Siebel ATG Implementation Guide

Product Configurator User Interface Generation

Implementation Guide for Product Configurator User Interface Generation.

Table of Contents

Introduction

This Implementation Guide (IG) is provided as documentation for the Product Configurator User Interface(UI) Generation Reference Implementation (RI). The RI enables administrators to automatically generate a UI for products imported into the ATG product catalog, to be used with Product Configurator (RI).

The RI builds on the Product Configurator RI which stored UI elements in an ATG repository, which were previously supplied as test data. This RI will dynamically produce UI elements based on the products imported from Siebel into the ATG product catalog.

There are no changes required to Siebel for this integration. The integration uses existing Siebel web services, available as of Siebel release 8.1.1.8 FP.

The integration builds on the previous integrations for Accounts, Product Catalog, Orders and Product Configurator.

References

[1] Siebel ATG Implementation Guide – Order Integration v1.0.

[2] Siebel ATG Implementation Guide - Product Data Integration for Siebel Innovation Pack 2012 v1.0.

[3] Siebel ATG Implementation Guide - Product Configurator Integration 2012 v1.0.

[4] Siebel CRM Web Services Reference Version 8.0 & Version 8.1.

[5] ATG Programming Guide Version 10.2.

[6] ATG Commerce Programming Guide Version 10.2.

[7] ATG Repository Guide Version 10.2.

Note: References 5, 6 and 7 give more details on basic ATG concepts. The reader should have a good understanding of these basic concepts such as, components, repositories, and pipelines, before reading this guide.

Software Dependencies

This guide is based on the following software and the respective versions:

- Oracle Siebel Version 8.1.1.10.
- Oracle ATG Web Commerce Version 10.2

Technical Requirements Summary

This chapter gives a brief overview of the high level requirements for the use of Siebel Product Configurator UI Generation within Oracle ATG Web Commerce.

Product Configuration UI Generation Requirements

The product configurator UI generation supports the following requirements:

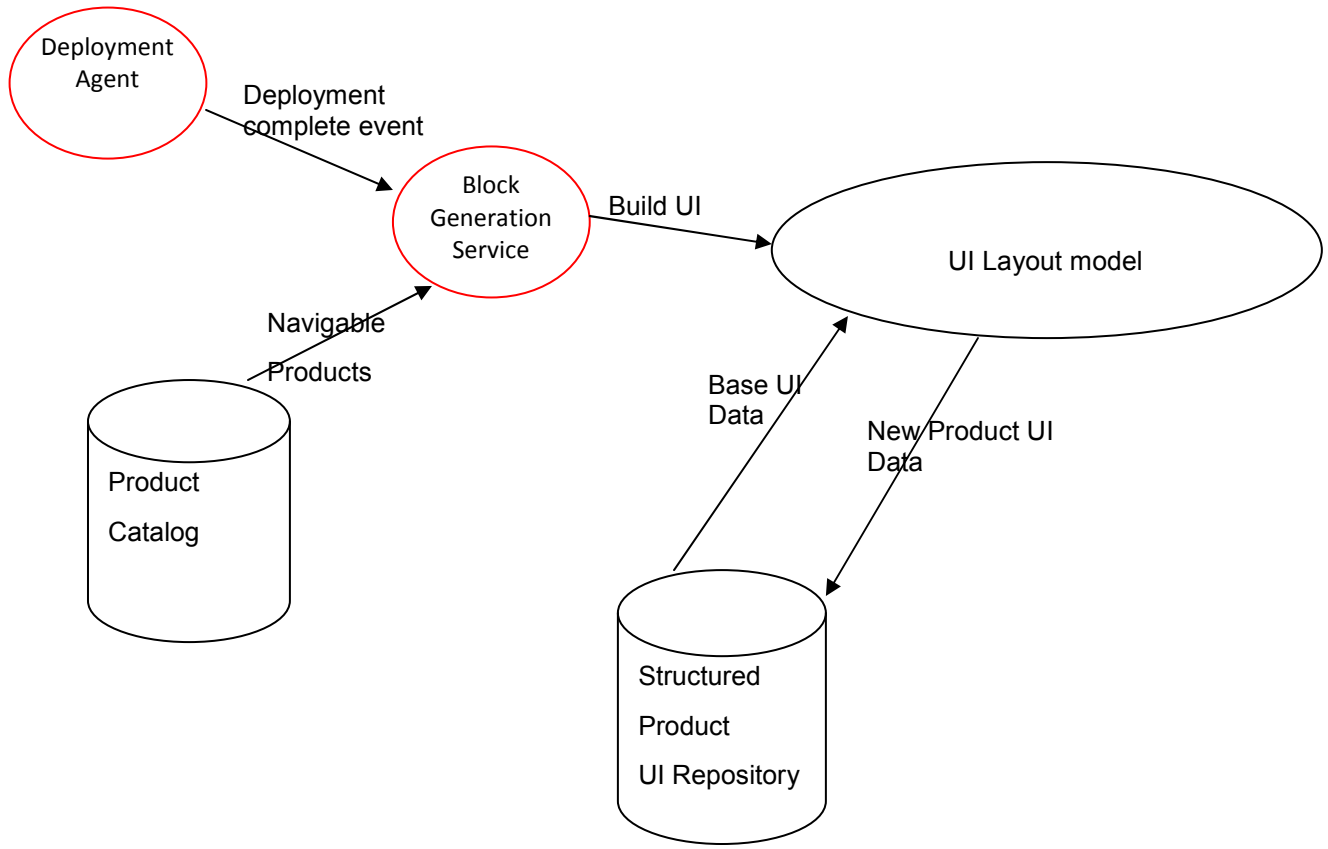
- Dynamically generate UI data for the Structured Product UI (SPU) Repository based on products which are imported into the ATG Product Catalog from Siebel.
- Distinct ProductUIs will be created for a standalone product, and for each Promotion and Simple Product Bundle it appears in.
- If data already exists in the SPU, and a change to the product or relationship are imported into the ATG Product Catalog, the data will be updated to reflect this.
- Administrators will be able to define default CSS styling based on Siebel Configurable Product and Promotion structure.

Product Configurator User Interface Generation Overview

This chapter gives an overview of all of the elements that make up the Siebel product configurator UI generation. The overview describes all of the basic flows, starting from importing products into the Product Catalog, and examples of how to customize how the UI appears based on properties and structure of the imported products.

Product Configuration User Interface Generation Processes Overview

In this guide, we will assume that products have been correctly imported into the Product Catalog, and the catalog has been successfully deployed to production. The process will begin when deployment has completed by calling into the buildProductUIs method of the BlockGenerationService.



Base Data

Provided with the reference implementation is some base data, consisting of content Blocks, which, during the UI Generation, will be copied and saved as new instances, and renderers, which will be assigned to Attribute and Relationship blocks. The renderers will be those described in the Product Configurator Integration Implementation Guide.

For base data, see appendix.

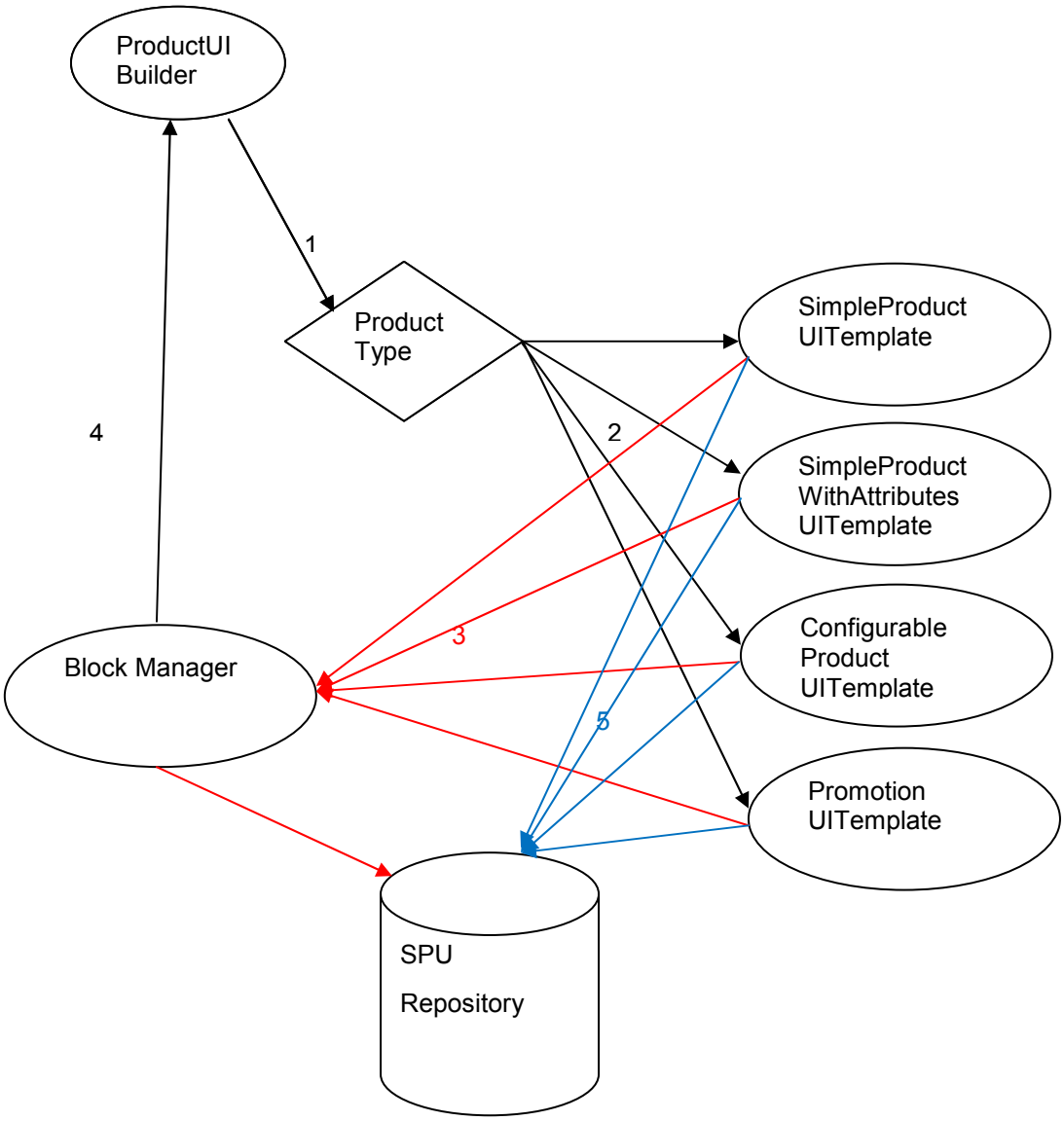
Block Generation Service

This component has a method buildProductUIs which is called to produce Product UIs for every Navigable Product in the Product Catalog. A Navigable Product is one which is referred to by a category. Therefore, if a product exists in the Product Catalog which is not referred to by a category, a Product UI will not be automatically generated for it.

UI Layout Model

The layout model consists of a hierarchy of components and associated classes used to model the hierarchical nature of the Configurable Product and Promotion structure. The entry point for generating a product UI is the buildProductUI method of the ProductUIBuilder class. This method accepts a SiebelCatalogProduct object, creates Product UI repository Item with associated Blocks and saves them in the StructuredProductUI Repository.

Basic Model:



1. The ProductUIBuilder determines the type of the SiebelCatalogProduct provided. It then determines which UITemplate to use based on a map of configured UITemplate components, keyed on Product Type.

2. The ProductUIBuilder will call the build method on the UITemplate.
3. The UITemplate will then delegate to the Block Manager to generate the Blocks needed to represent the UI of the product. In the Case of Simple Products With Attributes, the UITemplate will delegate to the Block Manager to create Attribute Blocks for each attribute on the SiebelCatalogProduct.
4. In the case of a Configurable Product or a Promotion, the UITemplate delegate to the Block Manager to create Relationship Blocks. To create Relationship Blocks, the Block Manager will recursively call the ProductUI builder, in order to create Product UIs for Child Products.
5. The UITemplate will set the appropriate properties on the Product UI repository item and save everything to the StructuredProductUIRepository

ProductUIBuilder

The ProductUIBuilder component is the entry point for building a ProductUI for a SiebelCatalogProduct. It has a map of UITemplates, keyed on product type. And example configuration would be:

```
UITemplates=\DEFAULT=/atg/siebel/configurator/spu/generation/template/SimpleProductUI
Template,\
  Simple Product With
Attributes=/atg/siebel/configurator/spu/generation/template/SimpleProductWithAttribut
esUITemplate,\
  Configurable
Product=/atg/siebel/configurator/spu/generation/template/ConfigurableProductUITemplat
e,\
  Promotion=/atg/siebel/configurator/spu/generation/template/PromotionUITemplate
```

Clients of this component will call into the buildProductUI method, passing in the target SiebelCatalogProduct as a parameter. The method will then get the appropriate UITemplate for the product based on the product type. The method will then call into the buildUI method on the UITemplate.

BlockManager

The BlockManager component deals with the creation and update of Blocks between the UITemplates and the SPU repository. The BlockManager deals with the tree types of blocks; Blocks (The base type for the block repository item, used for displaying content), Attribute Blocks (blocks for displaying attributes), and Relationship blocks (blocks for displaying relationships, which also have a reference to the child product UIs the relationships contain).

The BlockManager is also responsible for creating unique block repository Ids.

Blocks

The createContentBlock method is provided to create new Block.

```
createContentBlock(String pProductId, String pBaseBlockId,
      ProductUIBuilder pProductUIBuilder)
```

The method process is:

1. Look up the base block repository item from the StructuredProductUIRepository
2. Build a new ID for the block repository item. This is a concatenation of the Product ID and the Base Block Id. *
3. Set the new ID on the base block.
4. Save the block to the StructuredProductUIRepository as a new block item.

*This has the effect of making a block unique to each product UI. For example, if you wanted two “div” content blocks in your Product UI, two base blocks would have to be created for them, each pointing at the same renderer, eg

```
<add-item item-descriptor="block" id="markup-div-start-block-1">
  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>
</add-item>
<add-item item-descriptor="block" id="markup-div-start-block-2">
  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>
</add-item>
```

These should be added to the list of blocks in the UITemplate (see UITemplate section), eg:

```
blocks=heading-block, markup-div-start-block-1,product-name-block, markup-div-start-block-2, product-description-block.
```

This will mean 2 distinct “div” blocks will be generated, with IDs:

Product-ID-markup-div-start-block-1

Product-ID-markup-div-start-block-2

Attribute Blocks

When creating an Attribute block, the Block manager will decide which renderer to use based on the Attribute’s data type. A map of renderers will be stored in the components configuration, keyed to different data types, eg:

```
attributeDataTypeRenderers=
\ENUMERATED=product-attribute-select,
\INTERVAL=product-attribute-text,
\EXCLUSIONS=product-attribute-text
```

A method is provided for the creating and updating of an Attribute Block:

```
public Block createAttributeBlock(String pProductId, SiebelCatalogAttribute pAttribute,
      ProductUIBuilder pProductUIBuilder)
```

This process of the method is:

1. Create the ID for the Attribute Block. This is a concatenation of the Product Id, the SiebelCatalogAttribute's OrigID, and the configured Attribute Block Suffix.
2. If the block does not exist in the repository, create a new Block object, and set the ID, the name of the block (also the id), the active flag to true, the Attribute ID to the SiebelCatalogAttribute's OrigID, and the type to ATTRIBUTE_BLOCK.
3. Set the Attribute Name and the Display Name from the SiebelCatalogAttribute. This will also be performed if the Attribute Block already exists in the Repository, meaning the information on the Block will be kept up to date with any changes on the attribute.
4. Determine which renderer to use, by using the attributeDataTypeRenderer map with the SiebelCatalogAttribute's domainType. Set the renderer on the Block.
5. Delegate to the SPUTools component to either save the Block as new or update an existing one.

Relationship Blocks

The renderers for Relationship Blocks are determined by the nature of the relationship they represent; the cardinality and the number of domain products. To decide which to use, a list of RelationshipRendererRules are configured. A RelationshipRendererRule is the supertype of a class which has an abstract method, determineRenderer:

```
public abstract String determineRenderer(SiebelCatalogRelationship pRelationship)
```

The BlockManager will have a configured list of RelationshipRendererRule types, each corresponding to a different relationship permutation eg:

```
relationshipRendererRules=\
  /atg/siebel/configurator/spu/generation/rules/SingleChoiceSingleSelectRule,\
  /atg/siebel/configurator/spu/generation/rules/SingleChoiceMultiSelectRule,\
  /atg/siebel/configurator/spu/generation/rules/SingleChoiceOptionalSelectRule,\
  /atg/siebel/configurator/spu/generation/rules/MultiChoiceSingleSelectRule,\
  /atg/siebel/configurator/spu/generation/rules/MultiChoiceMultiSelectRule,\
  /atg/siebel/configurator/spu/generation/rules/MultiChoiceOptionalSelectRule
```

Each of the implementing classes will define the determineRenderer method, which will return a renderer for a different scenario. When creating a RelationshipBlock, the Block manager will iterate through these components until finds the renderer for the relationship in question. The rules for determining the renderers are:

Rule Name	Rule	Mapped Renderer
SingleChoiceSingleSelectRule	Child Products = 1 Minimum Cardinality = 1 Maximium Cardinality = 1	rel-man-single-select-single-choice-text

SingleChoiceMultiSelectRule	Child Products = 1 Maximum Cardinality > 1	rel-opt-multi-select-single-choice-button
SingleChoiceOptionalSelect	Child Products = 1 Maximium Cardinality = 1 Minimum Cardinality < 1	rel-opt-single-select-single-choice-check
MultiChoiceSingleSelectRule	Child Products > 1 Minimum Cardinaltiy = 1 Maximum Cardinality = 1	rel-man-single-select-multi-choice-radio
MultiChoiceMultiSelectRule	Child Products > 1 Maximum Cardinality > 1	rel-opt-multi-select-multi-choice-list
MultiChoiceOptionalSelectRule	Child Products > 1 Minimum Cardinality < 1 Maximum Cardinality = 1	rel-man-single-select-multi-choice-radio

To create a Relationship Block, the following method is provided:

```
public Block createRelationshipBlock(String pProductId, SiebelCatalogRelationship
pRelationship,
      ProductUIBuilder pProductUIBuilder)
```

The process for the method is:

1. Create a Relationship Block ID. This is a concatenation of the Product ID, the Relationship ID and the Relationship Block Suffix.
2. Query the StructuredProductUIRepository for a block with that ID.
3. If no block exists, create one and set the ID, the Relationship ID, the active flag to true and the type to RELATIONSHIP_BLOCK
4. Delegate to the ProductUIBuilder to create ProductUIs for each of the child products by iterating over the SiebelCatalogProducts in the relationship, passing each one to the buildProductUI method. This will have the effect of recursively generating the entire product tree structure. The resulting Product UIs will be set as the child product UIs on the Relationship Block. This step will also be performed on currently existing blocks, which will have the effect of adding or removing child Product UIs from the Relationship Block, thereby ensuring Relationships Blocks are kept up to date with the relationships they represent.
5. Set the display name on the Relationship Block, and the renderer using the rules described above.
6. Delegate to the SpuTools componenet to save the block to the StructuredProductUIRepository as new or update an existing one.

UITemplates

A UITemplate is a type of component which provides a pattern of block types from the Base UI Data to build a UI for the provided SiebelCatalogProduct.

The superclass has:

- a list of base blocks from which to construct a UI, and a CSS path property to associate with the Product UI.
- an abstract **build** method, which does the work of creating a Product UI and its blocks.
- a **buildId** method, which will construct an ID for the Product UI.
- an **isDeletable** method, which works out if the Product UI is deletable from its parent relationship.
- an **insertDeletableBlock** method, which determines whether or not to insert a delete button block.
- a **setExtraProperties** method, which allows subclasses to add extra properties to the ProductUI during the build method.

SimpleProductUITemplate

Example Configuration:

```
deleteButtonBlockId=rel-delete-button-block
CSSPath=../renderer/stylesheets/style.css
blocks=product-content-simple-product-block
```

This component is a template for Simple Product types. Its build method process is:

1. Build a product UI id using the id of the SiebelCatalogProduct
2. Query the SPU repository for a ProductUI with the same ID.
3. If a product UI exists, call insertDeletableBlock which will determine if the product's parent relationship has changed so that it requires a delete button block. Save the Product UI in the Structured Product UI Repository and return the Product UI object.
4. If no product UI exists, create a new one.
5. For each base block in the block list, call into the Block Manager and create a new Block. Add each new block to the list of blocks on the Product UI.
6. On the Product UI, set:

- The Id (constructed earlier)
- The CSS path
- The Product ID

- The Product Name

7. Call insertDeletableBlock which will add a delete button block if needed. . Save the Product UI in the Structured Product UI Repository and return the Product UI object.

SimpleProductWithAttributesUITemplate

Example Configuration

```
deleteButtonBlockId=rel-delete-button-block
CSSPath=../renderer/stylesheets/style.css
blocks=markup-div-start-block,product-content-heading-block,product-content-
description-block,\
        markup-table-start-block,ATTRIBUTES,markup-table-end-block,markup-div-end-
block
```

This component is the UITemplate for a Simple Product With Attributes type. Its build method is similar to that of the SimpleProductUITemplate, except includes the ability to create Attribute Blocks. This template must also check any existing Attribute Blocks are up to date with the Attributes found on the SiebelCatalogProduct, and vice versa. To allow this, the class has two methods:

```
protected List<Block> buildAttributeBlocks(SiebelCatalogProduct pProduct,
      BlockManager pBlockManager, ProductUIBuilder pProductUIBuilder)
```

This method iterates through the SiebelCatalogProduct's list of attributes, and delegates to the BlockManager to create an Attribute Block for each.

```
protected void replaceAttributeBlocks(SiebelCatalogProduct pProduct,
      BlockManager pBlockManager,
      ProductUIBuilder pProductUIBuilder,
      ProductUI pProductUI)
```

This method keeps existing Attribute Blocks up to date with the SiebelCatalogProduct definition. This involves creating a new Attribute Block if an attribute does not have an existing corresponding one. If this is the case, the new Attribute Block is inserted into the Product UI's block list after the last Attribute Block in the list. An Attribute Block will also be updated if the attribute's display name, attribute name, or data type has changed, and will also be removed if the attribute no longer exists in the SiebelCatalogProduct.

The build method's process is:

1. Build a product UI id using the id of the SiebelCatalogProduct
2. Query the SPU repository for a ProductUI with the same ID.
3. If a product UI exists,call into replaceAttributeBlocks, then call insertDeletableBlock which will determine if the product's parent relationship has changed so that it requires a delete button block. Save the Product UI in the Structured Product UI Repository and return the Product UI object.

4. If no product UI exists, create a new one.
5. For each base block in the block list, call into the Block Manager and create a new Block. Add each new block to the list of blocks on the Product UI.
6. When the Attribute Block marker (ATTRIBUTES) is encountered in the UITemplate's block list, call into buildAttributeBlocks to create the Product UI's attribute blocks.

7 On the Product UI, set:

- The Id (constructed earlier)
- The CSS path
- The Product ID
- The Product Name

8. Call insertDeletableBlock which will add a delete button block if needed. . Save the Product UI in the Structured Product UI Repository and return the Product UI object.

ConfigurableProductUITemplate

Example Configuration

```
deleteButtonBlockId=rel-delete-button-block
CSSPath=../renderer/stylesheets/style.css
blocks=markup-div-start-block,product-content-heading-block,product-content-
description-block,\
        markup-table-start-block,ATTRIBUTES,RELATIONSHIPS,markup-table-end-
block,markup-div-end-block
```

This is the UITemplate used to generate Product UIs for Configurable Products. The class extends SimpleProductWithAttributeUITemplate, but adds ability to deal with relationships. The component will also update existing ProductUIs of this type, and update any relationships which may have changed. To this end, it uses two methods:

```
protected List<Block> buildRelationshipBlocks(SiebelCatalogProduct pProduct,
      BlockManager pBlockManager, ProductUIBuilder pProductUIBuilder)
```

This method iterates through the relationships in the SiebelCatalogProduct, and delegates to the BlockManager to create a Relationship Block for the relationship.

```
protected void replaceRelationshipBlocks(SiebelCatalogProduct pProduct,
      BlockManager pBlockManager,
      ProductUIBuilder pProductUIBuilder,
      ProductUI pProductUI)
```

The purpose of this method is to determine if all the Relationship Blocks in an existing Product UI have corresponding relationships in the SiebelCatalogProduct. If a relationship no longer exists in the SiebelCatalogProduct, this method will remove the Relationship Block, and a new Relationship Block will be added to the Product UI if a new relationship has been added to the SiebelCatalogProduct and has been previously unaccounted for.

As Configurable Products can also have child products, this class can also have a configurable ProductUIBuilder. This is useful if you want to use a different ProductUIBuilder than the one supplied by the client class. The determineProductUI method will change the ProductUIBuilder to use the one configured as opposed to the one passed in as a parameter to the build process. We will see an example of how this can be useful when we configure different page styling for different level of products.

The build method's process is:

1. Build the Product UI's ID. Query the StructuredProductUIRepository to see if there are any Product UIs with that ID.
2. If a Product UI exists, call replaceAttributeBlocks and replaceRelationshipBlocks to make sure the Product UI is up to date with the SiebelCatalogProduct.
3. Call insertDeletableBlock in case the SiebelCatalogProduct's parent relationship has changed to allow the product to be deleted.
4. Update the Product UI by delegating to SPUTools to save to the repository.
5. If the Product UI does not already exist, create a new ProductUI object, and set the id, the product ID, the cssPath and the product's display name.
6. Similar to the SimpleProductWIthAttirbutesUITemplate, loop through the blocks, calling into the BlockManager to create new blocks as needed, and creating Attribute Blocks when the ATTRIBUTES marker in encountered.
7. When the RELATIONSHIPS marker is encountered, iterate through each child relationship on the SiebelCatalogProduct, calling into the BlockManager to create a relationship block for each. This will have the effect of recursively creating Product UIs for each child Product in the relationship.
8. Save the Product UI to the StructuredProductUIRepository.

PromotionUITemplate

Example Configuration:

```
controller=single-page-renderer
promotionProductUIBuilder=/atg/siebel/configurator/spu/generation/PromotionProductUIBuilder
CSSPath=../renderer/stylesheets/style.css
```

The PromotionUITemplate is slightly different in that there is no list of blocks needed to create a PromotionUI, but the Promotion's child products need to be created.

Promotions also have the added requirement of saving their promotion IDs to their child Product UIs. The Product UIs created must be distinct from the Product UIs created outside of a Promotion context, so that different styling may be applied to them.

To accomplish this, some of the classes already describe are sub-classed to allow the extra functionality needed to be added. The configured PromotionProductUIBuilder is a sub classed ProductUIBuilder which allows the Promotion ID to passed down to child products. These will be described in the next section.

The process for the build method of this UITemplate is:

1. Construct the Promotion UI ID, using the product Id and the Promotion UI suffix.
2. Query the StructuredProductUIRepository for a Promotion UI with that ID.
3. Iterate through the Child Relationships, and delegate to the PromotionProductUIBuilder to build each Child Product.
4. If the Promotion UI already exists, exit the method, otherwise create a new one, set the id, the controlling renderer, the promotion id and the CSS path on the item and save it to the repository, then iterate through the child products creating a Product UI for each.

Promotions

Product UIs in the context of a promotion must be distinct from those which exist outside of a promotion, even though they define the UI for the same product. As such, the Product UI repository item ID will include the Promotion ID.

As previously mentioned, the Promotions UI mechanism requires the Promotion ID to be added to any Product UIs associated with the promotion.

This is handled by sub-classing a lot of the previously described classes.

public class PromotionProductUIBuilder **extends** ProductUIBuilder

This sub class adds an ID property used to store the Promotion Id of the promotion in question. As the ProductUIBuilder is passed to the UITemplates when generating the ProductUIs, this allows the Promotion ID to be set on each associated Product UI.

public class PromotionBlockManager **extends** BlockManager

This sub class allows the BlockManager to construct unique IDs for each block in the Product UI associated with a Promotion. It does this by over-riding the createAttributeBlockID, createRelationshipBlockID and createContentBlockID methods, and having them prefix the ID constructed by the superclass with the Promotion ID.

public class PromotionSimpleProductWithAttributesUITemplate **extends** SimpleProductWithAttributesUITemplate

This sub class over-rides the buildID method to prefix the Product UI's ID with the promotion ID. It will also set the Promotion UI property on the Product UI by over-riding the setExtraProperties method.

public class PromotionSimpleProductUITemplate **extends** SimpleProductUITemplate

As above, except extends the SimpleProductUITemplate

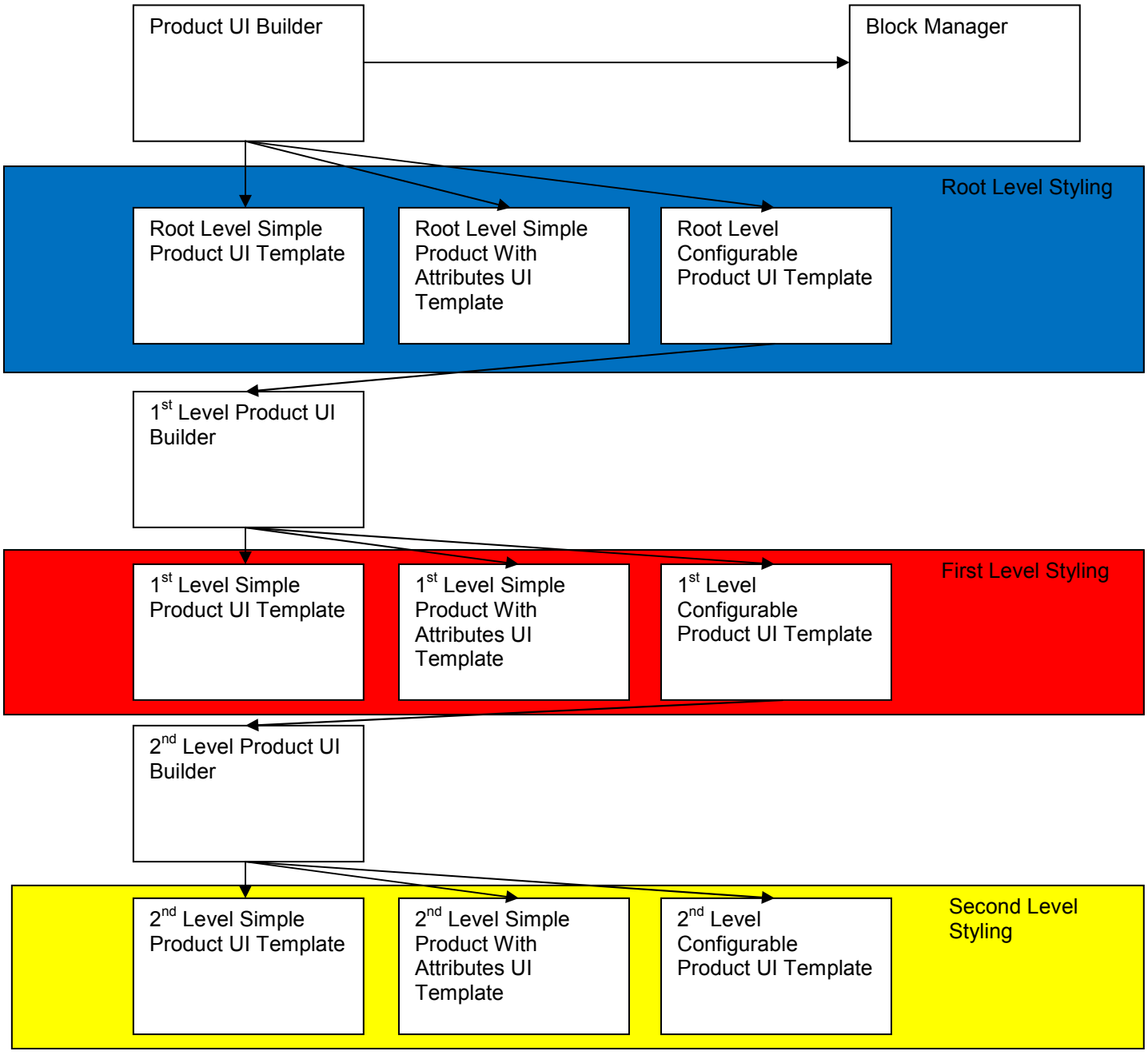
public class PromotionConfigurableProductUITemplate **extends** ConfigurableProductUITemplate

As above, but extending ConfigurableProductUITemplate. This class also over-rides the determineProductUIBuilder method to set the current Promotion ID to the configured ProductUIBuilder if needed.

Styling

For this reference implementation, we have styled the products based on their position in the configurable product tree. A root level product will have different styling to a 1st level product, which will be different than a 2nd level product and so on.

Below is a diagram showing the relationships between different components which can be used to achieve this.


Product UI Builder
Block Manager
Root Level Styling
Root Level Simple Product UI Template
Root Level Simple Product With Attributes UI Template
Root Level Configurable Product UI Template
1st Level Product UI Builder
First Level Styling
1st Level Simple Product UI Template
1st Level Simple Product With Attributes UI Template
1st Level Configurable Product UI Template
2nd Level Product UI Builder
Second Level Styling
2nd Level Simple Product UI Template
2nd Level Simple Product With Attributes UI Template
2nd Level Configurable Product UI Template

To configure styling for a root level product, we have a group of components which will be the UITemplates for root level products of a promotion. These templates will be based on the base data provided with this reference implementation. To take an example of the Root Level Simple Product With Attributes:

```
$class=atg.siebel.configurator.spu.generation.template.
SimpleProductWithAttributesUITemplate
$scope=global
deleteButtonBlockId=rel-delete-button-block
CSSPath=../renderer/stylesheets/atg.css
blocks=root-product-markup-div-start-block,product-content-heading-block,product-
content-description-block,\
        markup-table-start-block,ATTRIBUTES,markup-table-end-block,markup-div-end-
block
```

If we look at the blocks configured in this template, the root-product-markup-div-start-block gives the style to the UI when rendered:

```
<add-item item-descriptor="block" id="root-product-markup-div-start-block">
  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[root-product-markup-div-start-block]]></set-
property>
  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:75%;border:1px;border-color:#5C4A56;border-top-
style:solid;border-left-style:solid;border-right-style:solid;border-bottom-
style:solid;background-color:#F9F9F4"]]></set-property>
</add-item>
```

When the UI data is generated, and the UI rendered, each root product in a promotion will be displayed according to that styling:

As Configurable Products can have child products, they are configured with a link to a ProductUIBuilder component which is configured with UITemplates for products the next level down, for example:

```
$class=atg.siebel.configurator.spu.generation.ProductUIBuilder
$scope=global
spuTools=/atg/siebel/configurator/spu/StructuredProductUITools
blockManager=PromotionBlockManager
```

```
UITemplates=\DEFAULT=/atg/siebel/configurator/spu/generation/template/FirstLevelSimpl
eProductUITemplate,\
  Simple Product With
Attributes=/atg/siebel/configurator/spu/generation/template/FirstLevelSimpleProductWi
thAttributesUITemplate,\
  Configurable
Product=/atg/siebel/configurator/spu/generation/template/FirstLevelConfigurableProduc
tUITemplate
```

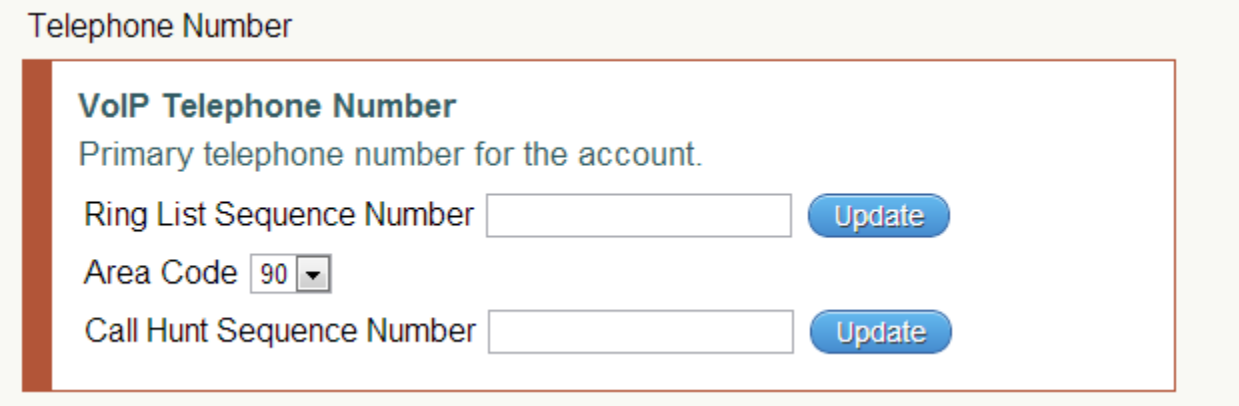
If we look at the Simple Product With Attributes UITemplate referenced here, we see it has a different configuration to that above:

```
$class=atg.siebel.configurator.spu.generation.template.
SimpleProductWithAttributesUITemplate
$scope=global
deleteButtonBlockId= rel-delete-button-block
CSSPath=../renderer/stylesheets/atg.css
blocks=first-level-product-markup-div-start-block,product-content-heading-
block,product-content-description-block,\
        markup-table-start-block,ATTRIBUTES,markup-table-end-block,markup-div-end-
block
```

The block referenced in this configuration has a different style to that above:

```
<add-item item-descriptor="block" id="first-level-product-markup-div-start-block">
  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[first-level-product-markup-div-start-
block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #B25538;border-left-
style:solid;border-left-width:15px;border-left-color:#B25538;background-
color:#FFFFFF"]]></set-property>
</add-item>
```

Which will render a product:



Product UI Templates can also be configured based on other properties on the product, as opposed to their productType. As an example, the VOIP Connection Charge product in a Promotion was configured with different styling.

The PromotionProductUIBuilder component at the level where this product appears was configured with a specialized template:

```
$class=atg.siebel.configurator.spu.generation.PromotionProductUIBuilder
$scope=global
spuTools=/atg/siebel/configurator/spu/StructuredProductUITools
isUsingDefaultTemplate=true
keyPropertyName=id
blockManager=PromotionBlockManager
UITemplates=\DEFAULT=/atg/siebel/configurator/spu/generation/template/PromotionFirstL
evelSimpleProductUITemplate,\
  Simple Product With
Attributes=/atg/siebel/configurator/spu/generation/template/PromotionFirstLevelSimple
ProductWithAttributesUITemplate,\
  Configurable
Product=/atg/siebel/configurator/spu/generation/template/PromotionFirstLevelConfigura
bleProductUITemplate,\
  3SIA-
2LAE5=/atg/siebel/configurator/spu/generation/template/PromotionConnectionChargeUITem
plate
```

So that we can decide which UITemplate to use, we need to let the ProductUIBuilder know which property to use a key. An example of how this is done is found in PromotionProductUIBuilder, in the getUITemplate method:

```
public UITemplate getUITemplate(SiebelCatalogProduct pProduct) throws
StructuredProductUIException
  {
    //set this initially to an empty string so don't get a null pointer from the Map
    String property = "";
```

```
    try
    {
       property = (String)new PropertyDescriptor(getKeyPropertyName(),
       SiebelCatalogProduct.class).getReadMethod().invoke(pProduct);
    }
    catch (IllegalAccessException e)
    {
      throw new StructuredProductUIException(e);
    }
    catch (InvocationTargetException e)
    {
      throw new StructuredProductUIException(e);
    }
    catch (IntrospectionException e)
    {
      //don't do anything, as we might not have a property name
    }

    UITemplate template = (UITemplate) mUITemplates.get(property);

    if(template==null)
    {
      template = super.getUITemplate(pProduct);
    }
    return template;
  }
```

In this example, we configure the ProductUIBuilder component with the name of the property to use as a key. The method will then use Java reflection introspection to find the property value from the SiebelCatalogProduct, in this case the Product's ID. If the product's ID returns no templates as a key, the over-rided method is called and the template used is based on the Product's productType.

Another example could be to use a Product's display name, and map that as a key to a template.

The specialized UITemplate example is:

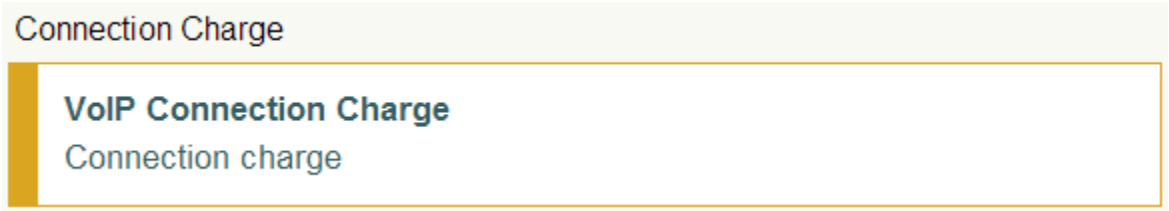
```
$class=atg.siebel.configurator.spu.generation.template.PromotionSimpleProductUITempla
te
$scope=global
deleteButtonBlockId=promo-rel-delete-button-block
CSSPath=../renderer/stylesheets/88-2EZSH.css
blocks=product-content-connection-charge-simple-product-block
```

This uses the specialized block product-content-connection-charge-simple-product-block:

```
<add-item item-descriptor="block" id="product-content-connection-charge-simple-
product-block">

  <set-property name="renderer"><![CDATA[product-content-simple-product]]></set-
property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>
```

```
  <set-property name="name"><![CDATA[product-content-connection-charge-simple-
product-block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #DAA520;border-left-
style:solid;border-left-width:15px;border-left-color:#DAA520;background-
color:#FFFFFF"]]></set-property>

</add-item>
```

This will result in a Product UI with different styling from others at the same level:



Convenience & Debugging tools

Building Product UI's without having to deploy a new project in Content Administration

The buildProductUI's method was left public so that it may be invoked from Dynamo admin. In this reference implementation, browse to:

http://<server-name>:<port>/dyn/admin/nucleus/atg/siebel/configurator/spu/generation/BlockGenerationService/

and invoke the buildProductUIs method.

Building the Product UI of a specific product.

The GenerateProductUIFormHandler is included, which accepts a Product ID as a parameter and calls into the ProductUIBuilder, passing in the target SiebelCatalogProduct.

Class Descriptions

atg.siebel.configurator.spu.generation.BlockGeneratorService

This is the entry point for the UI Generation process.

Class Declaration

public class BlockGenerationService extends GenericService

Class Data

The class maintains the following data

- catalogTools – a reference to the atg.siebel.catalog.SiebelCatalogToolsComponent.
- productUIBuilder – a reference to the atg.siebel.configurator.spu.generation.ProductUIBuilder component

Class Implementation

public void buildProductUIs()

This method calls into SiebelCatalogTools to get a list of SiebelCatalogProducts from the Product Catalog, which represent all the navigable products therein. It then iterates through this list and passes each SiebelCatalogProduct to the productUIBuilder's buildProductUI method.

atg.siebel.configurator.spu.generation.PublishingAgentBlockGenerationService

This class is located in the Siebel.PublishingAgent submodule. It extends the BlockGenerationService class to add the functionality of the buildProductUIs being called by a deployment complete event.

Class Declaration

public class PublishingAgentBlockGenerationService extends

BlockGenerationService implements DeploymentEventListener

Class Data

- deploymentAgent – a reference to the DepolymentAgent

Class Implementation

public void doStartService() throws ServiceException

This method is called when the component starts, and adds the current instance as a deployment listener on the DeploymentAgent.

public void doStopService() throws ServiceException

This method is called when the component stops, and removes the current instance as a deployment listener on the DeploymentAgent.

public void deploymentEvent(DeploymentEvent pEvent)

This method is called by the DeploymentAgent with which it has registered itself as a listener. The method will check the DeploymentEvent and if it is in a DEPLOYMENT_COMPLETE state, calls the buildProductUIs on the super class.

atg.siebel.configurator.spu.generation.ProductUIBuilder

This class holds a map of UITemplates keyed to ProductTypes of SiebelCatalogProducts, and is responsible for producing ProductUIs for each SiebelCatalogProduct it is given.

Class Declaration

public class ProductUIBuilder extends GenericService

Class Data

- blockManager – a reference to the atg.siebel.configurator.spu.generation.BlockManager component.
- spuTools – a reference to the atg.siebel.configurator.spu.StructuredProductUITools component.
- templates – a ServiceMap of UITemplates
- isUsingDefaultTemplate – a Boolean which switches if the ProductUIBuilder will use a default UITemplate (a UITemplate with the key DEFAULT) if no UITemplates are found in the templates map for a particular productType.

Class Implementation

public UITemplate getUITemplate(SiebelCatalogProduct pProduct) throws StructuredProductUIException

This method returns a UITemplate from the templates map for the SiebelCatalogProduct supplied. If no UITemplates are found for the productType, and isUsingDefaultTemplate is set to true, the UITemplate keyed to DEFAULT will be returned. Otherwise, null will be returned.

public String getTemplateKey(SiebelCatalogProduct pProduct)

This method returns the value of the property used as a key to the template map. This method returns the productType by default, but sub-classes should override this method if they wish to key the template map on a different property.

public ProductUI buildProductUI(SiebelCatalogProduct pProduct)

throws StructuredProductUIException

This method returns a ProductUI for the SiebelCatalogProduct supplied. This retrieves the UITemplate for the given product, and calls the build method on the UITemplate.

atg.siebel.configurator.spu.generation.PromotionProductUIBuilder

This is a specialized ProductUIBuilder, which allows the production of ProductUIs specifically for Promotions.

Class Declaration

public class PromotionProductUIBuilder extends ProductUIBuilder

Class Data

- id – A String representing the promotion ID, used to set the promotion id on child products. This is set by the client PromotionUITemplate class.
- keyPropertyName – An example of how to key the UITemplate map of different properties. See the Styling section.

Class Implementation

See the Styling section.

atg.siebel.configurator.spu.generation.BlockManager

This class deals with the interaction between the UITemplates and the StructuredProductUIRepository, via the SPU Tools component, creating Content, Attribute and Relationship blocks.

Class Declaration

public class BlockManager extends GenericService

Class Data

- attributeBlockSuffix – a String representing the suffix for Attribute Block Ids.
- relationshipBlockSuffix – a String representing the suffix for Relationship Block Ids.
- attributeDataTypeRenderers – a Map of Strings, which represent the attribute renderers for different data types. These are keyed by attribute domain types.
- spuTools – a reference to the StructuredProductUITools component.
- relationshipRendererRules – a List of RelationshipRendererRules components.

Class Implementation

public Block createAttributeBlock(String pProductId, SiebelCatalogAttribute pAttribute,

ProductUIBuilder pProductUIBuilder)

throws StructuredProductUIException

This method creates an attribute block, sets the properties on it, delegates to SPUTools to save it to the StructuredProductUIRepository, and returns it to the caller. If a block already exists in the repository with that ID, that block is updated with values from the SiebelCatalogAttribute and the repository is updated.

This method calls into getAttributeRendererForType to get the renderer for the SiebelCatalogAttributes domain type.

public String createAttributeBlockID(String pProductID, String pAttributeID, ProductUIBuilder pProductUIBuilder)

This method creates an ID for an attribute block by concatenating the Product Id, Attribute ID and the attribute block suffix.

public Block createContentBlock(String pProductId, String pBaseBlockId,

ProductUIBuilder pProductUIBuilder)

throws StructuredProductUIException

This method creates a content block, based on the block reference by pBaseBlockId, and saves it to the StructuredProductUIRepository.

public String createContentBlockId(String pProductId, String pBaseBlockId,

ProductUIBuilder pProductUIBuilder)

This method creates a content block id by concatenating the pProductId and the pBaseBlockId.

public Block createRelationshipBlock(String pProductId, SiebelCatalogRelationship pRelationship,

ProductUIBuilder pProductUIBuilder)

throws StructuredProductUIException

This method creates a Relationship Block, sets the properties on it, delegates to SPUTools to save it to the StructuredProductUIRepository, and returns it to the caller. If the block already exists in the repository with that ID, that block will be updated with values from the SiebelCatalogRelationship and the repository is updated.

This method calls into the getRelationshipRenderer method get the renderer to set on the block for this type of relationship.

public String createRelationshipBlockID(String pProductID, String pRelationshipID, ProductUIBuilder pProductUIBuilder)

This method creates an ID for an attribute block by concatenating the Product Id, Relationship ID and the relationship block suffix.

private String getAttributeRendererForType(String pType)

This method gets the String from the attributeRenderer Map and returns it to the caller.

public Block getBlock(String pBlockId) throws StructuredProductUIException

This method delegates to the SPUTools component to get a block from the repository with the supplied ID.

private List<ProductUI> getProductUIs(

SiebelCatalogRelationship pRelationship, ProductUIBuilder pProductUIBuilder)

throws StructuredProductUIException

This is called from createRelationshipBlock, and creates child ProductUIs for the SiebelCatalogRelationship, reference to which are set on the RelationshipBlock before they are saved. This method calls into the ProductUIBuilder with the SiebelCatalogProduct which is the child of the relationship, and returns a list of them to the caller.

public String getRendererForRelationship(SiebelCatalogRelationship pRelationship)

throws StructuredProductUIException

This method returns the ID of the renderer for the SiebelCatalogRelationship. It's process is described in the Relationship Block section of this Integration Guide.

private Renderer getRelationshipRenderer(

SiebelCatalogRelationship pRelationship) throws StructuredProductUIException

This method calls into getRendererForRelationship to get the renderer ID, and using this into SPUTools get the Renderer for the SiebelCatalogRelationship.

atg.siebel.configurator.spu.generation.PromotionBlockManager

This class specializes the BlockManager to add extra functionality for Promotions.

Class Declaration

public class PromotionBlockManager extends BlockManager

Class Implementation

public String createAttributeBlockID(String pProductID, String pAttributeID, ProductUIBuilder pProductUIBuilder)

This method prefixes the ID created by the super class with the ID of the Promoiton from the ProductUIBuilder.

public String createRelationshipBlockID(String pProductID, String pAttributeID, ProductUIBuilder pProductUIBuilder)

As above.

public String createContentBlockID(String pProductID, String pAttributeID, ProductUIBuilder pProductUIBuilder)

As above.

atg.siebel.configurator.spu.generation.rules.RelationsipRendererRule

This is an abstract class, which forms the superclass for any RelationshipRendererRules.

Class Declaration

public abstract class RelationshipRendererRule extends GenericService

Class Data

- renderer – a String representing the ID of the renderer configured for this rule.

Class Implementation

public abstract String determineRenderer(SiebelCatalogRelationship pRelationship)

throws StructuredProductUIException;

This method must be implemented by every RelationshipRuleType. If the SiebelCatalogRelationship meets the criteria to be rendered by the configured renderer, return the renderer ID.

For a detailed explanation of each RelationshipRuleRenderer, see the Relationship Blocks section.

atg.siebel.configurator.spu.generation.template.UITemplate

This is an abstract class, which forms the superclass for any UITemplates.

Class Declaration

public abstract class UITemplate extends GenericService

Class Data

- blocks – a list of Strings which are the repository IDs of Blocks found in base data.
- cssPath – the cssPath which will be set on the ProductUI which is created.
- deleteButtonBlockID – the ID of the Block in the base SPU data which will be saved as part of the ProductUI's block list if needed.

Class Implementation

public abstract ProductUI build(SiebelCatalogProduct pProduct,

BlockManager pBlockManager,

ProductUIBuilder pProductUIBuilder)

throws StructuredProductUIException;

This method is called by the ProductUIBuilder and must be implemented by each UITemplate type. It must construct a ProductUI, save it to the StructuredProductUIRepository and return it to the caller.

protected String buildID(SiebelCatalogProduct pProduct, ProductUIBuilder pProductUIBuilder)

This method builds the repository ID for the ProductUI. It constructs the ID by concatenation the parent product's ID, if it has one (which it gets from the SiebelCatalogProduct's parentRealtionship), and the product ID of the SiebelCatalogProduct and "prod-ui".

protected void setExtraProperties(ProductUI productUI, ProductUIBuilder pProductUIBuilder)

throws StructuredProductUIException

This is a no-op method, and is used by superclasses to set any extra data on the ProductUI specific to them.

protected boolean isDeletable(SiebelCatalogProduct pProduct)

This method returns true or false depending on whether the SiebelCatalogProduct is deletable from it's current parent relationship. The product is deletable from the parent relationship if the parent's relationship's maximum cardinality is greater than 1.

protected void insertDeletableBlock(SiebelCatalogProduct pProduct,

BlockManager pBlockManager, ProductUIBuilder pProductUIBuilder, ProductUI pProductUI) throws StructuredProductUIException

This method checks the product's deleteable status by calling isDeletable. If it is deletable, a delete button block is created by calling into the Block Manager to create a content Block based on the ID configured by deleteButtonBlockID and is set in the Block list of the ProductUI, and if not any delete button blocks are removed from the ProductUIs block list.

atg.siebel.configurator.spu.generation.template.SimpleProductUITemplate

This is the UITemplate for a SimpleProduct.

Class Declaration

public class SimpleProductUITemplate extends UITemplate

Class Data

Class Implementation

See description above in UITemplates.

atg.siebel.configurator.spu.generation.template.SimpleProductWithAttributesUITemplate

This is the UITemplate for a SimpleProductWithAttributes.

Class Declaration

public class SimpleProductWithAttributesUITemplate extends UITemplate

Class Data

Class Implementation

See description above in UITemplates.

atg.siebel.configurator.spu.generation.template.ConfigurableProductUITemplate

This is the UITemplate for a ConfigurableProduct.

public class ConfigurableProductUITemplate extends UITemplate

Class Data

Class Implementation

See description above in UITemplates.

atg.siebel.configurator.spu.generation.PromotionUITemplate

This is the UITemplate for a Promotion.

public class PromotionUITemplate extends UITemplate

Class Data

- controller – a String representing the default controller for PromotionUIs.
- promotionProductUIBuilder – a reference to a atg.siebel.configurator.spu.generation.PromotionProductUIBuilder

Class Implementation

See description above in UITemplates.

atg.siebel.configurator.spu.generation.template.PromotionSimpleProductUITemplate

This class specializes the SimpleProductUITemplate to add functionality for Promotions.

Class Declaration

public class PromotionSimpleProductUITemplate extends SimpleProductUITemplate

Class Data

Class Implementation

protected String buildID(SiebelCatalogProduct pProduct, ProductUIBuilder pProductUIBuilder)

This method builds the ID of ProductUI by prefixing the ID produced by the superclass with the ID of the Promotion from the ProductUIBuilder.

protected void setExtraProperties(ProductUI productUI, ProductUIBuilder pProductUIBuilder) throws StructuredProductUIException

This method adds the ID of the promotion which it gets from the ProductUIBuilder to the ProductUI.

atg.siebel.configurator.spu.generation.template. PromotionSimpleProductWithAttributesUITemplate

This class specializes the SimpleProductWithAttributesUITemplate to add functionality for Promotions.

Class Declaration

public class PromotionSimpleProductWithAttributesUITemplate extends SimpleProductWithAttributesUITemplate

Class Data

Class Implementation

protected String buildID(SiebelCatalogProduct pProduct, ProductUIBuilder pProductUIBuilder)

This method builds the ID of ProductUI by prefixing the ID produced by the superclass with the ID of the Promotion from the ProductUIBuilder.

protected void setExtraProperties(ProductUI productUI, ProductUIBuilder pProductUIBuilder) throws StructuredProductUIException

This method adds the ID of the promotion which it gets from the ProductUIBuilder to the ProductUI.

atg.siebel.configurator.spu.generation.template. PromotionConfigurableProductUITemplate

This class specializes the ConfigurableProductUITemplate to add functionality for Promotions.

Class Declaration

public class PromotionConfigurableProductUITemplate extends ConfigurableProductUITemplate

Class Data

Class Implementation

protected String buildID(SiebelCatalogProduct pProduct, ProductUIBuilder pProductUIBuilder)

This method builds the ID of ProductUI by prefixing the ID produced by the superclass with the ID of the Promotion from the ProductUIBuilder.

protected void setExtraProperties(ProductUI productUI, ProductUIBuilder pProductUIBuilder) throws StructuredProductUIException

This method adds the ID of the promotion which it gets from the ProductUIBuilder to the ProductUI.

protected ProductUIBuilder determineProductUIBuilder(

ProductUIBuilder pProductUIBuilder)

This method determines which ProductUIBuilder object to use, and returns it. The configured ProductUIBuilder will take precedence over the one supplied as a parameter, and the PromotionID will be copied to the one returned to the caller.

Appendix

Base Data

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!DOCTYPE gsa-template SYSTEM "dynamosystemresource:/atg/dtds/gsa/gsa_1.0.dtd">

<gsa-template>


<import-items>


<add-item item-descriptor="block" id="product-content-connection-charge-simple-
product-block">

  <set-property name="renderer"><![CDATA[product-content-simple-product]]></set-
property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[product-content-connection-charge-simple-
product-block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #DAA520;border-left-
style:solid;border-left-width:15px;border-left-color:#DAA520;background-
color:#FFFFFF"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="product-content-second-level-simple-product-
block">
```

```
  <set-property name="renderer"><![CDATA[product-content-simple-product]]></set-
property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[product-content-second-level-simple-product-
block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #507282;border-left-
style:solid;border-left-width:15px;border-left-color:#592B1E;background-
color:#FFFFFF"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="second-level-product-markup-div-start-block">

  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[second-level-product-markup-div-start-
block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #507282;border-left-
style:solid;border-left-width:15px;border-left-color:#592B1E;background-
color:#FFFFFF"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="product-content-first-level-simple-product-
block">

  <set-property name="renderer"><![CDATA[product-content-simple-product]]></set-
property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[product-content-first-level-simple-product-
block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #B25538;border-left-
style:solid;border-left-width:15px;border-left-color:#B25538;background-
color:#FFFFFF"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="first-level-product-markup-div-start-block">

  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>
```

```
  <set-property name="name"><![CDATA[first-level-product-markup-div-start-
block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:100%;border:1px solid #B25538;border-left-
style:solid;border-left-width:15px;border-left-color:#B25538;background-
color:#FFFFFF"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="product-content-root-simple-product-block">

  <set-property name="renderer"><![CDATA[product-content-simple-product]]></set-
property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[product-content-root-simple-product-
block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:75%;border:1px;border-color:#5C4A56;border-top-
style:solid;border-left-style:solid;border-right-style:solid;border-bottom-
style:solid;background-color:#F9F9F4"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="root-product-markup-div-start-block">

  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[root-product-markup-div-start-block]]></set-
property>

  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:75%;border:1px;border-color:#5C4A56;border-top-
style:solid;border-left-style:solid;border-right-style:solid;border-bottom-
style:solid;background-color:#F9F9F4"]]></set-property>

</add-item>


<add-item item-descriptor="block" id="promo-rel-delete-button-block">

  <set-property name="renderer"><![CDATA[rel-delete-button]]></set-property>

  <set-property name="activeFlag"><![CDATA[true]]></set-property>

  <set-property name="name"><![CDATA[promo-rel-delete-button-block]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[class="deletebut"]]></set-property>

</add-item>
```

```
<add-item item-descriptor="block" id="rel-delete-button-block">
  <set-property name="renderer"><![CDATA[rel-delete-button]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[rel-delete-button-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[class="deletebut"]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-div-end-block">
  <set-property name="renderer"><![CDATA[markup-div-end]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-div-end-block]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-div-start-block">
  <set-property name="renderer"><![CDATA[markup-div-start]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-div-start-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:75%;border:1px;border-color:#5C4A56;border-top-
style:solid;border-left-style:solid;border-right-style:solid;border-bottom-
style:solid;background-color:#F9F9F4"]]></set-property>
</add-item>

<add-item item-descriptor="block" id="product-content-description-block">
  <set-property name="displayName"><![CDATA[]]></set-property>
  <set-property name="renderer"><![CDATA[product-content-description]]></set-
property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[product-content-description-block]]></set-
property>
</add-item>

<add-item item-descriptor="block" id="product-content-heading-block">
```

```
  <set-property name="renderer"><![CDATA[product-content-heading]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[product-content-heading-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="block" id="product-content-simple-product-block">
  <set-property name="renderer"><![CDATA[product-content-simple-product]]></set-
property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[product-content-simple-product-block]]></set-
property>
  <set-property name="htmlAttributes"><![CDATA[style="padding:10px;font:17px
arial,sans-serif;color:#335C64;width:75%;border:1px;border-color:#5C4A56;border-top-
style:solid;border-left-style:solid;border-right-style:solid;border-bottom-
style:solid;background-color:#F9F9F4"]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-table-end-block">
  <set-property name="renderer"><![CDATA[markup-table-end]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-table-end-block]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-table-start-block">
  <set-property name="renderer"><![CDATA[markup-table-start]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[Configuration Options]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="block" id="product-quantity-block">
  <set-property name="displayName"><![CDATA[]]></set-property>
  <set-property name="renderer"><![CDATA[product-quantity-renderer-1]]></set-
property>
```

```
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[product-quantity-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-tr-start-block">
  <set-property name="renderer"><![CDATA[markup-tr-start]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-tr-start-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-tr-end-block">
  <set-property name="renderer"><![CDATA[markup-tr-end]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-tr-end-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-td-start-block">
  <set-property name="renderer"><![CDATA[markup-td-start]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-td-start-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="block" id="markup-td-end-block">
  <set-property name="renderer"><![CDATA[markup-td-end]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-td-end-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>
```

```
<add-item item-descriptor="block" id="markup-nbsp-block">
  <set-property name="renderer"><![CDATA[markup-nbsp]]></set-property>
  <set-property name="activeFlag"><![CDATA[true]]></set-property>
  <set-property name="name"><![CDATA[markup-nbsp-block]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="rel-opt-single-select-single-choice-check">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-opt-single-select-
single-choice-check.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="markup-table-end">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-table-
end.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="rel-man-single-select-single-choice-text">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-man-single-select-
single-choice-text.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="markup-table-start">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-table-
start.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="rel-delete-button">
```

```
  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-delete-
button.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="single-page-renderer">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/single-page-
renderer.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="product-attribute-select">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/product-attribute-
select.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="product-content-description">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/product-content-
description.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="rel-opt-multi-select-multi-choice-list">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-opt-multi-select-
multi-choice-list.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

  <set-property name="htmlAttributes"><![CDATA[class="addvtn"]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="markup-div-end">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-div-
end.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>
```

```
<add-item item-descriptor="renderer" id="rel-man-single-select-multi-choice-radio">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-man-single-select-multi-choice-radio.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="multi-page-renderer">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/multi-page-renderer.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="product-attribute-text">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/product-attribute-text.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[class="updatebut"]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="rel-opt-multi-select-single-choice-button">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-opt-multi-select-single-choice-button.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
  <set-property name="htmlAttributes"><![CDATA[class="addvtn"]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="markup-div-start">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-div-start.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="product-content-heading">
```

```
  <set-property name="jspPath"><![CDATA[/configurator/renderer/product-content-
heading.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="product-content-simple-product">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/product-content-
simple-product.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="rel-opt-multi-select-multi-choice-select">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/rel-opt-multi-select-
multi-choice-select.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="product-quantity-renderer-1">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/product-quantity-
renderer.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="markup-tr-start">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-tr-
start.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>


<add-item item-descriptor="renderer" id="markup-tr-end">

  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-tr-
end.jsp]]></set-property>

  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>

</add-item>
```

```
<add-item item-descriptor="renderer" id="markup-td-start">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-td-
start.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="markup-td-end">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-td-
end.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="markup-br-start">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-br-
start.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

<add-item item-descriptor="renderer" id="markup-nbsp">
  <set-property name="jspPath"><![CDATA[/configurator/renderer/markup-
nbsp.jsp]]></set-property>
  <set-property name="applicationContext"><![CDATA[/siebel]]></set-property>
</add-item>

</import-items>

</gsa-template>
```