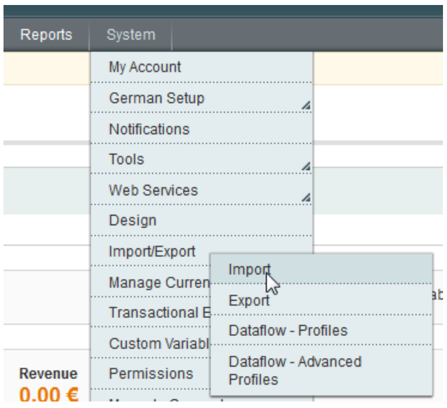
Importing Products with the ImportExport Interface

This article is about how to use the new "ImportExport" interface that has been introduced with Magento 1.5 (CE) and Magento 1.10 (PE and EE). I will demonstrate how to import product data via CSV files as well as elaborate on the characteristics of said file format. The main focus of this lecture will be on those functionalities that go beyond importing simple products: multilingualism, categories, tier prices, product images, product links, and grouped products.

For further reading I recommend the lecture of the slides of Vinai Kopp's Imagine 2011 talk. In it, the author explains how to import configurable products and individual options, which is why I will omit those for now, as well as developing custom converters for other file formats.

The ImportExport Interface

The **ImportExport** module replaces the old **Dataflow** module, which has been used in earlier versions for importing and exporting data. Both modules only allow the import and export of product and customer data to and from text files. Both modules are accessible both from the backend system and within custom modules. The main advantage of the ImportExport module is its **speed**, which has been **increased considerably**. Whereas the old module allowed the import of only a few products per second, the new one imports several hundred products per second.



You will find the import option in the Magento admin panel: System -> Import/Export -> Import.

In order to import product data with the ImportExport module, the data needs to be available in a specific format. Per default, the module only imports CSV files. Using my small FastSimpleImport module and a simple function call, one can also import PHP arrays, which can be generated within a custom module from any data source via PHP. This function call looks like this:

```
Mage::getSingleton('fastsimpleimport/import')
->processProductImport($productData);
```

Details can be found in my German blog article about the module or on GitHub.

Data Structure

The data that is to be imported needs to be available in a predefined format. For my examples I will use the default CSV format. The same structure can be used for other import methods as well.

In general, many characteristics of a file format can be gleaned from exporting existing product data und then taking a look at the resulting CSV file. The import format is basically identical, apart from the fact that the export does not support all functionalities that exist for the import, e.g. grouped products.

For CSV files, the following format applies:

- Column names in the first table row
- Field separator: comma
- Text separator: double quotation marks (optional)
- Character set: UTF-8

A file for importing basic product data can for example look like this:

```
sku,_type,_attribute_set,_product_websites,name,price,description,short_description,weight,sta
tus,visibility,tax_class_id,qty
```

```
1234567,simple,Default,base,Default,0.99,Default,Default,0,1,4,2,76
```

For clarity I will present the data in a table format:

sku	_type	_attribute_set	_product_websites	name	price	description	short_description	weight	status	visibility	tax_class_id	qty
1234567	simple	Default	base	Default	0.99	Default	Default	0	1	4	2	76

This table contains the **required fields** for new products. For existing products, fields can be omitted as preferred, except for the **sku** field, which is essential for identifying the product in question.

Updating the stock of two products could therefore look like this:

sku	_type	_attribute_set	_store	qty
1234567				74
1234568				12

The general rule is that the **_type**, **_attribute_set,** and **_store** columns need to exist for product updates as well; they may however contain the value **null** if you do not want to overwrite the existing value. For clarity I will omit those columns in the following examples.

Optional attributes (e.g. **cost**, **special_price**) can simply be added as additional columns. The same goes for custom created attributes (e.g. **color** or **manufacturer**).

Select fields

As select fields I label those fields that have predefined values, e.g. **status** (active or deactivated) or **color** (custom values). While the above listed fields **status**, **visibility**, and **tax_class_id** expect an ID value, other attributes need to be provided with specific option values:

sku	color	is_imported
1234567	red	yes
1234568	blue	yes

Please note that these attributes require global option values, not localized ones.

Unfortunately, the interface is unable to create missing option values by itself. This will therefore need to be done in a preceding step.

Multilingualism

For importing multi-lingual texts, a new row needs to be created for each store view. For example like this:

sku	_store	name	Description
1234567		Default	Default Description
	de	Standard	Default Description German
	en	Default	Default Description English

The **_store** column contains the store view code.

It is important that the **sku** field is only filled for the first one of those rows pertaining to the product. All further rows will automatically be assigned to that product. This mechanism is also used for other fields that may have several different values. These will be specified below.

Multiple Websites

Like multiple languages, multiple websites are each listed in a separate row:

sku	_product_websites
1234567	website_code_1
	website_code_2

Categories

Categories are identified by their names (default values), using the complete path, separated by slashes. For example like this:

sku	_category
1234567	Electronics/Cameras/Digital Cameras
	Apparel/Shoes/Mens

As above, values are each listed in a separate row.

Tier Prices

Tier prices require a number of special fields that can be filled. How to import scaled prices can best be shown by the following example:

sku	_tier_price_website	_tier_price_customer_group	_tier_price_qty	_tier_price_price
1234567	all	1	10	0.89
	all	1	20	0.79
	all	2	10	0.85
	all	2	20	0.70
1234568	website_code_1	all	10	16.50

In this case, "all" is the default value for "all websites" or "all customer groups".

Product Images

Importing product images has only been possible since Magento versions 1.6 (CE) and 1.11. The images that are to be imported need to be available in the media/import folder. The corresponding rows of the import file need to look like this:

sku	_media_image	_media _attribute_id	_media _is_disabled	_media _position	_media _lable	image	small _image	thumbnail
1234567	img1.jpg	77	1	1	Image 1	img1.jpg	img2.jpg	img2.jpg
	img2.jpg	77	0	2	Image 2			
	img3.jpg	77	0	3	Image 3			

Some annotations:

- The file name of the image that is to be imported is listed in the **_media_image** column.
- The **_media_attribute_id** column required the ID of the product attribute "image_gallery". It can be obtained either from the admin panel or your custom modules using the expression `Mage::getSingleton('catalog/product')->getResource()->getAttribute('media_gallery') ->getAttributeId();`
- The typing error **_media_lable** is intentional and hard-wired into Magento.
- In the **image**, **small_image**, and **thumbnail** columns, the first row contains the file names of the according main images.
- Unfortunately, existing images can not be deleted using the module.

Up-Selling, Cross-Selling, Related Products

Even linked products can be imported using the following pairs of fields:

- **_links_related_sku** and **_links_related_position**
- **_links_crosssell_sku** and **_links_crosssell_position**
- **_links_upsell_sku** and **_links_upsell_position**

An example could look like this:

sku	**_links_upsell_sku**	**_links_upsell_position**	**_links_crosssell_sku**	**_links_crosssell_position**
1234567	1234568	1	1234569	1
	1234569	2		

Grouped Products

Grouped products can also be imported. Two children products assigned to one grouped product should look like this:

sku	**_type**	**_associated_sku**	**_associated_position**	**_associated_default_qty**
1234567	grouped	1234568	1	1
		1234569	2	1

Contrary to the functionality accessible via Magento's admin panel, this is a way to edit product types afterwards.

Configurable Products and Individual Options

These functions are also supported. Since, however, Vinai has already presented those in detail in his lecture, let me point you to his lecture materials.

Indexing

It should be noted that when importing data using the ImportExport module, indices are not automatically updated. They are however marked as invalid and will have to be updated separately via the admin panel or a custom module.

Conclusion

The ImportExport module is an important improvement for Magento, because it allows the **fast** and **reliable** importing of products by default. Personally, I use this module for all my product imports (and for all my customer data imports), because it innately offers many functionalities and is relatively easy to use (especially when combined with my small FastSimpleImport module). Little weaknesses and cosmetic flaws can be overlooked in this case.

With this article I hope to have given some helpful instructions on how to better understand und use the ImportExport module.