Integrations and Data Management

Generated on: 2021-01-25 18:53:38 GMT+0000

SAP Commerce | 6.7.0

PUBLIC

Original content: https://help.sap.com/viewer/50c996852b32456c96d3161a95544cdb/6.7.0.0/en-US

Warning

This document has been generated from the SAP Help Portal and is an incomplete version of the official SAP product documentation. The information included in custom documentation may not reflect the arrangement of topics in the SAP Help Portal, and may be missing important aspects and/or correlations to other topics. For this reason, it is not for productive use.

For more information, please visit the https://help.sap.com/viewer/disclaimer.

ImpEx Syntax

SAP Commerce ships with an integrated text-based import/export extension called ImpEx, which allows creating, updating, removing, and exporting platform items such as customer, product, or order data to and from Comma-Separated Values (CSV) data files - both during run-time and during the SAP Commerce initialization or update process.

CSV Files

The `ImpEx` extension uses Comma-Separated Values (CSV) files as the data format for import and export. As there is no formal specification, there is a wide variety of ways in which you can interpret CSV files.

You can find some sort of quasi-standard in Common Format and MIME Type for Comma-Separated Values (CSV) Files document, as it is followed by most implementations. It is quoted here in excerpts:

1. Each record is located on a separate line, delimited by a line break (CRLF), for example:

```
aaa,bbb,ccc CRLF
zzz,yyy,xxx CRLF
```

2. The last record in the file may or may not have an ending line break, for example:

```
aaa,bbb,ccc CRLF
zzz,yyy,xxx
```

3. There may be an optional header line appearing as the first line of the file with the same format as normal record lines. This header contains names corresponding to the fields in the file and should contain the same number of fields as the records in the rest of the file, for example:

```
field_name,field_name,field_name CRLF
aaa,bbb,ccc CRLF
zzz,yyy,xxx CRLF
```

The presence or absence of the header line should be indicated using the optional `header` parameter of this MIME type.

4. Within the header and each record, there may be one or more fields, separated by commas. Each line should contain the same number of fields throughout the file. Spaces are considered part of a field and should not be ignored. The last field in the record must not be followed by a comma, for example:

```
aaa,bbb,ccc
```

5. Each field may or may not be enclosed in double quotes (however some programs, such as Microsoft Excel, do not use double quotes at all). If fields are not enclosed with double quotes, then double quotes may not appear inside the fields, for example:

```
"aaa","bbb","ccc" CRLF
zzz,yyy,xxx
```

6. Fields containing line breaks (CRLF), double quotes, and commas should be enclosed in double-quotes, for example:

```
"aaa","b CRLF
bb","ccc" CRLF
```

```
zzz,yyy,xxx
```

7. If double-quotes are used to enclose fields, then a double-quote appearing inside a field must be escaped by preceding it with another double quote, for example:

```
"aaa","b""bb","ccc"
```

Due to the simple structure, CSV files are common for data exchange purposes. Compared to XML files, CSV files require less processing and memory resources. For additional details on CSV files, please refer to Common Format and MIME Type for Comma-Separated Values (CSV) Files document.

In a CSV file, by default, one line of entries represents one record of data. In other words: in a CSV file, one line defines one item, that is one customer, one car, one t-shirt, and so on. A single line can contain several attribute values, for example in case of a t-shirt: size, color, and brand. CSV breaks down individual record entries via a so-called delimiter, which is, by default, the comma, hence the name. The following fictitious code sample gives you an example of this:

```
green,42,BlackLabel
red,42,BlackLabel
green,35,BlackLabel
```

CSV Files in ImpEx

The `ImpEx` extension relies on the above definition of the CSV format except for the defined special characters, which may be different but can be configured. These characters are:

- Line break: character for separating one line from another.

  Default is the Java system property `line.separator`.

- Column delimiter: isolates values that belong in columns, in other words records.

  Default is the semicolon (;). You can change it using the `csv.fieldseparator` property.

- Enclosing character: character for enclosing a column. Default is the double-quote ". You can change it using the `csv.quotecharacter` property.

- Individual delimiter: these delimiters isolate values that belong in one database field, such as with Collections (here the default is comma (,)). The individual delimiters depend on the column translator used and can be configured separately. Refer to Header.

Furthermore, the `ImpEx` extension provides the multi-line separator symbol for breaking lines only visually (soft line break). The `ImpEx` extension does not interpret the line break. The support of multi-lines can be disabled using the API.

For more information, see:

- https://docs.oracle.com/.../getProperties() : **getProperties()** documentation
- Configuration Properties
- JavaDocs and API Documentation

Managing CSV Files Using Third Party Tools

The default value for these delimiters depends on the system locale settings when using third party tools. Using an ImpEx CSV file on different locale settings has to be handled with care. For example, during the tests a German version of Microsoft Excel 2003 SP2 used commas for individual delimiters and semicolons for column delimiters by default if running on an English version of Microsoft Windows XP SP2. The default delimiters for Microsoft Windows XP versions with an English locale are semicolons for individual delimiters and commas for column delimiters.

The following code snippet, for example, would cause different interpretations, depending on the locale settings:

```
value1,value2,value3;value4;value5,value6
```

German locale	English locale
three values: - **value1,value2,value3** - **value4** - **value5,value6**	four values: - **value1** - **value2** - **value3;value4;value5** - **value6**

Furthermore, you can encounter serious problems when switching the editing programs because of the different character encoding used. Saving a CSV file in Windows encoding and importing it using **ImpEx** in UTF-8 encoding destroys special characters like umlauts. Always check the character encoding used by the program and configure it as described in ImpEx API.

The most convenient way of dealing with CSV files is using an editor with syntax highlighting or spreadsheet processor. Among such software, SAP recommends the following applications:

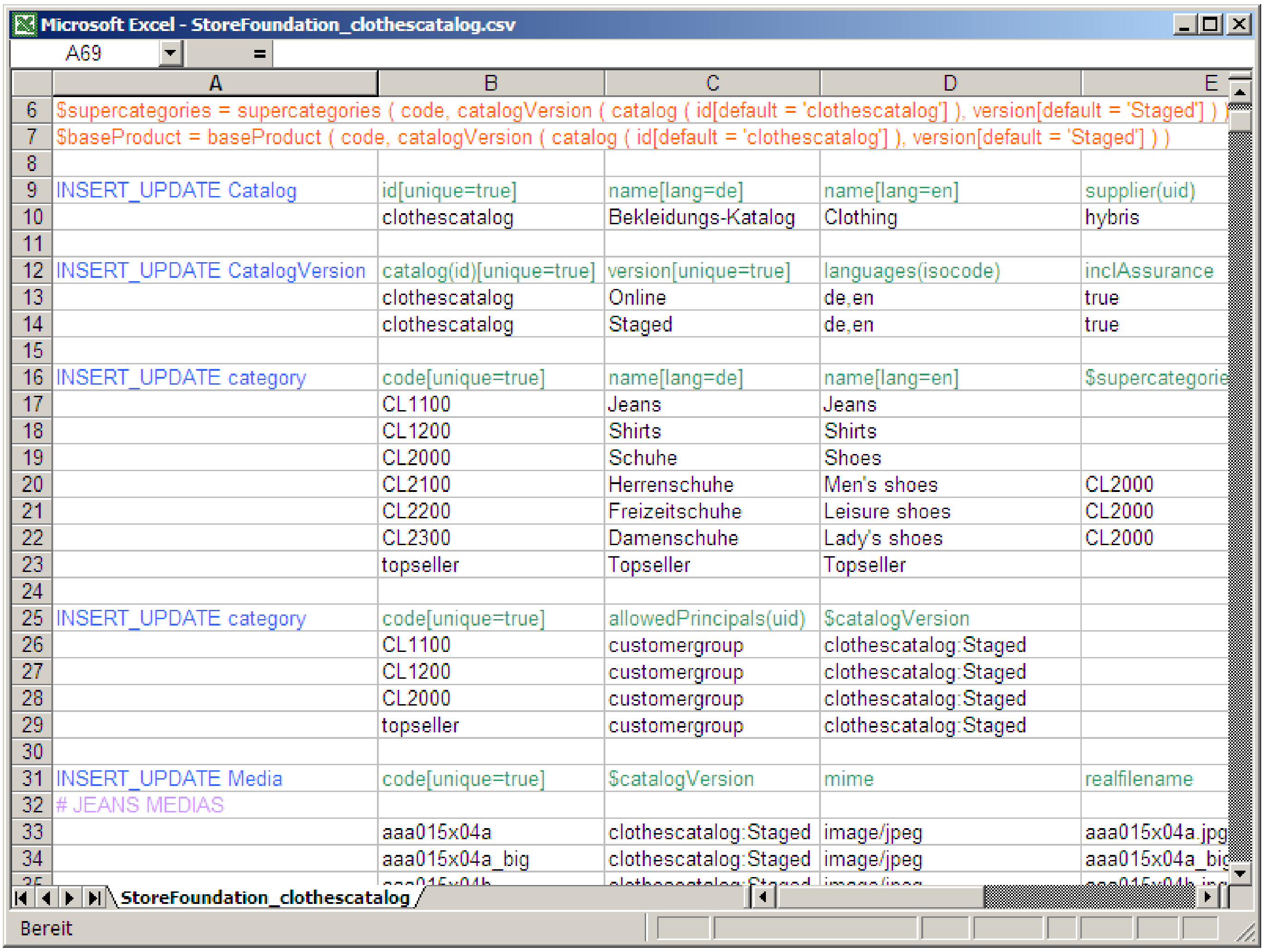
- UltraEdit - text editor that contains customizable syntax highlighting.
- Microsoft Office Excel - a powerful commercial spreadsheet processor. Microsoft Excel CSV spreadsheets are limited to a maximum of 65535 lines. This means that if you want to manage CSV files longer than 65535 lines, you have to split those files into parts. If you try to open a file longer than 65535 lines, Excel informs you that the file is beyond the specification. The problem was partially resolved since Microsoft Excel 2003, although the application supports more than 65535 lines only in its native format.
- OpenOffice.org Calc - freeware spreadsheet processor from Sun Microsystems Inc. It is free and also supports a customized import of CSV files (you can use the usual open file method, no extra import is needed). When saving a file as CSV the original encoding is used, or windows-1252 if the file was created from scratch.

ImpEx Syntax in CSV Files

An ImpEx-compliant CSV file contains several different kinds of data. The following screenshot is a representation of a sample CSV file with different colors for different kinds of data.

The different line types are described in detail below:

- Headers (blue and light green text color in the screenshot), refer to Header.
- Lines of values (**black** text color), refer to Value Line.
- Comments (lavender text color), refer to Comment section.
- Macro definitions (red text color), refer to Definition Line - Macro.
- BeanShell calls (not shown in the screenshot), refer to the *BeanShell* section of ImpEx API.
- Lines declaring user rights import (not shown in the screenshot), refer to the *User Rights* section of ImpEx API.


Microsoft Excel - StoreFoundation_clothescatalog.csv
A69
=
StoreFoundation_clothescatalog
Bereit

	A	B	C	D	E
6	$supercategories = supercategories (code, catalogVersion (catalog (id[default = 'clothescatalog']), version[default = 'Staged']))				
7	$baseProduct = baseProduct (code, catalogVersion (catalog (id[default = 'clothescatalog']), version[default = 'Staged']))				
8					
9	INSERT_UPDATE Catalog	id[unique=true]	name[lang=de]	name[lang=en]	supplier(uid)
10		clothescatalog	Bekleidungs-Katalog	Clothing	hybris
11					
12	INSERT_UPDATE CatalogVersion	catalog(id)[unique=true]	version[unique=true]	languages(isocode)	inclAssurance
13		clothescatalog	Online	de,en	true
14		clothescatalog	Staged	de,en	true
15					
16	INSERT_UPDATE category	code[unique=true]	name[lang=de]	name[lang=en]	$supercategorie
17		CL1100	Jeans	Jeans	
18		CL1200	Shirts	Shirts	
19		CL2000	Schuhe	Shoes	
20		CL2100	Herrenschuhe	Men's shoes	CL2000
21		CL2200	Freizeitschuhe	Leisure shoes	CL2000
22		CL2300	Damenschuhe	Lady's shoes	CL2000
23		topseller	Topseller	Topseller	
24					
25	INSERT_UPDATE category	code[unique=true]	allowedPrincipals(uid)	$catalogVersion	
26		CL1100	customergroup	clothescatalog:Staged	
27		CL1200	customergroup	clothescatalog:Staged	
28		CL2000	customergroup	clothescatalog:Staged	
29		topseller	customergroup	clothescatalog:Staged	
30					
31	INSERT_UPDATE Media	code[unique=true]	$catalogVersion	mime	realfilename
32	# JEANS MEDIAS				
33		aaa015x04a	clothescatalog:Staged	image/jpeg	aaa015x04a.jpg
34		aaa015x04a_big	clothescatalog:Staged	image/jpeg	aaa015x04a_big
35		aaa015x04b	clothescatalog:Staged	image/jpeg	aaa015x04b.jpg

Header

A header is a single line defining a mapping of the following value lines to the type system. A header applies to all processed value lines until the next header or until the end of file, whichever comes first. You can put any number of headers into a CSV file.

An ImpEx header line has the following structure:

```
mode type[modifier=value];attribute[modifier=value];attribute[modifier=value];attribute[modifier=value
```

- **Mode**: Specifies what is to be done with the following value lines (insert, update, and so on) - see Header Mode.
- **Type**: Defines the type of item to be processed (category, product, media, type, and so on) - see Header Type.
- **Attribute**: Describes which item attributes the columns are mapped to. The value lines supply the actual values for the items that are translated using the header settings - see Header Attributes
- **Modifier**: Gives additional information for translating a value record to the mapped type attribute - see Header and Attribute Modifier.

Example for a header row:

```
INSERT_UPDATE category;code[unique=true];name[lang=de];name[lang=en];$supercategories;$thumbnail;descr
```

This header states that each following value line creates or updates a category instance, until another header occurs.

i Note

The whole header syntax is case insensitive including the attribute qualifiers.

Related Information

The Type System

Header Mode

The mode of a header specifies what is to be done with the following data.

In case of an import it describes how to map and translate the following value lines to the data model of the platform. In case of an export, it describes how to map and translate the following items to the target CSV file. During export, headers are also written, so exported CSV files can be re-imported via the `ImpEx` extension.

There are four header modes:

Mode	Comment/Description
INSERT	Creates a new Item in SAP Commerce from the processed value line. By default, ImpEx does not check if an item with the already exists. Therefore, trying to insert an item with a code that already exists may throw an exception caused by the SA that reason, it is recommended to use the unique modifier, where ImpEx prechecks a violation of attribute uniqueness.

UPDATE	Selects an existing item in SAP Commerce from the combination of the values of all columns marked with the unique moc referred to as key attributes, and sets the item attributes to the values specified in the value line.
	If there is no item that matches the values of all key attributes, the value line cannot be resolved. The value line is dumpec and the `ImpEx` extension tries to resolve the value line later on. Refer to ImpEx API, section **Import Using an Importer Ins**
	Also note that the default value for attributes is `null`. This means that any attributes that are referenced by the header b are specified in the value line are not ignored, but are explicitly set to `null`. For example, the following ImpEx lines cause t have the `name` attribute set to null as the second line after the header specifies no value for the `name` attribute.
	`UPDATE User;uid[unique=true];name;customerID;`
	`;ahertz;Anja Hertz;;`
	`;ahertz;;K2006-C0005;`
	To set values for both the `name` and the `customerID` attributes, use one of the following approaches:
	• Specify all values in one single line:
	`UPDATE User;uid[unique=true];name;customerID;`
	`;ahertz;Anja Hertz;K2006-C0005;`
	• Re-specify all values that have been set in a prior line:
	`UPDATE User;uid[unique=true];name;customerID;`
	`;ahertz;Anja Hertz;;`
	`;ahertz;Anja Hertz;K2006-C0005;`
	• Use individual headers:
	`UPDATE User;uid[unique=true];name;`
	`;ahertz;Anja Hertz;`
	`UPDATE User;uid[unique=true];custmerID;`
	`;ahertz;K2006-C0005;`

INSERT_UPDATE	Combines the effects of the INSERT and UPDATE header modes: if an item with the values of all key attributes exists, this item exists that matches the combination of all key attributes, the value line is used to create the item. The `ImpEx` extens order: UPDATE, then INSERT.

Mode	Comment/Description
REMOVE	As in UPDATE mode, the `ImpEx` extension tries to find an existing item using the key attributes. If such an item exists, it is log message of level **warning** is printed to the log.

Header Type

The type code selects the type from the SAP Commerce Type System where the following data rows create, update or remove instances.

Optionally, each line can specify a subtype of the type specified in the header at the first column. That subtype is selected for that single line only - in the next line, the type specified in the header is active again. The following code snippet references the User type in the header, but creates an instance of the **Customer** and **Employee** types each.

```
INSERT User;uid[unique=true]
Customer;SampleCustomer
Employee;SampleEmployee
```

This allows specifying an abstract type, which cannot be instantiated, in the header for the INSERT mode, as with the Job type in the following code snippet. In that case, you need to specify a non-abstract type (that is, a type that can be instantiated) in every value line.

```
INSERT Job;code[unique=true]
ImpExImportJob;sampleJob
ImpExImportJob;sampleJob2
```

Although the header references the Job type, two instances of an ImpExImportJob will be created: sampleJob and sampleJob2.

Header Attributes

Each column followed by the header type at the header line describes an attribute of the header type where the column value of the following value lines are mapped to.

The `ImpEx` extension checks the specified type and its attributes and uses a Translator to match the value line's entry to the individual attribute. See Header and Attribute Modifier for more details on Translator.

It is easy to set an attribute value for attributes that are specified as AtomicTypes (such as `java.lang.String` or `java.lang.Integer`). For example, assume the following code snippet:

```
INSERT User;uid
;myUser
```

Here the `uid` attribute is of type `java.lang.String`, so additonal definitions are not required. You define the attribute at the header line and write a value line with the plain text of the desired user ID.

Item Reference Attributes

Setting attribute values is more complex with attributes specified to hold references to items in SAP Commerce.

By default, the ImpEx expects values that represent a primary key:

```
INSERT Product;code;unit
;MyProduct;2821057095693456
#2821057095693456 being the PK for the pieces unit
```

Since primary keys are not usually known externally, ImpEx alternatively allows using **attributes of the referenced item** for lookup. They're provided in brackets immediately after the column attribute name: `<attributeOfTypeX>'('<attributeFromX>(','<attributeFromX>)* ')'`.

For example, the following code snippet specifies the `unit` column to look up Unit items using its `code` attribute:

```
INSERT Product;code;unit(code)
;MyProduct;pieces
```

ℹ **Note**

Specifying referred-type attributes behind an attribute definition is called **item expression**.

You can use item expressions in a nested way, in cases where a lookup attribute is a reference to itself.

In the following example the `catalogVersion` attribute refers to a CatalogVersion item that is looked up using its `catalog` and `version` attribute. Within the catalog is a reference to itself (to a Catalog item) that is looked up using the `id` attribute:

```
INSERT Product;code;unit(code);catalogVersion(catalog(id),version)
;testCode;pieces;clothescatalog:Staged
```

If you specify an item expression using several attributes at a header line, you have to separate them with a comma. The values for such an expression at a value line are separated by a colon.

There are ComposedTypes that reference attributes of a very general type; for example, the `owner` attribute of the `Address` type is specified to be of the `Item` type and can therefore accept any item in the SAP Commerce. By consequence, you cannot directly reference an attribute specified for a subtype of `Item`, such as the `uid` attribute of a Customer instance, which would allow you to specify a specific customer. To refer to an attribute that belongs to a subtype of a type attribute, use the following syntax:

```
attribute(subtype.subtype_attribute)
```

For example, the following code snippet creates an `Address` type instance whose `owner` attribute (attribute of type `Item`) holds a reference to the admin user using the `Principal`'s type `uid` attribute.

```
INSERT Address;firstname;owner(Principal.uid)
;Hans;admin
```

Skipping Columns of Data

The `ImpEx` extension allows skipping of entire columns of data by omitting the attribute for that column in the header.

```
INSERT myProduct;myAttribute;myAttribute2;;myAttribute4
;myValue1;myValue2;myValue3;myValue4
```

`myValue3` is ignored, because there is no header attribute mapping defined for the related column. In the following code snippet, the second column (containing the value `deutsch`) is ignored as there is no header attribute to process it:

```
INSERT Language;code;;active
;de;deutsch;true
```

Localizing

Localized attributes are described by multiple columns using the `lang` modifier, each one containing the ISOcode or PK for a single language.

Refer to Header and Attribute Modifier section for more information about attribute modifiers.

```
INSERT Product;code[unique=true];name[lang=en];name[lang=de]
;myProduct1;myProduct1's localized name;lokalisierter Name von myProduct1
;myProduct2;myProduct2's localized name;lokalisierter Name von myProduct2
```

The `ImpEx` extension only overwrites localized values for the languages specified in the column header. Existing localized values for other languages remain unchanged, as the following example illustrates:

```
#create some products with a German and English name
INSERT Product;code[unique=true];name[lang=en];name[lang=de]
;myProduct1;myProduct1's localized name;lokalisierter Name von myProduct1
;myProduct2;myProduct2's localized name;lokalisierter Name von myProduct2

#overwrite the English name -> the German one remains untouched
UPDATE Product;code[unique=true];name[lang=en]
;myProduct1;"my product 1"
;myProduct2;
```

Product	Resulting English localization	Resulting German localization
myProduct1	my product 1	lokalisierter Name von myProduct1
myProduct2		lokalisierter Name von myProduct2

For more information, see Wikipedia on ISO Country Codes .

Document ID

Especially when importing `partOf` item values, you must reference these items by means other than the usual unique column technique because `partOf` items do not always provide a unique key but only hold their enclosing parent as a foreign key.

The `Customer.addresses` attribute provides an example of this. The customer owns addresses exclusively. The addresses hold the customer PK asa back reference. When using ImpEx, both customer and address have to be imported in separate lines.

Removing obsolete addresses requires that you fill the **Customer.addresses** cell with all currently valid addresses. This allows the `setter` method to remove obsolete ones. You must therefore reference valid addresses using a unique key, which addresses do not provide. For cases where a reference within the ImpEx script is required, ImpEx provides a special header attribute marked with the prefix **&** to specify a virtual ID called `Document ID`. Each imported item line can optionally define such a Document ID, which allows other lines to reference it later using the same attribute qualifier (with the **&** prefix).

Example:

```
INSERT Customer;uid[unique=true];...;defaultPaymentAddress(&addrID);...
;andi;...;addr1;...

INSERT Address;&addrID;owner(Customer.uid);...
;addr1;andi;...
```

The Document ID is case-sensitive, so the following code snippet would not work as the Document ID would not be recognized by the `ImpEx` extension (the document ID for the Customer type being **`&addrID`** and the header for the Address type being

`&addrid`).

```
INSERT Customer;uid[unique=true];...;defaultPaymentAddress(&addrID);...
;andi;...;addr1;...

INSERT Address;&addrid;owner(Customer.uid);...
;addr1;andi;...
```

Special Attributes

Sometimes you need to add data to an item that is not covered by an attribute, for example the media data. In these cases, the syntax of ImpEx provides a special kind of attribute definition called `special` attributes that do not have a mapping to a real attribute of the configured type.

They are marked with the @ symbol and always need the `translator` modifier, because there is no default implementation. Furthermore the specified translator has to be a realization of the SpecialValueTranslator interface.

The following code sample shows how to create a media with the data set from the specified file:

```
INSERT_UPDATE Media;code[unique=true];@media[translator=de.hybris.platform.impex.jalo.media.MediaDataT
;myMedia;file://c:/myMedia.txt
```

The type `Media` has no `media` attribute, but specifying the `media` attribute as a special attribute instructs ImpEx not to search for such an attribute. Instead, it will call the special translator specified with the translator modifier, which performs the import logic.

Alternative Pattern

For type defining attributes that belong to a generic type, such as the `Address` type `owner` attribute, which is specified to hold an instance of `Item`, the `ImpEx` extension allows you to specify Alternative Patterns in the header.

An Alternative Pattern specifies more than one attribute and makes the `ImpEx` extension determine the matching type instance from all the specified attributes. Alternative Patterns are processed from left to right.

The following code snippet, specifies two possible target attributes for the Address instances:

- the `Principal` type `uid` attribute
- the `AbstractOrder` type `code` attribute:

```
INSERT Address;firstname;owner(Principal.uid|AbstractOrder.code)
;Hans;admin
;Klaus;O-K2006-C0000-001
```

During import, ImpEx compares the values for the `owner` attributes against all available instances of the `Principal` and `AbstractOrder` types and automatically chooses the instance whose target attribute value (`uid` or `code`, respectively) matches the target value (`admin` and `O-K2006-C0000-001`, respectively).

When importing the first value line `;Hans;admin`, the `ImpEx` extension searches all instances of the `Principal` type to verify if any instance exists where the `uid` attribute is set to `admin`.

- If an instance exists (and there is, by platform default), the new instance of the `Address` type has its `owner` attribute set to a reference to that `Principal` type instance.

- If no instance exists of the Principal type whose uid attribute is set to admin, the ImpEx continues the retrieval by trying to find an instance of the AbstractOrder type whose code attribute is set to admin.
- If there was no instance of the AbstractOrder type whose code attribute is set to admin, the ImpEx extension would try to continue the retrieval on the next specified Alternative Pattern attribute. Since there would be no further Alternative Pattern attribute, the ImpEx extension would dump the line and try to resolve it on the next import pass.

  In this sample case, there is a Principal instance whose uid attribute is set to admin by platform default, so ImpEx is able to resolve the value line.

Basically, the same thing happens for the second value line ;Klaus;O-K2006-C0000-001: the ImpEx extension searches all instances of the Principal type if there is any instance whose uid attribute is set to O-K2006-C0000-001.

- If there was an instance (but there is none, by platform default), then the new instance of the Address type would have its owner attribute set to a reference to that Principal type instance.
- Since there is an instance of the AbstractOrder type whose code attribute is set to O-K2006-C0000-001 (at least for SAP Commerce installations with the sampledata extension enabled), the new instance of the Address type has its owner attribute set to a reference to that AbstractOrder type instance.
- If there was no instance of the AbstractOrder type whose code attribute is set to O-K2006-C0000-001, then the ImpEx extension tries to continue the retrieval on the next specified Alternative Pattern attribute. Since there would be no further Alternative Pattern attribute, the ImpEx extension would dump the line and try to resolve it on the next import pass.

  In this sample case, there is a AbstractOrder instance whose code attribute is set to O-K2006-C0000-001 by using the sampledata extension, so ImpEx is able to resolve the value line right now.

Header and Attribute Modifier

A modifier is an addition to an entry in a header line that specifies additional processing instructions.

For example:

```
...;attribute[modifier=value];...
```

The ImpEx extension allows specifying several modifiers at once, like this:

```
...;attribute[modifier=value,modifier=value,modifier=value];...
```

or

```
...;attribute[modifier=value][modifier=value][modifier=value];...
```

For example, the following code snippet defines that the value specified for the name attribute is the English localization version ("en" according to Wikipedia on ISO 639-1) of the name attribute.

```
name[lang=en]
```

To escape a modifier value, use the inverted comma '. To use an inverted comma in a modifier value, escape the inverted comma itself:

```
attribute[modifier='My modifier''s value']
```

Header-Related Modifiers

The following table contains a reference of all header-related modifiers. These modifiers apply to the entire header and are specified directly to the type referenced by the header, as in the following code snippet:

```
INSERT_UPDATE ClassificationSystemversion[cacheUnique=true];...
```

Modifier	Allowed values	Description
batchmode Import only	true / false Default is false	In UPDATE or REMOVE mode, the batch mode allows modifying more than one item that matches for a combination of all unique attributes of a value line. So, if for a value line more t one item is found that matches the combination of unique attributes, the attributes specifie non-unique are updated at all found items. `[batchmode=true]` • If the batch mode is not enabled (as by default), the hybris `ImpEx` extension throws exception if more than one item matches for a value line. • If the batch mode is enabled, the hybris `ImpEx` extension modifies any platform iten matches the value line.
cacheUnique Import only	true / false default is false	If this option is enabled, the CachingExistingItemResolver is used for existing item resolving case of update or remove mode) which caches by the combination of unique keys already resolved items. So when processing a value line first, it is tried to find the related item by searching the cache using the unique keys. The usage is only meaningful if an item has to be processed more than one time within a header scope. The maximum size of the cache is not restricted at the moment. `[cacheUnique=true]`
processor Import only	A ImportProcessor class Default is the DefaultImportProcessor	Unlike a Translator, which handles a certain column of a value line, a Processor handles an er value line. It contains all business logic to transform a value line into values for attributes of a item in SAP Commerce (such as calls for translator classes, for example) and performs the r setting of the values. In other words: a Processor is passed a value line as input, and it create updates an item in SAP Commerce as its output. `[processor=de.hybris.platform.impex.jalo.imp.DefaultImportProcess`
impex.legacy.mode (since SAP Commerce version 5.1.1)	true / false. Default: false	This modifer allows enabling the legacy mode (jalo) per header . This way - in case of service layer mode enabled globally, impex will switch dynamically to legacy mode, just like for existi 'allowNull' or 'forceWrite' modifiers. The only difference is - it's set for the type, not for the column. For example: `INSERT_UPDATE myProduct[impex.legacy.mode=true];myAttribute`

Attribute-Related Modifiers

Modifier	Allowed Values	Description
alias Export only	A text such as **myAttribute** Default: no alias is set	With that modifier you can specify an alias name for an attribute, whic Example: `[alias=myAttribute]`

Modifier	Allowed Values	Description
allownull Import only	true / false Default: false	If set to `true`, this modifier explicitly allows `null` values for the colun `null` values in mandatory attributes, such as the `catalogVersion` Example: `[allownull=true]` → Tip In the Service Layer mode, import may fail if `allownull` is set. Imp processing a given line, the import will switch back to the SL mode.
cellDecorator Import only	An AbstractImpExCSVCellDecorator class Default: no decorator is applied.	Specifies a decorator class for modifying the cell value before interpre Example: `[cellDecorator=de.hybris.platform.catalog.j`
collection-delimiter Restricted to attributes of type `Collection`.	Any character Default: comma (`,`)	Allows specifying a delimiter for separating values in a Collection. For Example: `INSERT_UPDATE Product;collectionAttribute[c`
dateformat Restricted to `Date` and `StandardDateRange` types.	A date format such as ***MM-dd-yyyy*** or ***dd.MM.yyyy*** (See the Date and Time Patterns in Java API documentation) Default: `DateFormat.getDateTimeInstance (DateFormat.MEDIUM, DateFormat.MEDIUM, impexReader.getLocale())`	Specifies the format in which the column specifies Date values. Example: `UPDATE myProduct;myAttribute [dateformat=dd` A more complex example would be to implement a ISO 8601 complian yyyy-MM-dd'T'HH:mm:ss.SSSZ. The T in the date format representat quotes have to be escaped by doubling. Example: `UPDATE myProduct;myAttribute[dateformat='yy`
default	A value that matches the current attribute column type.	Sets the default for values of this column Example: `INSERT_UPDATE myProduct;myAttribute[default`
forceWrite Import only	true / false Default: false	If set, it tries to write into read-only columns. Success depends on bus Example: `[forceWrite=true]` → Tip In the Service Layer mode, import may fail if `forceWrite` is set. lin processing a given line, the import will switch back to the SL mode.
ignoreKeyCase Import only Restricted to attributes of type `Item`	true / false Default: false	If set, the case of the text that specifies attributes of items for resolvin Example: `[ignoreKeyCase=true]`
ignorenull Only for import Restricted to attributes of type `Collection`.	true / false Default: false	For Collection-type attribute values, this allows ignoring of null values • If set to `false` as by default, a null value in a Collection-relate value line contained the value `myValue;null;myValue2`, tl null value, as in `myValue;null;myValue2`. • If set to `true`, a null value in a Collection-related column caus contained the value `myValue;null;myValue2`, then settin value, as in `myValue;myValue2`. Example: `INSERT_UPDATE myProduct;myAttribute[ignoreN`

Modifier	Allowed Values	Description
key2value-delimiter Restricted to attributes of type **Map**.	any character Default: "->"	Specifies the assignment delimiter for a key-value pair. Example: `INSERT_UPDATE myProduct;myAttribute[key2val`
lang Restricted to localized attributes (`Map<String, Language>`)	The ISO code or PK of a language defined for the SAP Commerce (such as de, en, or de_ch, for example) Default: session language	Specifies the language for a column containing localized attribute valu Example: `INSERT_UPDATE myProduct;myAttribute[lang=de`

map-delimiter
Restricted to attributes of type `Map`.
	Any character
	Default: ';'
		Specifies the delimiter between two key-value pairs.
		Example:
		`INSERT_UPDATE myProduct;myAttribute[map-delimiter`
		`;myKey->myValue|myKey2->myValue2;`

mode
Import only
Restricted to attributes of type **`Collection`**
	append, remove
	Default: replace (not explicitly specifiable) - an already existing Collection will be replaced
		Specifies the modification mode for `Collection` instances.
		• `mode=append`: In `append` mode, the references to the eleme
		• `mode=remove`: In `remove` mode, the references to the eleme
		• No mode specified (implicit `mode=replace`): Replaces the e
		For example, in the following code snippet,
		○ the myCategory category's `supercategories` attril `mySuperCategory2` categories (`mode=append`).
		○ In the case of `mode=remove`, the references to the `my` `myCategory` category's `supercategories` attribu
		○ For the default (no `mode` modifier specified, implicit re `mySuperCategory2`, overwriting all previous values.
		Example:
		`$catalogVersion=catalogVersion(catalog`
		`$supercategories=supercategories( code`
		
		`INSERT_UPDATE category;code[unique=tru`
		`;myCategory;meine Kategorie;mySuperCat`
		`;myCategory;meine Kategorie;mySuperCat`

Modifier	Allowed Values	Description

numberformat
Restricted to attributes of type `Number`
	A number format like ***#.###,##*** (For further information, refer to the `NumberFormat` class documentation by Sun Microsystems)
	Default:
	`NumberFormat.getNumberInstance( impexReader.getLocale() )`
		Specifies the format the column uses to specify Number values.
		Be aware that the delimiters for number values depend on the locale s `numberformat` modifier only. If you need to specify the delimiters ex delimiters using the BeanShell code.
		Example:
		`#% impex.setLocale( Locale.GERMAN );`
		The following code snippet shows two settings for the `numberforma`
		Example:
		`INSERT_UPDATE myProduct;myAttribute[numberformat=`
		`;1.299,99`
		`INSERT_UPDATE myProduct;myAttribute[numberformat=`
		`;1,299.99`

path-delimiter
Restricted to attribute of type `ComposedType`
	Any character
	Default: ":"
		Defines the delimiter to separate values for attributes using an item e
		Example:
		`INSERT_UPDATE myProduct;myAttribute(code,id)[path`
		`;myCode:myID`

pos	A positive integer Default: column number as positioned in header.	Specifies the column position where the values for this attribute can b a header description, it has to be used for all attributes in that header Example: [pos=3]
translator	An `AbstractValueTranslator` or a `SpecialValueTranslator` Default varies; each Type has a predefined default Translator.	A `Translator` class resolves a column of a value line into an attribut out-of-the-box `AtomicType` and `ComposedType` instances, such as the default `Translator` classes. Example: `INSERT_UPDATE myProduct;myAttribute[translator=de`
unique	true / false default: false	Marks this column as a key attribute - that is, the value is used to dete key attributes to check whether the item already exists. This means th must be unique. Each row must provide a unique set of values for ther item already exists. Example: `INSERT_UPDATE myProduct;myAttribute[unique=true]` • If this modifier is set to `false` as by default, an insert is alway • If set to `true`, the column contains key attribute values and a existing item will be updated by the non-unique values.

virtual	true / false
	Default: false
		If set, a related column within the following value lines is expected. Th
		Example:
		`INSERT MyProduct;myAttribute1[virtual=true,defaul`
		`;myValue2;`
		Sets myValue1 to attribute `myAttribute1` and `myValue2` to attribu This could lead to shiftings in the value line. For a better readability, it'

Value Line

Items are translated into a text representation called a value line completely driven by their business type definition. Generally, one item is represented by one value line - its attribute values by the columns of this row.

In a CSV file for the `ImpEx` extension, a value line holds a set of data such as a product definition with name, price and description, for example. This set of data spreads over several columns, each of which contains the value for an individual attribute. Within a row, columns are separated by a delimiter - the semicolon **;** by default, for example:

```
Value1;Value2;Value3;Value4;Value5;Value6
```

For each value line, an item is created, updated, or removed depending on the mode set by the header. The basic structure of a value line is:

```
value;value;value;value;value
```

Instead of a value, you can also skip entries - as in the following code sample where two values are skipped. A skipped value is replaced by the default value if the currently active header defined one. If no default value is defined, then a **null** value is passed to the translator class. What happens then depends on the translator implementation.

```
value;;;value;value
```

The value `<ignore>` makes the `ImpEx` extension skip the entry and leave the item value as it is. This is useful in combination with the UPDATE mode, for example. The resolving mechanism also makes use of this value. For more information about the resolving mechanism see the *Resolving Mechanism* section of ImpEx API.

```
value;<ignore>;<ignore>;value;value
```

Comment

Comments start with the dash **#** and are ignored during import. Blank lines and lines without values, such as ones occasionally generated by Microsoft Excel like ;;;;;;;;;, are also ignored.

```
INSERT_UPDATE Product;code[unique=true];varianttype(code);name[lang=en];
;;;;;;;;;;;;;;;
#;;;;;;;;;;;;;;;
#BASEPRODUCTS JEANS;;;;;;;;;;;;;;;
#;;;;;;;;;;;;;;;
```

Definition Line - Macro

The `ImpEx` extension allows you to define macros so that you do not have to type repeating strings into your CSV files, and you can keep your CSV files more manageable.

During import, these macros are parsed and any occurrence of the macro name is replaced by the macro value. You can call macros in headers and in value lines. You can even call macros within macro definitions. If you define two macros with the same key, the latest definition is used.

Macro definitions start with the dollar symbol (**$**), and they are referenced by `$macroname`. For example:

```
$catalog=catalog(id)
$catalogVersion=catalogVersion($catalog,version)

INSERT Product;code;$catalogVersion
```

Wherever the term $catalogVersion appears, it is replaced by: `catalogVersion(catalog(id),version)`:

```
INSERT Product;code;catalogVersion(catalog(id),version)
```

Abbreviations

Although the `ImpEx` header definition language provides the most flexible way of using custom column translators, header declaration can be rather long.

You could shorten them a bit using the ImpEx alias syntax **`$xyz=...`**. You can also use regexp patterns and replacements to shorten frequently-used definitions. The following example shows how to use them to shorten classification columns declarations by giving them a new syntax. Check if the syntax shown in the example below is in your SAP Commerce `local.properties` file:

```
impex.header.replacement.1 =
                                              C@(\\w+) ...
                                              @$1[ system='\\$systemName', version='\\$systemVersior
                                              translator='de...ClassificationAttributeTranslator']
```

i Note

The line has been wrapped to make it more readable. You must leave it in one line. Also note that the Java string notation has to be used, that is why there are double '\'s.

All parameters starting with impex.header.replacement are parsed as ImpEx column replacement rules. The parameter has to end with a number that defines the priority of the rule. This way ambigous rules could be sorted.

So what is this for? The first part of the property C@(\w+) defines the new abbreviation pattern to be used to declare classification attribute columns. The second part is the replacement text including the attribute qualifer match group $1. In fact, it contains the original special column declaration. Both parts are to be separated by '...'.

So all that is needed now to declare a classification attribute column is this:

```
$systemName=MySys
                                              $systemVersion=1.0

                                              INSERT_UPDATE Product; code[unique=true]; C@attr1; C@a
```

Now the whole translator definition is hidden and you do not need to declare any helper alias for that. Nevertheless both alias definitions are mandatory to tell the classification column which system and version it should take its attribute from.

ImpEx Syntax Highlighting with UltraEdit

You can use syntax highlighting rules in UltraEdit to have a more colored view upon ImpEx files.

Insert the following into your UltraEdit wordfile (you can find it at Advanced Configuration Editor Display Syntax Highlighting :

```
/L12"ImpEx" CSS_LANG Nocase Line Comment = # Line Comment Valid Columns = [0-1] Escape Char = \ String
/Delimiters = , : ;     ( ) [ ] ' " = | @ $
/Open Brace Strings =  "(" "["
/Close Brace Strings = ")" "]"
/C1"ImpEx Tags" STYLE_KEYWORD
END_USERRIGHTS
INSERT_UPDATE INSERT
REMOVE
START_USERRIGHTS
UPDATE
<ignore>
```

```
/C2"modifiers" STYLE_ATTRIBUTE
alias
allownull
batchmode
case
cellDecorator
collection-delimiter
dateformat
default
filter
forceWrite
ignoreKeyCase
ignorenull
key2value-delimiter
lang
map-delimiter
mode
numberformat
path-delimiter
pos
translator
unique
virtual

/C3"values"
append
false
remove
true

/C4"definitions"
** $
```

Language Support

UltraEdit supports up to twenty languages - `/L1` to `/L20` - in one wordfile at one time. The `/Lnn` number corresponds to the language's position in the Language Lists that are available through the following:

- View View As (Highlighting File Type)
- Advanced Configuration Editor Display Syntax Highlighting Language Selection

If you add a new language as **`/L20`**, for example, the language list looks like this:

```
8. ...
9. JavaScript
10. language 10
...
19. language 19
20. The New Language You Added
```

The default languages in `wordfile.txt` are numbered from /L1 to /L9. New languages that you added shouldn't conflict with those, or with each other. For example, Python 2 and and Unix Shell Scripts are **both** numbered /L20. So one or the other has to change, if both extensions are to be added. To add new languages, append the contents of appropriate wordfile to `wordfile.txt`.