Installer\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.

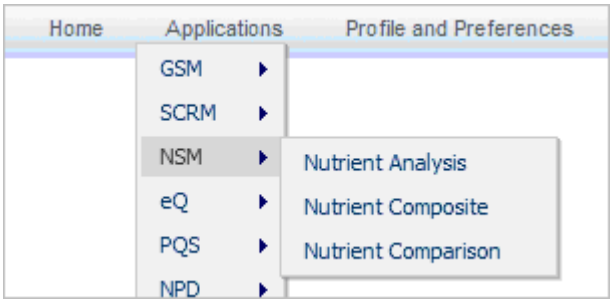
See the reference implementation in [ProdikaHome]\Installer\ReferenceImplementations\CalculationExtensions\SourceCode for implementation details.

Navigation Extensibility

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

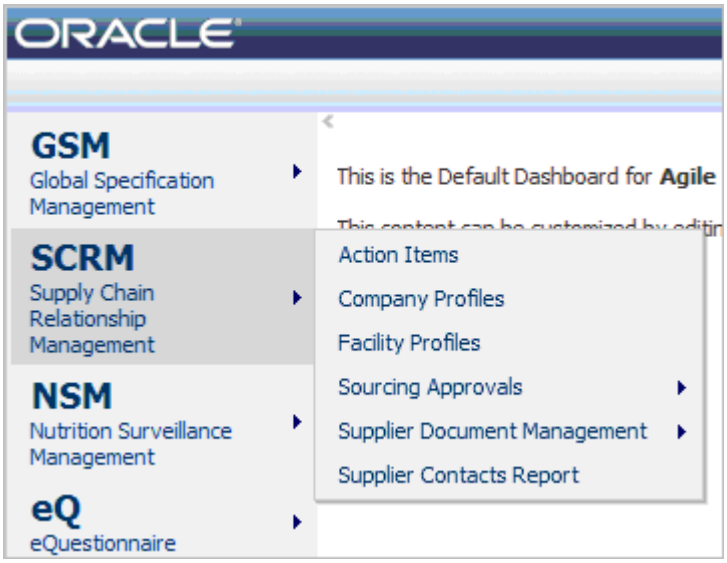
1. Platform Navigation—The navigation menu available in the top right of the browser window inside the suite header. This menu can be adjusted in the following ways:
   a. Add items
   b. Remove items
   c. Re-arrange items
   d. Apply visibility and security controls

Figure 2–8 Platform navigation



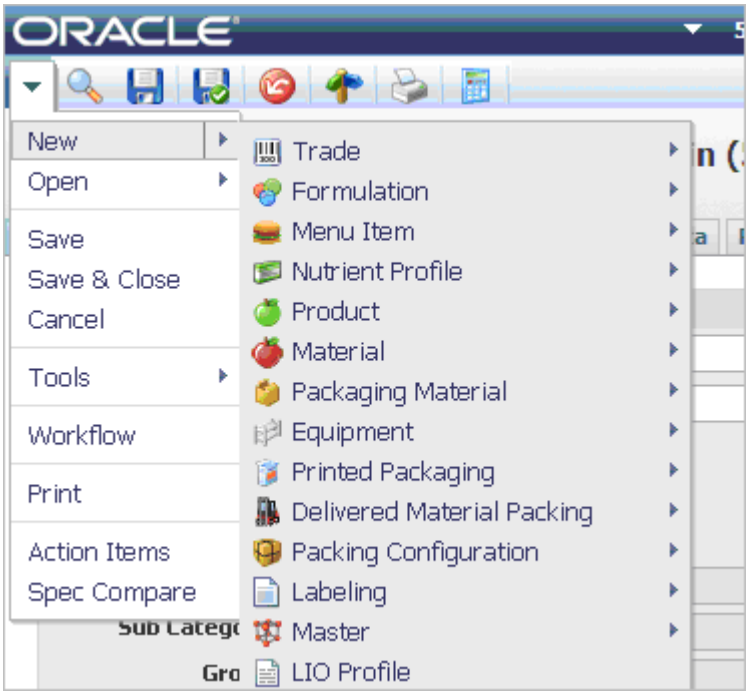
2. Portal Navigation—Available on the portal homepage listing in the left navigation panel. This menu can be adjusted in the following ways:
   a. Add items
   b. Remove items
   c. Re-arrange items
   d. Apply visibility and security controls

Figure 2–9 Portal navigation



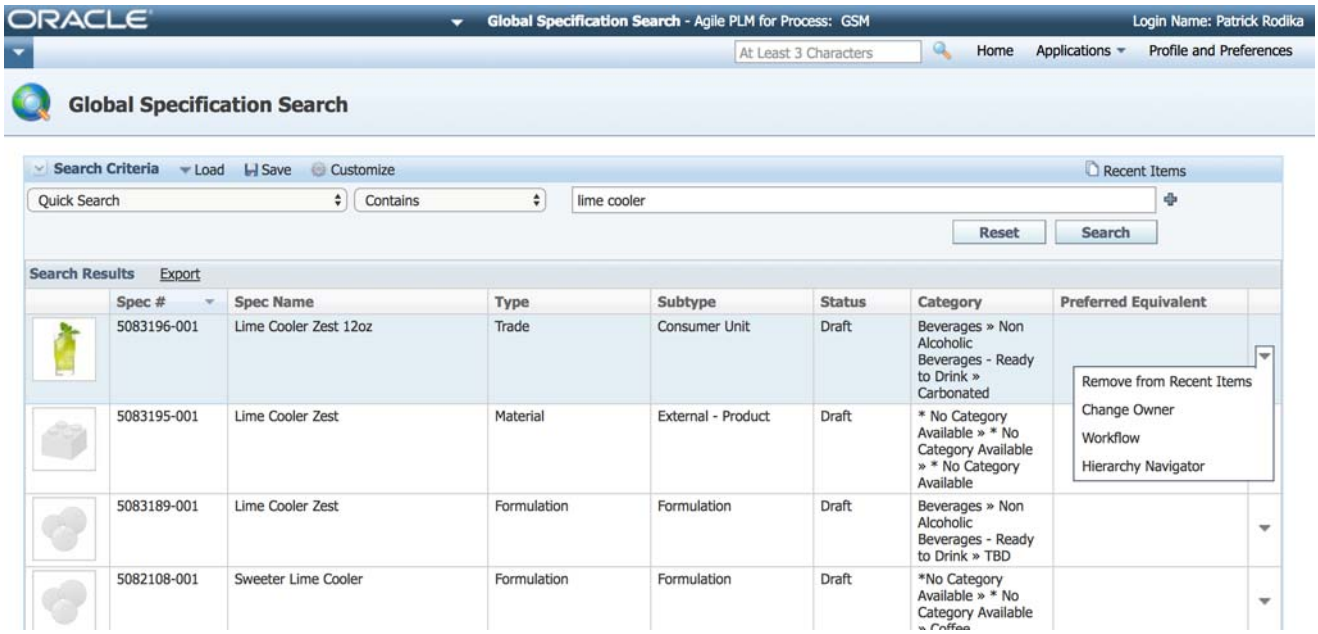
3. Action Navigation—Available in the top left corner of all objects. This navigation also includes the quick access icons. This menu can be adjusted in the following ways:
   a. Add menu items
   b. Remove menu items
   c. Add quick access icons
   d. Remove quick access icons
   e. Adjust hot keys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

Figure 2–10 Action navigation



4. Search Navigation—Available on the search pages. When selected, it opens the search results action menu, as shown below:

Figure 2–11 Search navigation, search results



PLM for Process is able to use the context of the search result row. For example, you can direct a user to a report and include the specification context in your URL to use as a report parameter.

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.
4. Add a link to the "Where Used" report on each material specification search result row. This would allow the user to investigate where the material is used without having to open each individual specification they wish to investigate.

Technical Overview

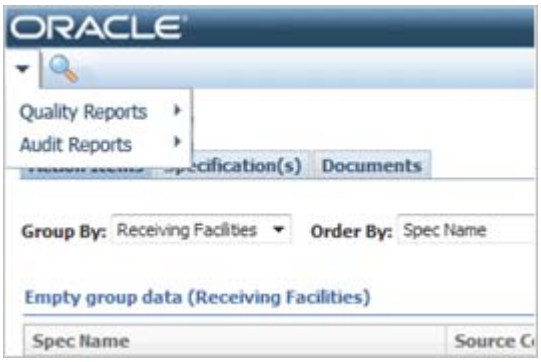
For more information, refer to the following the *Agile Product Lifecycle Management for Process Navigation Configuration Guide*.

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 2–12 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 2–13 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 2–14 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 [ProdikaHome]\Installer\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.

Product Portfolio Management Integration

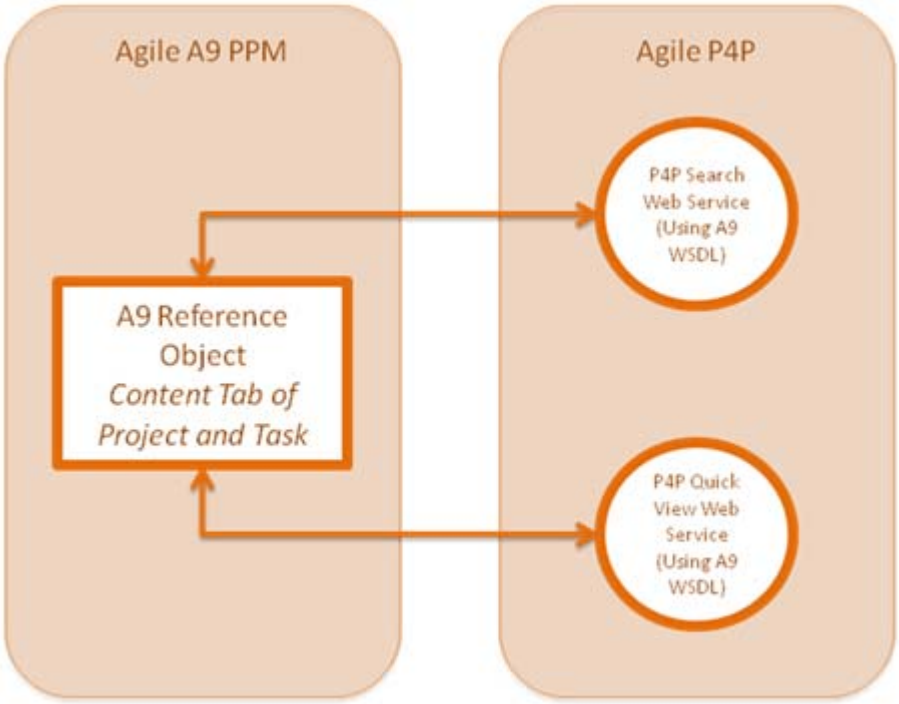
Agile PPM is a web-based application that enables users to manage all aspects of a project or program. PPM is fully integrated with the complete Agile PLM suite of products to maintain a centralized view of project records and associated product information within the organization. Executives use the PPM Dashboards to view portfolio data pertaining to all projects or programs. Portfolio data includes risks such as schedule slips, lack of resources, and project costs that directly contribute to the overall status of the project.

Use Cases

Let's examine three use cases demonstrating the PPM integration:

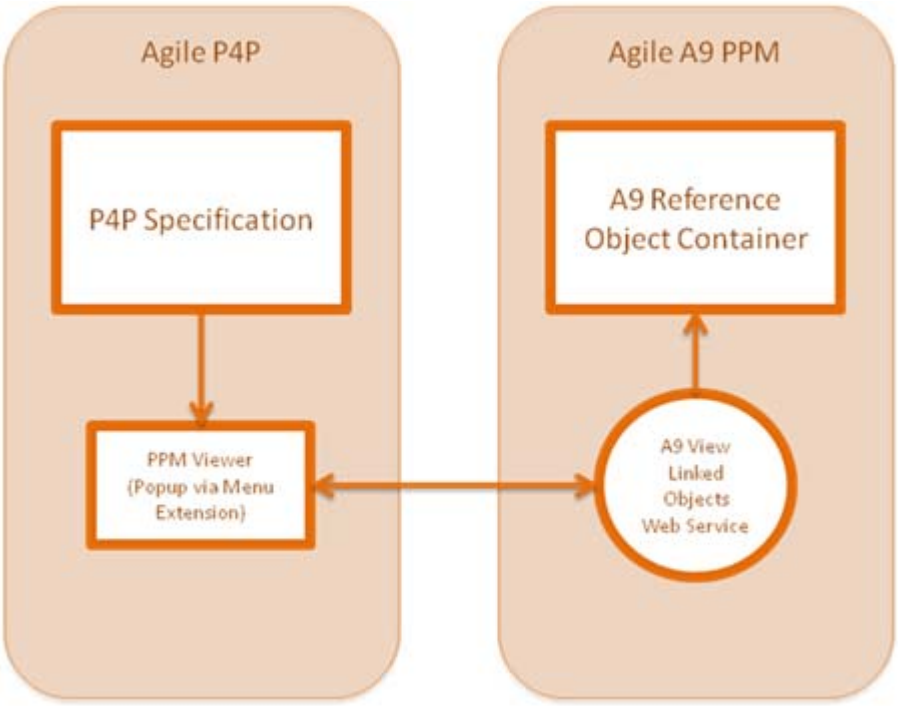
1. Adding a PLM for Process specification as an A9 reference object:

Figure 2–15 Adding a specification



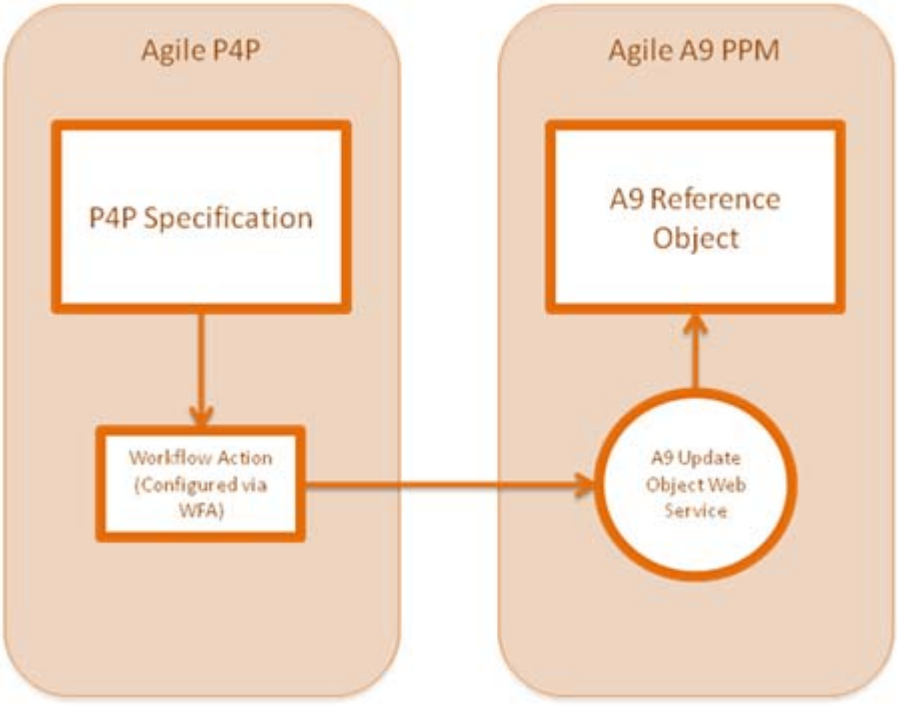
2. Viewing linked A9 objects from a PLM for Process specification:

Figure 2–16 Viewing a linked object



3. Updating reference object's status in A9 from a PLM for Process specification:

Figure 2–17 Updating a status



Supported Versions

The following versions of PLM for Process and A9 are supported:

- PLM for Process 6.2.3.x
- A9 9.3.5

Technical Documentation

Refer to the *Integration Guide* in [ProdikaHome]\Installer\Extensions\PPM_Integration\Documentation folder for more details.

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 2–12 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

Refresh Hierarchy Warning Plugin

As part of the Refresh Hierarchy feature, a warning icon appears next to the input or formulation specification that is required to refresh.

Technical Overview

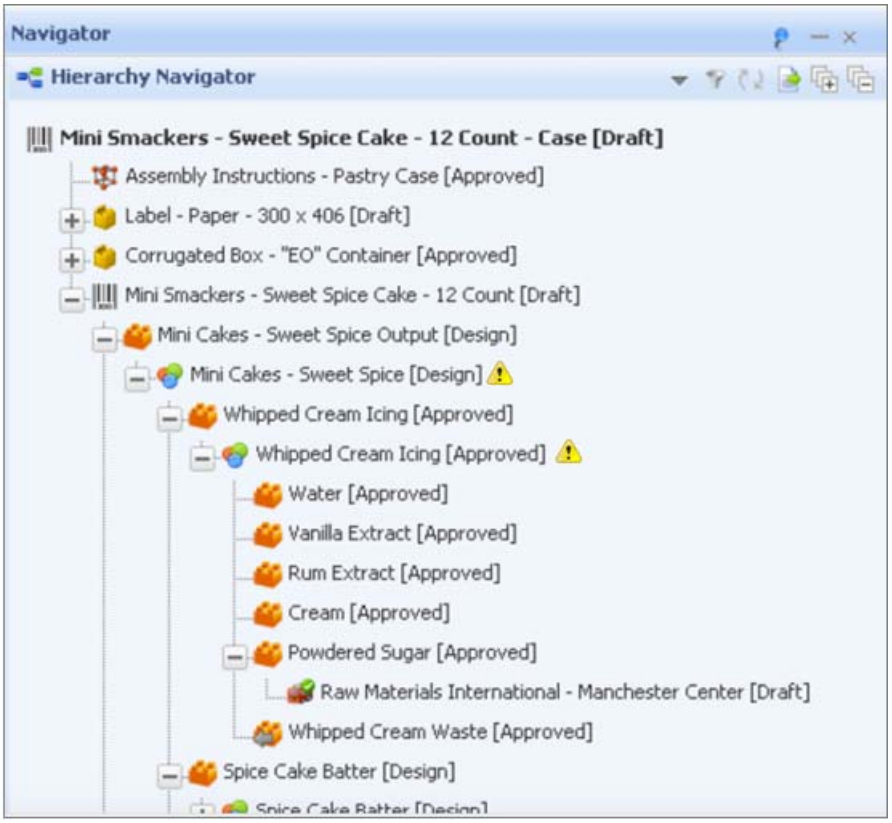
The Refresh Hierarchy Warning plugin extensibility point will call the PluginExtensions framework to check if a Validate plugin is configured for this extension point in the CustomPluginExtensions.xml file. If no plugin is configured, a default plugin is used that simply returns true and gives permission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

The Refresh Hierarchy Warning plugin is configured using the name FormulationRefreshHierarchyValidatePlugin.

Example CustomPluginExtensions.xml configuration:

```
<ValidatePlugins configChildKey="name">
    <Plugin name="FormulationRefreshHierarchyValidatePlugin"
    FactoryURL="Class:Xeno.Prodika.GSMLib.Security.Plugins.FormulationRefreshHierar
    chyValidatePluginFactory,GSMLib" />
</ValidatePlugins>
```

Figure 2–18 Warning icon



Rich Text Extensibility

Rich Text Extensibility is an extension point that allows customer to use their own Rich Text Editors. There are three types of rich text editors in PLM for Process: rich text editor with NLS, rich text editor without NLS, and rich text editor in Supplier Portal. Each of them is extensible and can be replaced with other supported Rich Text Editors.

Possible Uses

1. Every time a customer wants to use their own Rich Text Editor in Supporting Documents in GSM.
2. Every time customer wants to use their own Rich Text Editor with NLS for an ingredient statement in a material specification.
3. Every time customer wants to use their own Rich Text Editor in Supplier Portal.

Technical Overview

When using a new rich text editor, customers have to create a new javascript wrapper for the new rich text editor, and have to create a new provider class as well.

The javascript wrapper is created as a Closure for initializing the new rich text editor.

The Rich Text Editor provider is created as custom classes, packaged into a DLL, and added to the relevant web applications (web\gsm\bin, web\scrm\bin, and web\supplierPortal\bin). Customers have to configure the EditorProviders node in the config\Custom\EnvironmentSettings.config file to indicate their own Rich Text Editor providers.

Available Reference Implementations

Important: If the new rich text editor includes some Javascript libraries which are not compatible with PLM for Process, customers should research and implement an alternate solution.

For creating a new javascript wrapper, it should include the `_init` function to initialize the new rich text editor, functions for getting and setting values, handlers for event change or blur, etc.

For creating a new Provider for new Rich Text Editor, the new provider class must extend class `AbstractEditorProvider` and implement interface `IEditorProvider`.

Javascript Wrapper Example for the CkEditor

```
(function () {
    CkEditorMultiLingualEditor = function (config) {
        this.config = config;
        this.currentLanguage = config.currentLanguage.value;
        this.__proto__ = CKEDITOR.replace(config.clientID, config);

        this._init = function () {
            var myEditor = this;
            myEditor.on("change", function () {
                var pt = myEditor.getData();
                var maxLength = myEditor.config.maxLength || 3800;
```

```
                if (pt.length >= maxLength) {
                    var errMsg = window.__
prodikaValidate.TooLongError[parseInt(myEditor.config.currentLanguage.value.replac
e("lang", ""))];
                    alert(errMsg);
                    setTimeout(function(){myEditor.focus();},0);
                } else {
                    myEditor.config.multiLang ? myEditor.saveHTMLEx() :
myEditor.saveHTML();
                }
            });
            myEditor.switchToLang(myEditor.currentLanguage);
        }

        this.getLangControlID = function (langID) {
            return this.config.clientID + "_" + langID;
        }

        this.saveHTMLEx = function () {
            $("#" +
this.getLangControlID(this.currentLanguage)).val(this.filterHTML(this.getData()));
        }

        this.getAllLangValues = function () {
            var dict = {};
            for (var index in this.config.supportLanguage){
                var lang = this.config.supportLanguage[index];
                if (lang.value == this.currentLanguage)
                    dict[lang.value] = this.filterHTML(this.getData());
                else
                    dict[lang.value] = $("#" +
this.getLangControlID(lang.value)).val() || "";
            }
            return dict;
        }

        this.setLangValue = function (lang, value) {
            if (this.currentLanguage == lang)
                this.setData(value);
            $("#" + this.getLangControlID(lang)).val(this.filterHTML(value));
        }

        this.clearAllLangValues = function () {
            this.setData("");
            for (var index in this.config.supportLanguage) {
                var lang = this.config.supportLanguage[index];
                $("#" + this.getLangControlID(lang.value)).val("");
            }
        }

        this.filterHTML = function (html) {
            var filteredHTML = html;
            var regex = /^<span([\w\W]*?)>([\w\W]*?)<\/span>$/gi;
            var match = regex.exec(filteredHTML);
            if(match && match.length == 3){
                filteredHTML = "<p" + match[1] + ">" + match[2] + "</p>";
            }
            return filteredHTML;
        }
```

```
        this.switchToLang = function (langID) {
            this.setData($("#" + this.getLangControlID(langID)).val() || "");
            console.log(langID);
            this.currentLanguage = langID;
            $("#" + this.config.clientID + "_currentLanguage").val(langID);
        }

        this.getPlainText = function () {
            var stripHTML = /<\S[^><]*>/g;
            return this.getData().replace(/<br>/gi, '\n').replace(stripHTML,
'').replace(/^\s+|\s+$/g, '');
        }
        this._init();
    };
})();
```

Provider Class Example for the CkEditor

```
    public class CkEditorProvider : AbstractEditorProvider, IEditorProvider
    {
        public CkEditorProvider()
        {
        }

        void IEditorProvider.IncludeJS()
        {
            string editorJS =
@"/WebCommon/scripts/Ui/RichEditor/ckeditor/ckeditor.js";
            string editorWrapperJS =
@"/WebCommon/scripts/Ui/RichEditor/CkEditorMultiLingualEditor.js";

            if (Caller.GetType().BaseType.FullName ==
"Xeno.Web.UI.Common.Controls.Client.RichEditor.SimpleMultiLingualEditor")
            {
                editorWrapperJS =
@"/WebCommon/scripts/Ui/RichEditor/CkEditorMultiLingualEditor.js";
            }
            if (Caller.GetType().BaseType.FullName ==
"Xeno.Web.UI.Common.Controls.Client.RichEditor.RichEditorControl")
            {
                editorWrapperJS =
@"/WebCommon/scripts/Ui/RichEditor/CkEditorRichEditor.js";
            }


Caller.Page.ClientScript.RegisterClientScriptInclude(Caller.Page.Request.Applicati
onPath + editorJS, Caller.Page.Request.ApplicationPath + editorJS);

Caller.Page.ClientScript.RegisterClientScriptInclude(Caller.Page.Request.Applicati
onPath + editorWrapperJS, Caller.Page.Request.ApplicationPath + editorWrapperJS);

        }

        string IEditorProvider.Initializer(string config)
        {
            if (Caller.GetType().BaseType.FullName ==
"Xeno.Web.UI.Common.Controls.Client.RichEditor.SimpleMultiLingualEditor")
            {
                return @"new CkEditorMultiLingualEditor(" + config + ")";
```

```
            }
            if (Caller.GetType().BaseType.FullName ==
"Xeno.Web.UI.Common.Controls.Client.RichEditor.RichEditorControl")
            {
                return @"new CkEditorRichEditor(" + config + ")";
            }

            return @"new CkEditorMultiLingualEditor(" + config + ")";
        }


    }
```

Search Extensibility

While the standard application search behavior allows users to customize their experience individually, search extensibility allows you to extend this behavior. Using search extensibility you can do the following:

- Adjust the default top search criteria
- Adjust the default and available display columns
- Add new search criteria and display columns

Possible Uses

1. Most users search for specifications using Equivalent # instead of Spec Name. With just an XML configuration change you can make Equivalent # the default search criteria.
2. Most users would prefer to see Concepts as a default search result column.
3. There are a handful of distinct extended attributes that users would like to see in search result columns. You can add these extended attributes as new display columns.

Technical Documentation

Detailed documentation explaining search extensibility can be found in the *Agile Product Lifecycle Management for Process Search Extensibility Guide*.

.

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 [ProdikaHome]\Installer\ReferenceImplementations\SectionLevelEditing\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 [ProdikaHome]\Installer\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\ReferenceSectionLevelEditingExtensions\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
   ```

Side Bar

The PLM for Process user interface provides an extensible area for customers to include their own UI components, which are displayed alongside application pages. This Sidebar can host various components that can be launched by the user as needed. These components can display useful contextual information, such as a specific BI report based on the current specification, a historical event listing of object, and out-of-the-box Hierarchy Navigator, and more.

The PLM for Process Sidebar also provides helpful features, including easily selecting placement of the sidebar, launching a menu of components to display, and more. A set of helpful JavaScript functions are available to the sidebar components that can act on the sidebar's appearance and behavior.

For more information see the *Agile Product Lifecycle Management for Process Sidebar Extensibility Guide*.

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[ProdikaHome]\Installer\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 ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins for details.

PQM Veto Plugins

Basic PQM read, write, and workflow permissions for issues, actions, and audits 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 [ProdikaHome]\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.

Supporting Document Extensions

You can configure Supporting Document Extensions to get different functionalities and behaviors on Supporting Documents. Please see the *Agile Product Lifecycle Management for Process Configuration Guide* for more details.

Here is an example of how to enable the External URL configuration.

External URL Sample

The External URL configuration allows customers to punch out to their own systems and pull back a URL. For example, let's say customers have their own document management system or artwork project management system. They would put JavaScript in their systems' pages (like a search page). The user would click browse, it would open their system search page and the JavaScript they placed in their pages would create a URL that is sent back to our URL field.

Technical Documentation

Configuration Changes

In config\Extensions\SupportingDocConfig.config, add or modify the SupportingDocument node for your specification types, or other object types. The child node Document[@type="url"] is used for URL attachments.

The features and settings of the url document are as follows. (This table can also be found in *Agile Product Lifecycle Management for Process Configuration Guide*.):

Table 2–13 url document

id	Description
EditableURL	enabled: whether to enable the URL textfield to be editable as attachment
	factory: class used to determine enabled property dynamically
PassUserContextID	enabled: whether to enable passing the user token to the 3rd party site to login the external system.
	factory: class used to determine enabled property dynamically
ValidateProperURLFormatting	enabled: whether to enable the URL validator
	factory: class used to determine enabled property dynamically
External.URL	external system URL to the 3rd party site
External.UserToken	a token to be passed as a parameter when we are accessing 3rd party site
External.UserTokenParam	the name of the parameter representing user ID when we are accessing 3rd party site
URL.Display.Length	the maximum number of characters to be shown in URL path in toolbox doc and attachment grids

Taking a GSM activity (2283) as an example, the configuration node should be as follows:

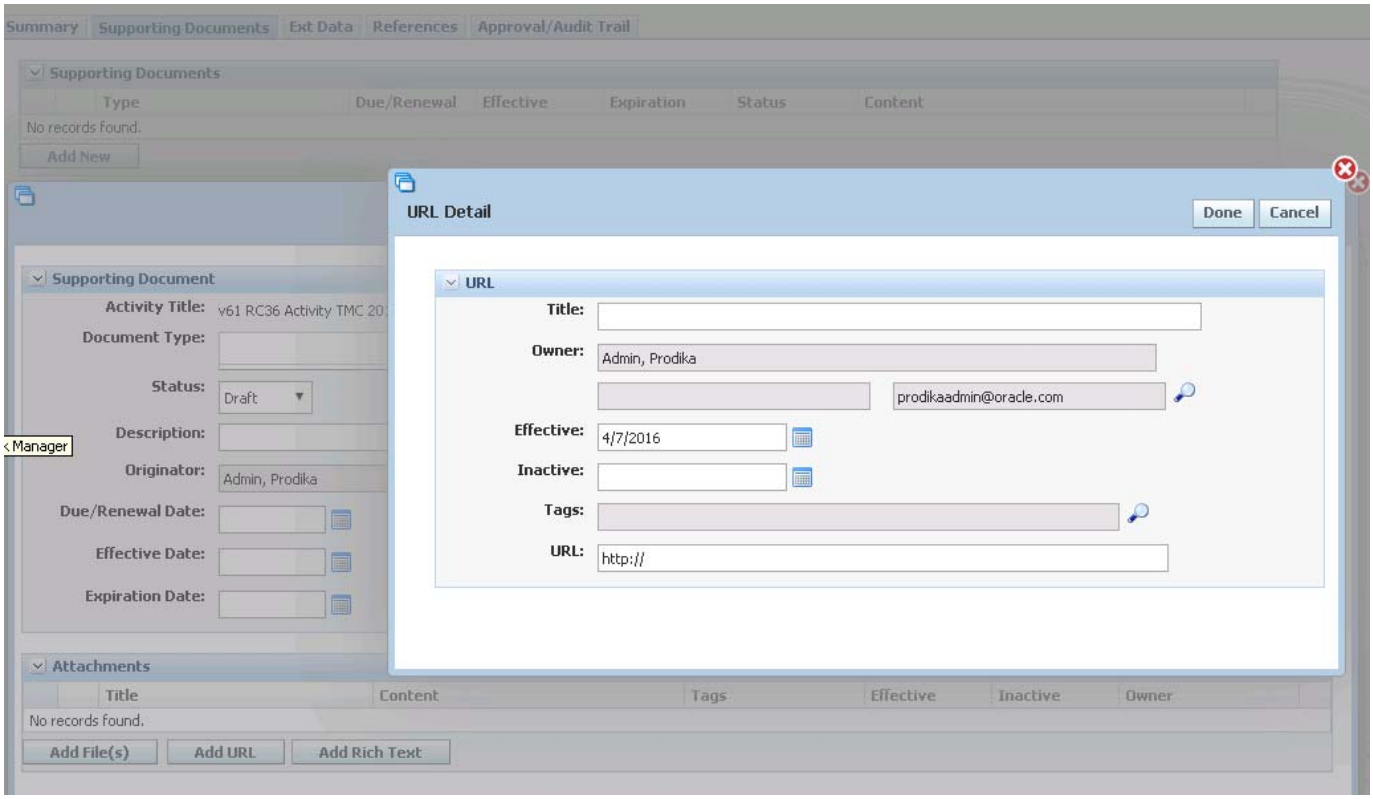
Figure 2–19 GSM activity sample

```
<SupportingDocument hostObjectType="2283">
  <Feature id="DocumentType" enabled="true" gridEnabled="true"></Feature>
  <Feature id="SecurityClassification" enabled="true" gridEnabled="true"
           factory="Class:…">
  </Feature>
  <Feature id="OLS_IfNoAccess_Hide" enabled="true"></Feature>
  <Document type="attachment" enabled="true" factory="">
    <Feature id="Version" enabled="true" gridEnabled="true" factory=""></Feature>
  </Document>
  <Document type="richtext" enabled="true" factory="">
    <Setting id="Keywords.MaxLength" value="500"></Setting>
  </Document>
  <Document type="url" enabled="true" factory="">
    <Feature id="EditableURL" enabled="true" factory="Class…"></Feature>
    <Feature id="PassUserContextID" enabled="true" factory="Class:…"></Feature>
    <Feature id="ValidateProperURLFormatting" enabled="true" factory="Class:…"></Feature>
    <Setting id="External.URL" value="http://slc01bga.us.oracle.com/ExternalURL.aspx"></Setting>
    <Setting id="External.UserToken" value="PLM4P"></Setting>
    <Setting id="External.UserTokenParam" value="UserID"></Setting>
    <Setting id="URL.Display.Length" value="50"></Setting>
  </Document>
</SupportingDocument>
```

- To enable External URL, 'External.URL' should be set to a page that contains the code which invokes the callback JavaScript method. The method name is passed from query string field CallBackFunc.
- If the opener system needs to be verified, please turn on PassUserContextID. The user token will be added to the URL, like *&UserID=PLM4P*.

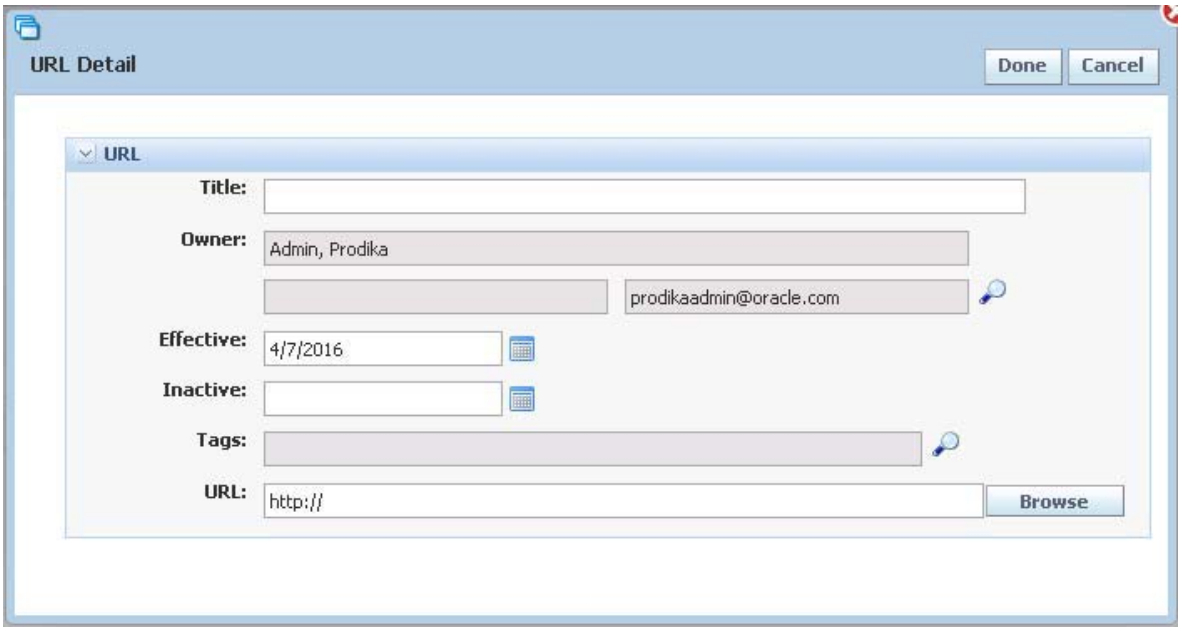
If 'External.URL' is empty, the URL Detail is as follows:

Figure 2–20 URL Detail, 'External.URL' is empty



If 'External.URL' has a value, there would be a **Browser** button next to the URL input. The external page will be opened when this button is clicked.

Figure 2–21 URL Detail,'External.URL' has a value



External URL Page Changes

In the External URL page, invoke the callback method with JavaScript.

Figure 2–22 External URL page

```
<script type="text/javascript" src="/NPD/WebCommon/scripts/jquery/1.5.1/jquery-1.5.1.min.js"></script>
<script type="text/javascript" src="/NPD/WebCommon/Scripts/YDialog.js"></script>
<script type="text/javascript">
  function returnUrlToPLM4P(){
    YDialog.opener().<%=Request["CallBackFunc"]%>("http://www.oracle.com", "oracle");
    YDialog.close();
  }
</script>
```

- Include jquery and YDialog JavaScript files from the PLM4P application into the page.
- Use YDialog.opener() to refer to the opener window.
- Use query string field CallBackFunc to refer to the callback method, and the first parameter is the URL to be returned.
- Use YDialog.close() to close the page.

> **Note:** This page should be under the same domain of PLM4P applications, or the callback method cannot be invoked in modern browsers. If cross domain calling is necessary, please contact Oracle Support for help.

User Interface Extensions

Clients wishing to customize specific user interface behavior, such as marking certain fields as read only, or applying formatting to show required fields, as well as other front-end customizations, will be able to do this using the User Interface Extensions (UIE) feature. Clients can write simple Javascript code which then gets pulled into the desired user interface pages. The Javascript code can leverage useful new Javascript variables and functions that make it easier to perform user interface manipulation, as well as access specific data from the given item, such as a specification's status, the user's groups, etc.

Out-of-the-box UIE extension points are now available for most major objects in the system, including GSM specifications, SCRM companies, facilities, and sourcing approvals, PQM actions, issues, and audits, NPD objects, and more.

Technical Overview

The extensibility points will call Format Plugins to return Javascript code to the page as part of the front end rendering. To add the extensibility point, a simple single-line code statement can be inserted into a page (.aspx) or control (.ascx) code-in-front. The Format Plugin orchestrates the retrieval of core and client specific Javascript files, along with helper classes that generate useful script variables and functions. Clients can write customizations in Javascript leveraging commonly used functions and variables from separate Javascript files and some .NET Reflection to extract some properties from the current business object.

Additionally, clients can write their own C# classes that generate their own Javascript variables and functions, if desired.

Technical Documentation

See the User Interface Extensions.doc document in the \Utilities\UIExtensions folder for more information.

Watch the introductory video on the User Collaboration site for a high level overview.

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
- UGM users and groups

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

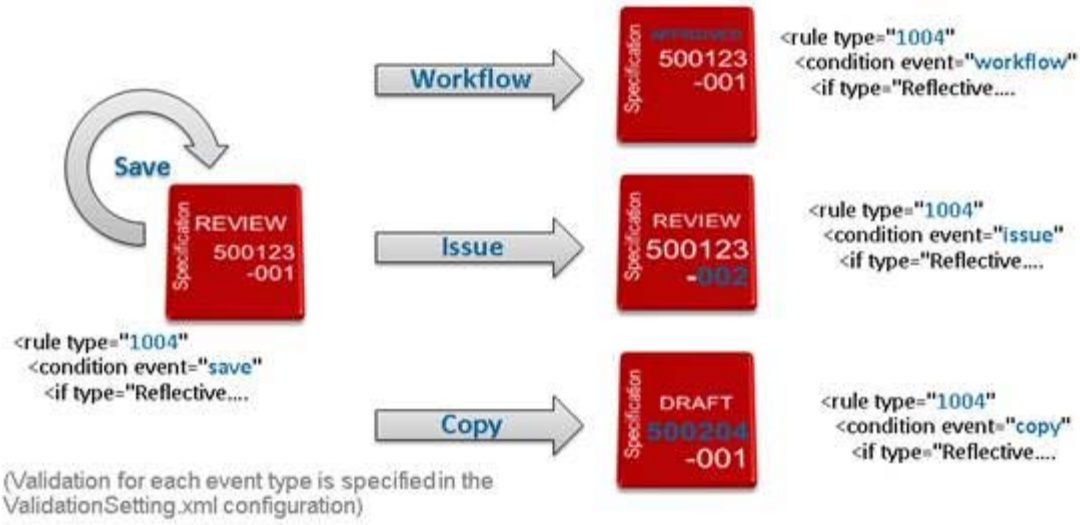
Possible Uses

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.

Figure 2–23 Validation classes



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>
```

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, along with the corresponding error messages, which are then displayed to the user.

In an effort to improve the process of writing custom validation, decrease the development and testing time, and simplify configuration and installation, a new Validator has been created that leverages a script-based approach to validation. The ScriptBasedValidator uses a JavaScript Interpreter engine to run simple and more intuitive validation code that is easier to write and requires no class compilations. Customers can write straight-forward JavaScript code to perform server-side validation, while leveraging existing helper methods and properties. For instance, helper methods such as GetEA(<eaID>) will return the desired Extended Attribute (if it exists).

See the Script-based Validation Guide for details.

Default Language

UGM Admins can now set their default language to any NLS approved language using the default validation framework plugin that is required when managing groups. This plugin auto populates an English group name on save if one does not exist. Turn on the configuration in Prodika\config\Extensions\CustomPluginExtensions.xml:

```
<Plugin name="PreSaveUGMPlugin" ignoreInheritFromPluginName="true"
FactoryURL="Class:Xeno.Prodika.Services.PrincipalManagementService.Plugins.UGMChan
geGroupPluginFactory,PrincipalManagementService" />
```

```
<Plugin name="PreWorkflowUGMPlugin" ignoreInheritFromPluginName="true"
FactoryURL="Class:Xeno.Prodika.Services.PrincipalManagementService.Plugins.UGMChan
geGroupPluginFactory,PrincipalManagementService" />
```

Technical Documentation

Detailed technical training of the Validation Framework is available in the [ProdikaHome]\Installer\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 [ProdikaHome]\Installer\ReferenceImplementations\WorkflowActions\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.

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

1. SpecStatusChangeLogger—Logs workflow status changes, along with specification identifier information, to a file.
2. We offer several out of the box workflow actions that will allow you to workflow multiple specifications at once. Learn more in the following section, Workflow Actions—Automatic Workflow.

Workflow Actions—Automatic Workflow

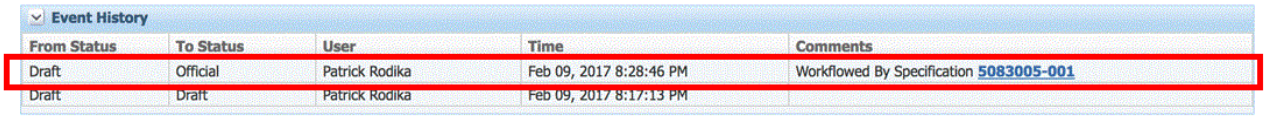
You can configure WFA workflow templates to automatically workflow child specifications or quality issues using automatic workflow. These reference implementations allow you to automatically workflow GSM specifications or PQM issues directly associated to the parent. You determine when they are enacted by selecting the workflow action inside the Transitions section of a WFA template.

In the example below, every time this GSM activity transitions to "Approved", the Related Items associated to that activity will transition to "Official".

Figure 2–24 GSM activity WFA template: Transitions section

All child specifications transitioned will have an entry added to the Event History section on the Approval/Audit Trail tab. The specification (or GSM activity) responsible for transitioning the specification will be indicated and linked.

Figure 2–25 GSM specification: Event History section



Important: Make sure your child workflow templates are able to auto resolve to owners, signature requests, and notification participants. The user moving the parent object will not have the opportunity to select participants for every possible child specification being transitioned.

The following relationships are supported:

Parent	Child	Workflow Action Name
Trade	Material Specification (Reference Output)	GSMWorkflowActionUpdateReferencedOutputMaterialForTrade
	Material Specification (Owned Output)	GSMWorkflowActionUpdateMaterialForTrade
	Lower Level Trades	GSMWorkflowActionUpdateLowerLevelSpecForTrade
	Packaging Specifications	GSMWorkflowActionUpdatePackagingSpecForTrade
	Sourcing Approval	GSMWorkflowActionUpdateSACForTrade
Formulation	Material Inputs	GSMWorkflowActionUpdateMaterialInputForFormula

Parent	Child	Workflow Action Name
Reference Output	Produced By Formulations	GSMWorkflowActionUpdateContextForMaterial
Packaging	Packaging Sub Components	GSMWorkflowActionUpdateSubComponentForPkg
	Sourcing Approval	GSMWorkflowActionUpdateSACForPkg
GSM Activity	Related Items	GSMWorkflowActionUpdateRelatedItemForActivity
Material	Material Breakdown Components	GSMWorkflowActionUpdateBDComponentForMaterial
	Sourcing Approval	GSMWorkflowActionUpdateSACForMaterial
Product	Sourcing Approval	GSMWorkflowActionUpdateSACForProduct
Equipment	Sourcing Approval	GSMWorkflowActionUpdateSACForEquipment
Associated Specs	Associated Specs	GSMWorkflowActionUpdateAssociatedSpecs
PQM Action	PQM Issues	PQMWorkflowActionUpdateRelatedIssuesForAction

Possible Uses

1. When a GSM Activity resolved to the WFA template "New Packaging Assembly" transitions to "Approved", workflow the packaging specifications in the Related Items section to "Approved".
2. When a Trade Specification resolved to WFA template "Baked Goods" transitions to "Inactive", workflow the associated material specification and owning formulation to "Inactive".
3. When a Packaging Specification resolved to WFA template "Packaging Assembly" transitions to "Ready for Review", workflow the associated Sub Components to "Ready for Review". Do not move packaging specifications already in "Approved".
4. When a Quality Action resolved to WFA template "Corrective Action" transitions to "Cancelled", workflow the associated Quality Issues to "Cancelled".

Technical Overview

Automatic workflow actions are made available by un-commenting them from CustomWFAExtensionsConfig.xml in config\Extensions.

The following attributes are available for Automatic Workflow actions:

Name—The value specified here will be the workflow action name displayed in WFA transitions.

ProcessTemplateType—Specifies which workflow process type can use it. Automatic workflow actions are supported in GSM and PQM.

IncludeObjectIDs—Indicates which kind of parent specification can use this workflow action.

FactoryURL—The object loader URL of the workflow action class, followed by the assembly name.

Example:

```
Class:Xeno.Prodika.GSMLib.WorkflowActions.GSMWorkflowActionUpdateRelatedItemForActivity,GSMLib$4|1|1004
```

Variables behind FactoryURL—These variables determine the specific behavior of the workflow action.

Example:
`Class:Xeno.Prodika.GSMLib.WorkflowActions.GSMWorkflowActionUpdateAssociatedSpecs,GSMLib$4|1|1004`

These IDs are separated by a vertical bar (|) and represented in the following order:

- **Child Transition Status**—This represents the status the child specification will be moved to. This is the behavior ID of the workflow tag added to the status definition on the child's WFA template. The example above shows 4. This represents the "Is Approved" tag which was added to the "Approved" status of the child's workflow template.
- **Status to Ignore**—This represents the status that will be ignored if the child specification is already in that state. The example above shows 1. This represents the "Hide Specs" tag which was added to the "Archived" status of the child workflow.
- **Spec Types to Transition**—Represents the child spec type that will be moved. Use ',' for multiple spec types; for example 1004,1009,2147 = material, packaging, trade. In the example above, only material specifications (1004) will be transitioned to "Approved".

Here's an example configuration key. In this example, the workflow action is advancing associated material specifications to "Approved" (WFA Tag behavior ID=4) when the parent trade specification advances to "Approved". The action will not advance associated material specifications in an Archived status (WFA Tag behavior ID=1).

```
<WorkflowAction processTemplateTypes="GSM"

IncludeObjectIDs="2147"

name="Trade Spec Workflow Action - Update Associated Specs"
FactoryURL="Class:Xeno.Prodika.GSMLib.WorkflowActions.GSMWorkflowActionUpdateAssociatedSpecs,GSMLib$4|1|1004"

></WorkflowAction>
```

Technical Documentation

See the [ProdikaHome]\Installer\ReferenceImplementations\WorkflowActions\Documentation folder for more details.

Workflow UI Extensions

Hierarchy Navigator allows you to add workflow UI extensions. Workflow actions are available for GSM specifications and SCRM sourcing approvals.

Possible Uses

1. You can workflow an object anywhere configured without opening the object. For example, on the Formulation tab, add a workflow action for referenced output materials.

Technical Overview

Creating a Workflow Action

Complete the following steps for adding a new workflow action. The user can right click on a node to get the menu item configured.

1. Add an entry to Prodika\Web\GSMExtensions\scripts\toggler.js.

```
function PopupWorkflowAction(pkid) {
var url =
"/gsm/PopUps/Workflow/frmWorkflowActionPopup.aspx?MaintainSpec=true&DataSource=
Class:Xeno.Prodika.GSMLib.WorkflowCommon.DataSources.ResolutionSetAwareSpecLine
arWorkflowPopupDataSource,GSMLib$SpecID=" + pkid + "&IsSigDoc=false";
YDialog.open( url, "WorkflowPopUp",
"height=550,width=550,status=no,toolbar=no,menubar=no,location=no,dependent=yes
,scrollbars=yes" );
}
```

2. Add the following entry to Prodika\Web\GSMExtensions\Formulation\Process\ctlFormulationOutputs.asc x, after the formulation tag icon node.

```
<asp:Panel id="pnlWorkflow" style="display:inline" runat="server"
Visible="<%#((FormulationOutputRowModel)Container.DataItem).IsReferenced%>">
<td>
<a href="#"
onclick="javascript:PopupWorkflowAction('<%#((FormulationOutputRowModel)Contain
er.DataItem).OutputItem.Material.PKID%>');"><img id="imgWorkflow"
src='../images/navmenu/Action-Workflow.gif'/></a>
</td>
</asp:Panel>
```

Notes:

- Workflow actions cannot be applied to specifications which do not support workflow behavior, or for internal output materials and external output materials. Please remember to add visible condition like `visible="<%#((FormulationOutputRowModel)Container.DataItem).IsReferenced%>"` to avoid abuse.
- The popup URL can be changed via workflow objects. If the current workflow object is a specification, the popup URL should be:

```
"/gsm/PopUps/Workflow/frmWorkflowActionPopup.aspx?Maintain
Spec=true&DataSource=Class:Xeno.Prodika.GSMLib.WorkflowCom
mon.DataSources.ResolutionSetAwareSpecLinearWorkflowPopupD
ataSource,GSMLib$SpecID=" + GSM.SpecificationPKID +
"&IsSigDoc=false"
```

Otherwise, if the current workflow object is a souring approval, the popup URL should be:

```
"/scrm/PopUps/Workflow/frmWorkflowActionPopup.aspx?Maintai
nSpec=true&DataSource=Class:Xeno.Prodika.SCRM.WorkflowComm
on.DataSources.SACLinearWorkflowPopupDataSource,SCRMLib$En
tityID=" + SCRM.SouringApprovalPKID +
"&IsSigDoc=false&hdnActiveSACType=SPEC";
```

3. Navigate to the target formulation specification. You can see the additional workflow icon displayed for the referenced output materials.
4. Click the workflow icon on the Formulation tab. A workflow popup window appears with the Spec Name and Spec #. Now you can workflow the specification to any status you prefer.

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%';
```