Creating ePages Portal Cartridges

e.pages

Version:

September 9, 2010

Table of Contents

1 Creating ePages Product-Portal Cartridges

Beginning with version 6.10, ePages provides developers with a streamlined development procedure for product portal cartridges. The *createCartridge.pl* script can be used to generate an almost-complete cartridge, depending on how much detail you add to an optional configuration file.

✓ **NOTE: This routine only produces usable .csv export files, with an export-driver in the API/CSV/ folder. If your portal needs XML-formatted exports, you will need to add your own XML export-driver in a new API/XML/ folder (see final chapter for details).**

Using the createCartridge Templates

You can generate an almost-finished cartridge just by running the normal *createCartridge.pl* script found in *%EPAGES_CARTRIDGES%\DE_EPAGES\Cartridge\Scripts*.

It needs two extra parameters in order to generate the extra code:

- -**configfile** path/filename to the .ini file to be used for guiding the code generation
- -**basepackage** always "DE_EPAGES::ProductPortal"

Example:

```
perl createCartridge.pl –configfile
..\..\ProductPortal\CartridgeTemplates\myconfig.ini
–basepackage DE_EPAGES::ProductPortal MyCompany::MyPortal
```

Config File

The generation process is guided by an .ini configuration file. You should make a copy of the existing example file, *config.ini*, included in *%EPAGES_CARTRIDGES%\DE_EPAGES\ProductPortal\CartridgeTemplates*, and then modify that copy.

There are several parameters which should be modified for your own new cartridge:

- **PortalName** enter your desired name (to be displayed in MBO)
- **PortalSites** remove/modify the existing German site (country where portal is active), add any extra sites needed. You must enter each site on a separate line between the two "EOL" markers.
- **MarketingBannerPosition** where should your logo be placed in the rotating logo slide-show in the main MBO marketing page? The existing logos have positions of 10, 20, 30, etc. Enter a number between these to put your logo between existing logos.
- **NavElementPosition** where should the navbar-element for your portal logo be located in the list of portal logos in the MBO ›› Design ›› add navbar element list.
- **CSVSepChar** desired .csv separator character(s), usually comma (","), semi-colon (";") or tabulator ("\t").
- **CSVAlwaysQuote** should strings always be enclosed in quotes?
- **CSVEncoding** which character set should be used for the resulting .csv file?
- **CSVExportAttributes** this important parameter defines the names of the export file columns, and the Perl code needed to fill them. For each needed export-file field name, you must enter a line with:
  `Entry=field_name, GetWithParent=product_attribute_name`
  or

```
Entry=field_name,
Special=name_for_special_perl_code_to_retrieve_its_data
```

- The field-name is as you wish it to appear in your export file. The part following the comma is the method for getting the ePages values (GetWithParent or Special), and the ePages attribute name.
- The GetWithParent=product_attribute_name syntax can be used for retrieving any product data which is stored directly as an attribute value for ePages product objects.

  NOTE: for a list of the available Product class attributes, see the list at the end of this chapter.
- The *Special=name_for_special_data* which is not directly stored in a product attribute, such as calculated value, like the price (depends on site's currency), shipping-price (could depend on weight), etc. There arer several pre-defined names:
  Alias = Product alias (product number)
  Description = Product description (MBO-configurable)
  Price = Product price (uses currency from PortalSite LocaleID)
  Category = Product category
  Image = Product image (MBO-configurable)
  ShippingPrice (MBO-configurable)

  NOTE: If you need any further special fields, you must implement them in the generated CSV-export-driver Perl module in your cartridge's API/CSV/ folder. The "MBO-configurable" fields mean that the merchant can choose the desired attribute used to fill this field using the back-office settings page for the portal

Example config file code:

```
; General Settings

[MAIN]

; Feature position

FeaturePosition=311

; List of Package dependencies. (multi-line) ; Package = ePages
Package ; Comment = Text Comment

AllDependencies=<<EOL

Package=DE_EPAGES::ProductPortal, Comment=Product Portal framework
Package=DE_EPAGES::ThirdPartyConfig, Comment=ThirdParty config
framework EOL


; Product Portal settings

[Portal]

; Portal name

PortalName=My Fancy Portal

; Marketing banner position

MarketingBannerPosition=21

; Navigation Element position

NavElementPosition=21

; List of portal sites (multi-line, 1 site per line)

PortalSites=<<EOL

Alias=MyPortal_DE, LocaleID=de_DE, PortalURL=http://www.myportal.de,
RegisterURL=http://www.myportal.de/register.html,
AdminURL=https://www.myportal.de/login/, EMail=shops@myportal.de,
Phone=+49 (0)1805-123456-0 EOL
```

```
; CSV: seperator
CSVSepChar=;
; CSV: always quote?
CSVAlwaysQuote=1
; CSV: file encoding
CSVEncoding=utf8
; CSV: export columns (multi-line)
; Entry         = Column name
; GetWithParent = Use simple Product->getWithParent
; Special       = Use special template code. Possible vals:
;                 - Alias  = Product alias
;                 - Description = Product description (MBO-
configurable)
;                 - Price = Product price
;                 - Category = Product category
;                 - Image = Product image (MBO-configurable)
;                 - ShippingPrice (MBO-configurable)
CSVExportAttributes=<<EOL
Entry=aid, Special=Alias
Entry=brand, GetWithParent=Manufacturer
Entry=mpnr, GetWithParent=ManufacturerSKU Entry=ean,
GetWithParent=UPCEAN Entry=name, GetWithParent=NameOrAlias
Entry=desc, Special=Description Entry=shop_cat, Special=Category
Entry=price, Special=Price Entry=link, Special=ProductURL
Entry=image, Special=Image Entry=dlv_time,
GetWithParent=DeliveryPeriod Entry=dlv_cost, Special=ShippingPrice
EOL
```

Post-Generation Modifications

After the cartridge code is created, you must modify several elements:

- All images found in the Data/Public/ structure must be replaced with real logos for your portal
- The MBO templates and their language files need to be modified for your desired texts (RelatedTopic, TabPage)
- If your portal uses languages other than English and German, you need to add the desired language files in the Templates/ folders.
- You should check the generated export-driver in API/CSV/ to make sure all desired fields are included, and using the correct code.

Product Class Simple Attributes

(As of version 6.10)

Attribute Name	Data Type	Description
Alias	String	Public object identifier. Alias is unique for object with same parent object (see Parent).
AvailabilityDate	Date	Date when the product is going to be available/is for sale.

CanAddToBasket	Boolean	True if product can be added to basket (MinOrder is used for stock level test).
CreationDate	DateTime	creation date
DeliveryPeriod	String	Delivery period.
DisplayDeliveryWeight	Boolean	True if delivery weight should be displayed.
FileAlias	String	Object string without special characters, suitable as file or directory name.
FilePath	String	Directory name of the object relative to the root object.
FilePathPart	String	file path part from site to object
GUID	String	Globally unique id.
HasSubOwnPrices	Boolean	True if sub product has own prices, used by pricelist cartridge.
Height	Float	Height in millimeters.
ImageHotDeal	File	Hot deal image.
ImageLarge	File	Large image, click on medium image on detail page.
ImageMedium	File	medium image, used detail page
ImageSmall	File	small image, used for lists
ImageThumbnail	File	Thumbnail image.
IntervalOrder	Float	Increment steps for order quantity.
IsAvailable	Boolean	Indicates that the product is for sale.
IsDailyPrice	Boolean	Hint for customer, price can be changed daily (e.g. gas/petrol).
IsDeliveryTimeLonger	Boolean	Tests if a product has a longer delivery time (not on stock and shop has selected 'possible delivery delay'), the current basket is not involved.
IsNew	Boolean	Marked as New
IsSpecialOffer	Boolean	True if product is assiged to the special offer category.
IsUPCEANValid	Boolean	Is the UPC or EAN No. valid
IsVisible	Boolean	True if the object is visible in the shop.
Length	Float	Length in millimeters.
Manufacturer	String	manufactured by
ManufacturerSKU	String	Manufacturer Product No.
MinOrder	Float	Minimum amount of the product that can be ordered.
NavigationString	String	NavigationCaption or Name or Alias
PrepaymentType	String	Type of prepayment (percent, fix
PrepaymentValue	Float	Percental value of prepayment
PriceQuantity	Float	All prices are based on this amount of order units.
PublicPath	String	Public path (file system directory) of this object. This directory is accessable via web server (to save e.g. images, pdf ...).
RatingAverage	Float	product rating averaging
RefAmount	Integer	Amount (in reference unit) for which the reference price is calculated, for example: 100 (gram).

RefContentAmount	Float	Amount (in reference unit) that is included in one sales unit of the product, for example: 150 (gram).
StockAlert	String	Current stock status: 'NO_STOCK','ON_STOCK','WARN_STOCK','OUT_STOCK'.
StockLevel	Float	Number of products on stock.
StockLevelAlert	Float	Yellow light on stock level below this alert level.
UPCEAN	String	UPC or EAN No.
WebUrl	String	HTTP URL to this object.
WebUrlSSL	String	HTTP/SSL URL to this object.
Weight	Float	Weight Amount in Specified Units
Width	Float	Width in millimeters.
_StockLevel	Float	Quantity of products on stock from database.

Summary

- ✓ **Using the createCartridge templates for portal cartridges, you can generate an almost-complete cartridge is very little time.**

2 XML Export Drivers

At the end of this lesson you will know:

- **How an ePages portal cartridge can handle XML-formatted exports**

Overview: Available XML Drivers

If your portal server needs XML-formatted import files, the ProductPortal cartridge provides 2 different XML drivers to handle it. One of these should be used as the base class for your own export driver instead of DE_EPAGES::ProductPortal::API::CSV::PortalExportDriver.pm.

The 2 XML drivers are:

- **DE_EPAGES::ProductPortal::API::XML::PortalExportDriver.pm** – the standard XML driver
- **DE_EPAGES::ProductPortal::API::XML::ProductPortalExportDriver.pm** – the SAX2 XML driver

✓ **NOTE: See the enclosed example MyPortalProductsExportDriver.pm code in the PortalCartridgeDev_coursefiles\ExampleXMLDriver folder for an example an XML-formatted SAX2 export driver!**

Main Methods

Methods inherited from DE_EPAGES::XML::API::BaseExportDriver:

- *exportXML()* writes XML file with given encoding
  - Example: $Driver->exportXML('data.xml', {encoding =>'utf8'});
- *drive()* starts export of items
  - Syntax: $self->drive();
- *items()* returns ref. array of items to export
- *addElement()* adds XML element, executes hooks for subelements
  - Example: $self->addElement('Company', { CompanyID => $CompanyID });
  - Can pass parameters as argument to the addHook-method in the hash
  - You can enter a value for the element directly as a third argument in the method Example: $self->addElement('Company', { CompanyID => $CompanyID }, $Shop->get('Slogan',$LanguageID));
- *addHook()* creates sub-elements for given parent element

  The "addHook()" method of the DE_EPAGES::XML::API::BaseExportDriver is used to create sub-elements within a given parent element of an XML structure. It's the standard method in ePages to build up such structures. It needs two parameters:

  - The TagName of the XML tag which has to be created
  - The MethodName of the function (declared in the same file) which is used to implement the functionality (and fill the value for the given tag). This normally includes something like an "*addElement*()" method and the handling of the values stored there.
  - Hooks are tags holding values

✓ **IMPORTANT: The *addHook*-Method must come BEFORE the addElement-Method creates the element itself which triggers the hook, or *addHook*() won't run.**

- *topID()* returns current value of a parameter passed by the *addElement*-Method as an argument
  - Example: `$CompanyID = $self->topID( 'CompanyID' );`
- NOTE: To create and set attributes within the elements, you need to create a method with the same name as the element itself.
  - You must return a hash of attribute names and their values
  - Example: `return { sale => 'no' };`

How to Structure your XML Driver

Here are a few tips on creating the structure for your export driver:

1. In the exportXML()-method, add needed global arguments to $self hash:

```
sub exportXML {
    my $self = shift;
    my ($PortalConfig, $Store, $Shop) = @_;
    # save arg in $self hash for use in all subs
    $self->{'Shop'} = $Shop;
```

2. In the drive()-method, add an addHook()-method for an AllSubTags subroutine from the root element. Then add an addElement()-method for the root element. Read arguments from

```
sub drive {
    my $self = shift;
    # call 'AllSubtags' for each 'Shop' element
    $self->addHook( 'Shop' => 'AllSubtags' );
    # start the export by adding the root element
    $self->addElement( 'Shop' );
```

3. In your AllSubTags subroutine, use an addElement()-method for each needed second-level tag. Insert values, if needed, and pass parameters in the hash as arguments for the addHook()-methods.

```
sub AllSubtags {
    my $self = shift;
    # get saved Shop argument from $self hash
    my $Shop = $self->{'Shop'};
   # addHook must be run before addElement for "Common" tag
    $self->addHook( 'Common' => 'CommonSubtags' );
    $self->addElement( 'Common', {'PortalConfig'=> $PortalConfig }
   ...
```

4. NOTE: you *ONLY* need a subroutine with the same name as the tags if you want to work on its attributes, such as for "length and "type" in the following example:

```
<item length="24" type="category">
```

Example XML Export

For example, a portal system might demand import via XML and SAX2.

A typical SAX2 XML export driver does the following:

- Implements exportXML{} with $self->addMyPortalHooks;
- Implements drive{} with $self->addElement(<root_element>', {});
  - NOTE: To add attributes to the tag, place them within {}
  - Example: $self->addElement('Company', {ID => $CompanyID });
- Implements addMyPortalHooks{} with addHook() for each sub-element
  - addHook() adds element name and calls subroutine which implements it
- Implements isProductValidPortal{} for required fields at portal
- Implements the various subroutines for setting values of hooks
  - These are called by addHook() routines within addMyPortalHooks{}

```
#===============================================================================
# $package
DE_EPAGES::MyPortal::API::XML::MyPortalProductsExportDriver
# $state        private
#-------------------------------------------------------------------------------
# $description  This module implements a SAX2 driver to export object
MyPortal products
#
# $example      use
DE_EPAGES::MyPortal::API::XML::MyPortalProductsExportDriver;
#
#               my $Driver =
DE_EPAGES::MyPortal::API::XML::MyPortalProductsExportDriver->new(
#                   Handler => $Handler
#               );
#               $Driver->addMyPortalHooks;
#               $Driver->exportXML;
#===============================================================================
package DE_EPAGES::MyPortal::API::XML::MyPortalProductsExportDriver;
use base DE_EPAGES::ProductPortal::API::XML::PortalExportDriver;


use strict;
use DE_EPAGES::Core::API::Error             qw ( GetError ExistsError
);
use DE_EPAGES::Core::API::Warning           qw ( Warning );
use DE_EPAGES::Object::API::Factory         qw ( LoadObject );
use DE_EPAGES::Object::API::Table::Currency qw ( GetInfoCurrency );
use DE_EPAGES::Core::API::Html              qw ( RemoveHtmlTags );


my @ADD_MyPortal_ATTRIBUTES = qw (
```

```
    PortalMerchantID
    ShortDescriptionAttribute
);


sub ErrorDefinition { return {
    'ValueTooLong' => 'attribute ( #AttributeName ) for product (
#ProductID ) is too long (Maximum #MaxLength characters). Value has
been cut.',
}}


sub exportXML {
    my $self = shift;
    my ($PathFileName, $hFormats, $MyPortalConfig, $NoMonitor) = @_;
    $hFormats = {'encoding'=>'UTF-8'} unless defined $hFormats;
    $self->addMyPortalHooks;
    $self->{$_} = $MyPortalConfig->get($_) foreach
@ADD_MyPortal_ATTRIBUTES;
    $self->{'MinorUnit'} = GetInfoCurrency($MyPortalConfig-
>get('CurrencyID'))->{'MinorUnit'};


    $self->SUPER::exportXML($PathFileName, $hFormats,
$MyPortalConfig, $NoMonitor);
    return;
}


sub portalMerchantID          { return shift->{'PortalMerchantID'}; }
sub shortDescriptionAttribute { return shift-
>{'ShortDescriptionAttribute'}; }
sub minorUnit                 { return shift->{'MinorUnit'}; }


sub drive {
    my $self = shift;
    my ($aObjects) = @_;


    $self->SUPER::drive($aObjects);
    $self->addElement( 'Products', {});
}


sub addMyPortalHooks {
    my $self = shift;
    $self->addHook( 'Products' => 'FeedOfMyPortalConfig' );
    $self->addHook( 'Product'  => 'MerchantCategoryByProduct' );
    $self->addHook( 'Product'  => 'IDByProduct' );
```

```
    $self->addHook( 'Product'  => 'NameByProduct' );
    $self->addHook( 'Product'  => 'BrandByProduct' );
    $self->addHook( 'Product'  => 'PromotionTextByProduct' );
    $self->addHook( 'Product'  => 'DeepLinkByProduct' );
    $self->addHook( 'Product'  => 'ImageUrlByProduct' );
    $self->addHook( 'Product'  => 'DeliveryByProduct' );
    $self->addHook( 'Product'  => 'PriceByProduct' );
    $self->addHook( 'Product'  => 'ShippingByProduct' );
    $self->addHook( 'Product'  => 'VendorCodeByProduct' );
    $self->addHook( 'Product'  => 'CurrencyByProduct' );
}

#=====================================================================
====================
# $function     isProductValidPortal
# $state        public
#---------------------------------------------------------------------
--------------------
# $syntax       $self->isProductValidPortal;
#---------------------------------------------------------------------
--------------------
# $description  Throws error if not valid (not visible, no name, no
price, no assigned
#               category or no manufacturer).
#=====================================================================
====================
sub isProductValidPortal {
    my $self = shift;
    my ($Product) = @_;

    $self->isProductValidPortalExt1($Product);

    return;
}

#=====================================================================
====================
# $function     formatPrice
# $state        public
#---------------------------------------------------------------------
--------------------
# $syntax       $FormattedValue = $self->formatPrice($Value);
# $example      $FormattedShippingValue = $self->formatPrice($self-
>shippingPrice($Product));
```

```
#-------------------------------------------------------------------------
--------------------

# $description  formats prices (from shipping or products) to portal
specific format.

#=========================================================================
====================

sub formatPrice {

    my $self = shift;

    my ($Value) = @_;


    # round the price according to currency setting and add trailing
zeros if necessary

    my $Digits = $self->minorUnit;

    my $FormattedValue = $Digits ? sprintf( "%0.${Digits}f", $Value )
: $Value;

    return $FormattedValue;

}


sub FeedOfMyPortalConfig {

    my $self = shift;


    my $aProductIDs = $self->items;

    my $Monitor = $self->monitor;


    #grep right products

    foreach my $ProductID (@$aProductIDs ) {

        my $Product = LoadObject($ProductID);

        # monitor handling

        if (defined $Monitor) {

            $self->monitorIncrement;

            $self->monitorContent;

        }

        eval {

            $self->isProductValidPortal($Product);

        };

        if (ExistsError() ){

            Warning(GetError()) ;

        } else {

            $self->addElement( 'Product', { 'Product' => $Product }
);

        }

    }

    return;
```

```
}

sub getProductCategory {
    my $self = shift;
    my ($Product) = @_;
    my $LanguageID = $self->languageID;

    my $Categories  = $Product->getWithParent('Categories');
    my @CategoryPathNames;
    foreach my $Category (@$Categories) {
        my $aCategoryPathParts = GetMyPortalCategoryPath($Category,
undef, $LanguageID);
        next if not defined $aCategoryPathParts;
        if (scalar @$aCategoryPathParts) {
            @$aCategoryPathParts = reverse @$aCategoryPathParts;
            my $CategoryPathPart = join ('>', @$aCategoryPathParts);
            push @CategoryPathNames, $CategoryPathPart;
        }
    }
    my $MerchantCategoriesString;
    if (scalar @CategoryPathNames) {
        $MerchantCategoriesString = join (';', @CategoryPathNames);
    }
    $MerchantCategoriesString = substr($MerchantCategoriesString, 0,
255) if defined $MerchantCategoriesString;
    return $MerchantCategoriesString;
}

sub GetMyPortalCategoryPath {
    my ($Category, $aCategoryPathParts, $LanguageID) = @_;

    my $Shop   = $Category->getSite();
    my $Parent = $Category->parent;
    return $aCategoryPathParts if ($Parent->id == $Shop->id);

    my $CategoryPath = $Category->get('NameOrAlias', $LanguageID);
    push @$aCategoryPathParts, $CategoryPath;
    GetMyPortalCategoryPath($Parent, $aCategoryPathParts,
$LanguageID);
}

#-----------------------------------------
```

```
# export MerchantCategory
sub MerchantCategoryByProduct {
    my $self = shift;
    my $LanguageID  = $self->languageID;
    my $Product     = $self->topID('Product');

    my $aCategories  = $Product->getWithParent('Categories');
    my $Category = $aCategories->[0];
    my @CategoryPathNames;
    my $aCategoryPathParts = defined $Category
        ? GetMyPortalCategoryPath($Category, undef, $LanguageID)
        : undef;

    return if not defined $aCategoryPathParts;
    my $CategoryPathPart;
    if (scalar @$aCategoryPathParts) {
        @$aCategoryPathParts = reverse @$aCategoryPathParts;
        $CategoryPathPart = join ('>', @$aCategoryPathParts);
    }
    $CategoryPathPart = substr($CategoryPathPart, 0, 255) if defined
$CategoryPathPart;
    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'MerchantCategory',
'MaxLength' => 255}) if defined $CategoryPathPart and
length($CategoryPathPart) > 255;
    $self->addElement( 'MerchantCategory', {}, $CategoryPathPart ) if
defined $CategoryPathPart;
}

# export OfferID
sub IDByProduct {
    my $self = shift;
    my $Product = $self->topID('Product');
    my $LanguageID = $self->languageID;
    my $Alias = $Product->getWithParent('Alias',$LanguageID);
    $Alias = substr($Alias, 0, 50) if defined $Alias;
    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'OfferID', 'MaxLength' =>
50}) if defined $Alias and length($Alias) > 50;
    $self->addElement( 'OfferID', {}, $Alias ) if defined $Alias;
    return;
}
```

```
# export Name
sub NameByProduct {
    my $self = shift;
    my $Product = $self->topID('Product');
    my $LanguageID = $self->languageID;
    my $Name = $Product->getWithParent('Name',$LanguageID);
    $Name = substr($Name, 0, 200) if defined $Name;
    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'Name', 'MaxLength' => 200})
if defined $Name and length($Name) > 200;
    $self->addElement( 'Name', {}, $Name ) if defined $Name;
    return;
}


# export Brand
sub BrandByProduct {
    my $self = shift;
    my $Product = $self->topID('Product');
    my $Brand = $Product->getWithParent('Manufacturer');
    $Brand = substr($Brand, 0, 100) if defined $Brand;
    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'Brand', 'MaxLength' => 100})
if defined $Brand and length($Brand) > 100;
    $self->addElement( 'Brand', {}, $Brand ) if defined $Brand;
    return;
}


# export Description
sub PromotionTextByProduct {
    my $self = shift;
    my $Product              = $self->topID('Product');
    my $LanguageID           = $self->languageID;

    my $DescriptionAttribute = $self->descriptionAttribute;
    if ( defined $DescriptionAttribute ) {
        my $Alias = $DescriptionAttribute->get('Alias');
        my $Value = $Product->getWithParent($Alias, $LanguageID );
        if (defined $Value) {
            $Value = RemoveHtmlTags($Value);
        }
        $Value = substr($Value, 0, 4000) if defined $Value;
```

```
        Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'Description', 'MaxLength' =>
4000}) if defined $Value and length($Value) > 4000;

        $self->addElement( 'Description', {}, $Value) if defined
$Value;

    }

}


# export column DeepLink

sub DeepLinkByProduct {

    my $self = shift;

    my $Product = $self->topID('Product');


    my $Value = $self->productUrl($Product);

    $Value = substr($Value, 0, 250) if defined $Value;

    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'DeepLink', 'MaxLength' =>
250}) if defined $Value and length($Value) > 250;

    $self->addElement( 'DeepLink', {}, $Value) if defined $Value;

}


# export ImageURL

sub ImageUrlByProduct {

    my $self = shift;

    my $Product = $self->topID('Product');


    my $image = $self->imageUrl($self->topID('Product'), $self-
>imageAttribute);

    $image = substr($image, 0, 250) if defined $image;

    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'ImageUrl', 'MaxLength' =>
250}) if defined $image and length($image) > 250;

    $self->addElement( 'ImageUrl', {}, $image) if defined $image;

}


# export Delivery

sub DeliveryByProduct {

    my $self = shift;

    my $Product = $self->topID('Product');

    my $LanguageID = $self->languageID;

    my $DeliveryPeriod = $Product->getWithParent('DeliveryPeriod');

    $DeliveryPeriod = substr($DeliveryPeriod, 0, 30) if defined
$DeliveryPeriod;
```

```
    Warning('ValueTooLong', {'ProductID'=>$Product->id,
'Product'=>$Product, 'AttributeName' => 'Delivery', 'MaxLength' =>
30}) if defined $DeliveryPeriod and length($DeliveryPeriod) > 30;
    $self->addElement( 'Delivery', {}, $DeliveryPeriod) if defined
$DeliveryPeriod;
}

# export Prices
sub PriceByProduct {
    my $self = shift;

    my $Value = $self->productPrice($self->topID('Product'));

    # round the price according to currency setting and add trailing
zeros if necessary
    my $Digits = GetInfoCurrency($self->currencyID)->{'MinorUnit'};
    $Value = sprintf( "%0.${Digits}f", $Value ) if $Digits;

    $self->addElement( 'Prices', {}, $self->formatPrice($Value));
}

# export ShippingCost
sub ShippingByProduct {
    my $self = shift;
    my $Product     = $self->topID('Product');

    my $ShippingPrice = defined $self->shippingMethod ? $self-
>shippingPrice($self->topID('Product')) : 0;
    $self->addElement( 'ShippingCost', {}, $self-
>formatPrice($ShippingPrice));
}

# export Vendor_Code
sub VendorCodeByProduct {
    my $self = shift;
    my $Product     = $self->topID('Product');

    my $SKU = $Product->get('ManufacturerSKU');
    $self->addElement( 'Vendor_Code', {}, $SKU) if defined $SKU;
}

# export Currency
sub CurrencyByProduct {
```

```
    my $self = shift;
    $self->addElement( 'Currency', {}, $self->currencyID );
}

#-----------------------------------------

sub Products {
    my $self = shift;

    my $MerchantID = $self->portalMerchantID;
    return { 'MyPortalID' => $MerchantID } if defined $MerchantID;
}

1;
```

Summary

- ✓ **XML-formatting can also be handled when exporting to a portal server.**