ImpEx

The Context…

When you need to **import** or **export** state data **into** or **from** SAP Commerce Cloud, **ImpEx** is just the tool for the job!

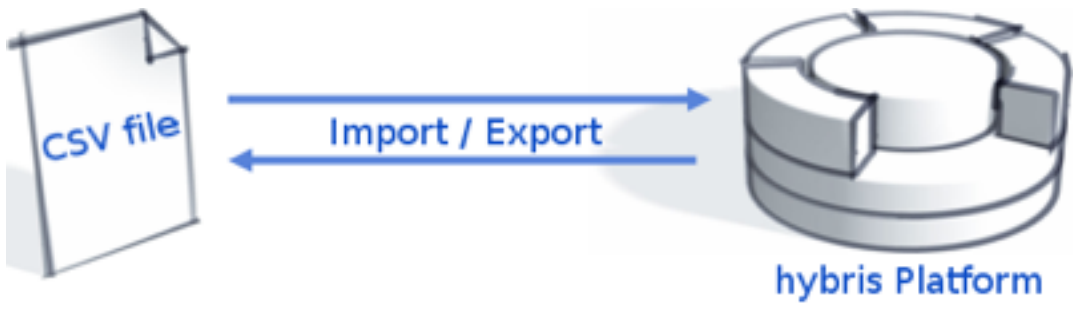
Overview

Run
Preparation Script
for
Exercise 4

Overview
Syntax and Examples
Invoking
Scripting
Distributed ImpEx

ImpEx – Overview

- ImpEx is an out-of-the-box import / export framework
- It’s an interface between CSV files and the SAP Commerce Suite’s Type System
  - you can “import” instances of types from CSV files
  - you can “export” instances of types into CSV files
- You can create, update, remove, and export items



ImpEx – Typical fields of use

- In live operation:
  - to import customer data into a production system
  - to synchronize data with other systems, such as an ERP or LDAP
  - to create backups
  - to update data at runtime
  - can be run from CronJobs
- In migrations:
  - to migrate data from one SAP Commerce installation to another
- In development:
  - to import sample data (e.g. on system initialization)
  - to import test data into testing system

ImpEx – Features

- ImpEx abstracts from database
  - **No table information** (deployment)
  - **No foreign keys** (use “business keys,” which we will discuss in a moment)
- ImpEx simplifies imports
  - The order of the imported data **is irrelevant**! (Failed lines are retried)
  - Validation constraints may be disabled

```
impex.legacy.mode=true
```

- ImpEx concerns
  - no transactional imports
  - Performance – use multithreaded imports:

```
impex.import.workers=4
```

- Note: ImpEx does not provide XML import out-of-the-box

Order of imported data has no impact on result

However, the order of imported data does impact performance

Syntax and Examples

Overview
Syntax and Examples
Invoking
Scripting
Distributed ImpEx

Syntax Basics

- Header syntax:

```
Operation     itemType; attributes(refAttr)[modifiers];...

INSERT        Product;  code; name[lang=en];
UPDATE        Car;      code[unique=true]; name[lang=en];
INSERT_UPDATECustomer; customerID[unique=true]; groups(uid);
REMOVE        Media;    code[unique=true];
```

- Data row syntax:

```
;attr1value;attr2value;...

;CanonPS430;PowerShot 430;
;Peugeot 403;Columbo's Car;
;FrankColumbo;customergroup;
;P403Pic;
```

Before We Dive In…

In the Data Modeling Chapter, you learned that:

- Each database entity (item) in SAP Commerce has a surrogate (system-generated) key called the PK (for Primary Key)
  - The PK is used when an entity (item) refers to another entity. For example, if the Customer entity needs a reference to an Address entity, the PK of the Address is stored in the customer table.
- Data imported from / exported to other systems will have a business (natural) key
  - The business key can be a single data field
    - In this case, this data field is unique
  - When the business key is comprised of multiple fields, we call it a **composite** business key
    - In this instance, it is the combination of all fields in the key that is unique

Basic syntax example

```
INSERT_UPDATE Promotion; code[unique=true]; name[lang=en]; name[lang=fr]; country(isocode)
;Maranello3; Antarctica Ferrari launch; Lancement Ferrari en Antartique; AQ
;DeLorean_CN; De Lorean China Campaign; Campagne De Lorean en Chine; CN
```

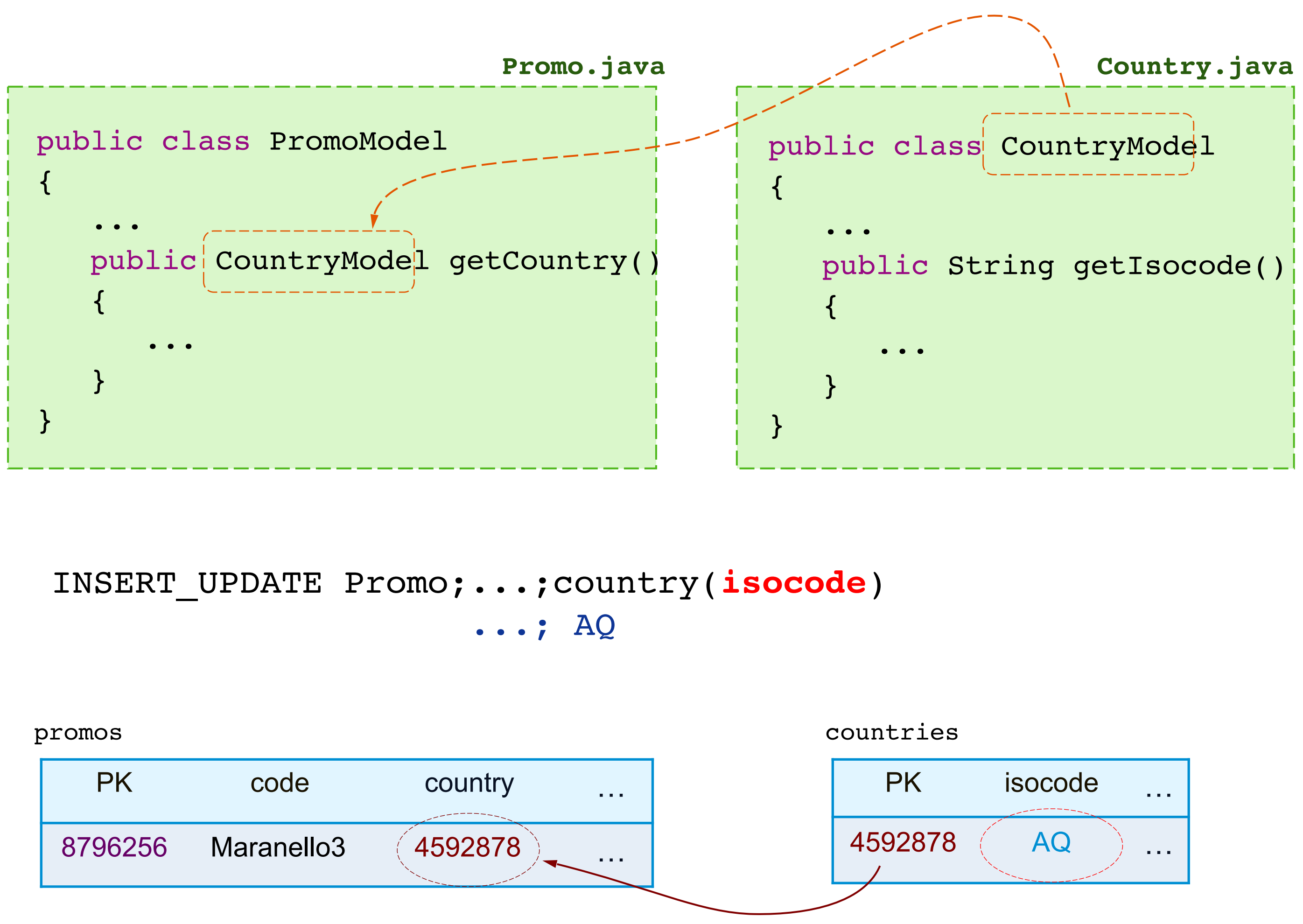
- **Localized name references a lookup table using keys en and fr**
- **Country is another entity in its own table, referenced using its isocode property**

Key points:

- The `code[unique=true]` is so called "key attribute" or "business key".
  ImpEx will search for product with code *`Maranello3`* before triggering import. If more than one column is marked as unique, then ImpEx will consider the business key to be multi-column.
- The `[lang=en]` qualifier indicates the language of the value provided. This is only valid for localized attributes, and many languages can be loaded on the same line.
- The header field `country(isocode)` is a reference to another item using its code ("business key"). In this example, the *country* property of Promo item *`Maranello3`* is a reference to another SAP Commerce item, whose *isocode* attribute has the value *`AQ`*. Here, SAP Commerce will look that item up, and use its PK in the Promo table.

Why We Need to Specify the Business Key in Impex

Direct reference to Country in Object Model



Promo.java

```
public class PromoModel
{
   ...
   public CountryModel getCountry()
   {
      ...
   }
}
```

Country.java

```
public class CountryModel
{
   ...
   public String getIsocode()
   {
      ...
   }
}
```

```
INSERT_UPDATE Promo;...;country(isocode)
                   ...; AQ
```

promos

PK	code	country	…
8796256	Maranello3	4592878	…

countries

PK	isocode	…
4592878	AQ	…

- Object reference in Java code
- Foreign key in DB
- No foreign keys allowed in ImpEx
- ? How do we represent this reference in ImpEx?
- Use its business key!

ImpEx Syntax Elements

- Macros
  - Allows aliases to stand in for frequently used statements
- BeanShell, Groovy, and Javascript scripting
  - Allows script to be added to a CSV file.
  - Predefined `hooks beforeEach, afterEach, getLastImportedItem()` etc.
- Translators
  - Implement custom ImpEx logic e.g. to import *medias* (binary files).
- Inclusion of data
  - Allows you to split your ImpEx operations over several files.
- Collections and HashMaps:
  - Allows you to use these types as attributes
- Different validation modes for export
  - E.g. the mode “Strict (Re)Import” ensures that the export is re-importable

Catalog example

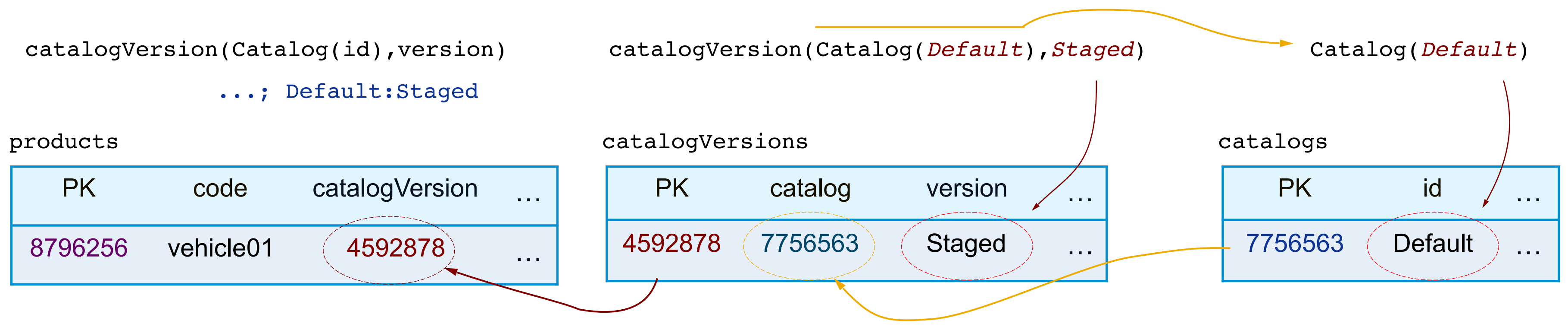
```
$catalogVersion=catalogVersion(catalog(id),version)[unique=true]

INSERT_UPDATE Car; code[unique=true]; name[lang=en]; unit(code); $catalogVersion

;DB5;Aston Martin DB5; pieces; Default:Staged
;ES1;Lotus Esprit S1; pieces; Default:Online
```

- This example uses a macro, which is substituted verbatim.
- A catalog-aware item like product uses a composite key, since more than one instance can exist (in different catalog versions).
  - The composite key is denoted by having two header fields listed as unique (code and catalogVersion).
  - The catalog version itself uses a composite business key — so we need to reference it using a pair of values.
    - The value pair is separated by commas in the header, and a colon (:) in the data line.

Catalog reference details

```
$catalogVersion=catalogVersion(catalog(id),version)[unique=true]
INSERT_UPDATE Car; code[unique=true]; name[lang=en]; unit(code); $catalogVersion
;DB5;Aston Martin DB5; pieces; Default:Staged
;ES1;Lotus Esprit LS1; pieces; Default:Online
```

- References
  - The product item references a catalogVersion item, which is identified using two keys: a catalog reference and a *version* string. The catalog reference, in turn, is identified by an *id* string.



catalogVersion(Catalog(id),version)
...; Default:Staged
catalogVersion(Catalog(*Default*),*Staged*)
Catalog(*Default*)
products
catalogVersions
catalogs

products

PK	code	catalogVersion	…
8796256	vehicle01	4592878	…

catalogVersions

PK	catalog	version	…
4592878	7756563	Staged	…

catalogs

PK	id	…
7756563	Default	…

Document Id

- Normally, all business keys must be supplied when cross-referencing

```
$catalogVersion=catalogVersion(catalog(id),version)[unique=true]
INSERT_UPDATE Car; code[unique=true]; name[lang=en]; $catalogVersion
;DB5;Aston Martin DB5; Default:Staged
;ES1;Lotus Esprit LS1; Default:Online

INSERT_UPDATE Employee; uid[unique=true]; car(code, $catalogVersion)
 ;FrankColumbo; DB5:Default:Staged
```

- Use Document ID to simplify cross-reference imports

```
$catalogVersion=catalogVersion(catalog(id),version)[unique=true]
INSERT_UPDATE Car; code[unique=true]; name[lang=en]; $catalogVersion;&CarRef
;DB5;Aston Martin DB5; Default:Staged; db5
;ES1;Lotus Esprit LS1; Default:Online; es1

INSERT_UPDATE Employee; uid[unique=true]; car(&CarRef)
 ;FrankColumbo; db5
```

Here we define the references

Here we use (point to) references defined above

This is a simple example, but in BIG ImpEx files, using Document IDs simplifies things a lot and reduces errors!

Using Macros and Defaults

```
$prodCat=myCatalog
$version=Staged
INSERT Category;code;catalogVersion(catalog(id),version)
;cars;$prodCat:$version
;convertibles;$prodCat:$version
```

```
$catVersion=catalogVersion(catalog(id[default=$prodCat]),version[default=$version])
INSERT Category;code;$catVersion
;cars;
;cars;myCatalog
;cars;myCatalog:$version
;cars;:Staged
```

Every line here is equivalent

- Notes
  - macros can be used in both header and data rows
  - use default values to simplify data rows

Maps and Collections

- When importing maps, define delimiter (default is ; and -> escape-out with "")

```
UPDATE Employee;uid[unique=true];preferences
               ;FrankColumbo    ;"drink->whiskey;game->poker;colour->beige"
```

- Redefine map-delimiter and key-value delimiter if you like

```
UPDATE Employee;uid[unique=true];relatives[map-delimiter=|][key2value-delimiter=>>]
               ;FrankColumbo    ;wife>>Mrs. Columbo|sister>>Rose|brother>>Fred
```

- For collections, default mode is 'replace'
  - use ‘append’ mode to avoid overriding existing references

```
INSERT_UPDATE Employee;uid[unique=true];groups(uid)[mode=append]
                      ;FrankColumbo    ;approvers,dummygroup,reviewers
```

  - use ‘remove’ mode to eliminate existing references

```
INSERT_UPDATE Employee;uid[unique=true];groups(uid)[mode=remove]
                      ;FrankColumbo    ;approvers
```

Only **dummygroup** and **reviewers** remain

Advanced Qualifiers

- Use ‘translators’ for custom interpretation of imported values

```
INSERT_UPDATE Employee;uid[unique=true];@password[translator=de.hybris.….PasswordTranslator]
                      ;FrankColumbo    ;aVeryStrongPassword;

INSERT_UPDATE Media;code[unique=true];@media[translator=de.hybris.….MediaDataTranslator]
                   ;media01          ;/path/to/my/picture.jpg;
```

- Batch update

```
UPDATE Product [batchmode=true];itemType(code)[unique=true];approvalStatus(code)
                               ;Product                    ;approved
```

All items matching the key itemType(**code**) **== Product** are set to **approved**

- Date Format

```
UPDATE Rental;rentalId[unique=true];startDate[dateformat='yyyy.MM.dd'];endDate[dateformat='MM/dd/yyyy’]
             ;101                  ;2005.01.31                        ;03/15/2005
```

ImpEx Script For Export

- Specify the target file:

```
"#%beanshell% impex.setTargetFile( ""Product.csv"" );"
```

- Specify the attributes to be exported usingg an ImpEx header:

```
INSERT_UPDATE Product;code[unique=true];description[lang=en];name[lang=fr];unit(code)
```

  - You can use the same header for re-import.
  - Consider using the Script Generator feature of the Backoffice

- Full export

```
"#%beanshell% impex.exportItems( ""Product"" , false );"
```

- Selective export

```
 "#%beanshell% impex.exportItemsFlexibleSearch(
    ""select {pk} from {Product} where {code} like '%happy%'"");"
```

Invoking

Overview
Syntax and Examples
Invoking
Scripting
Distributed ImpEx

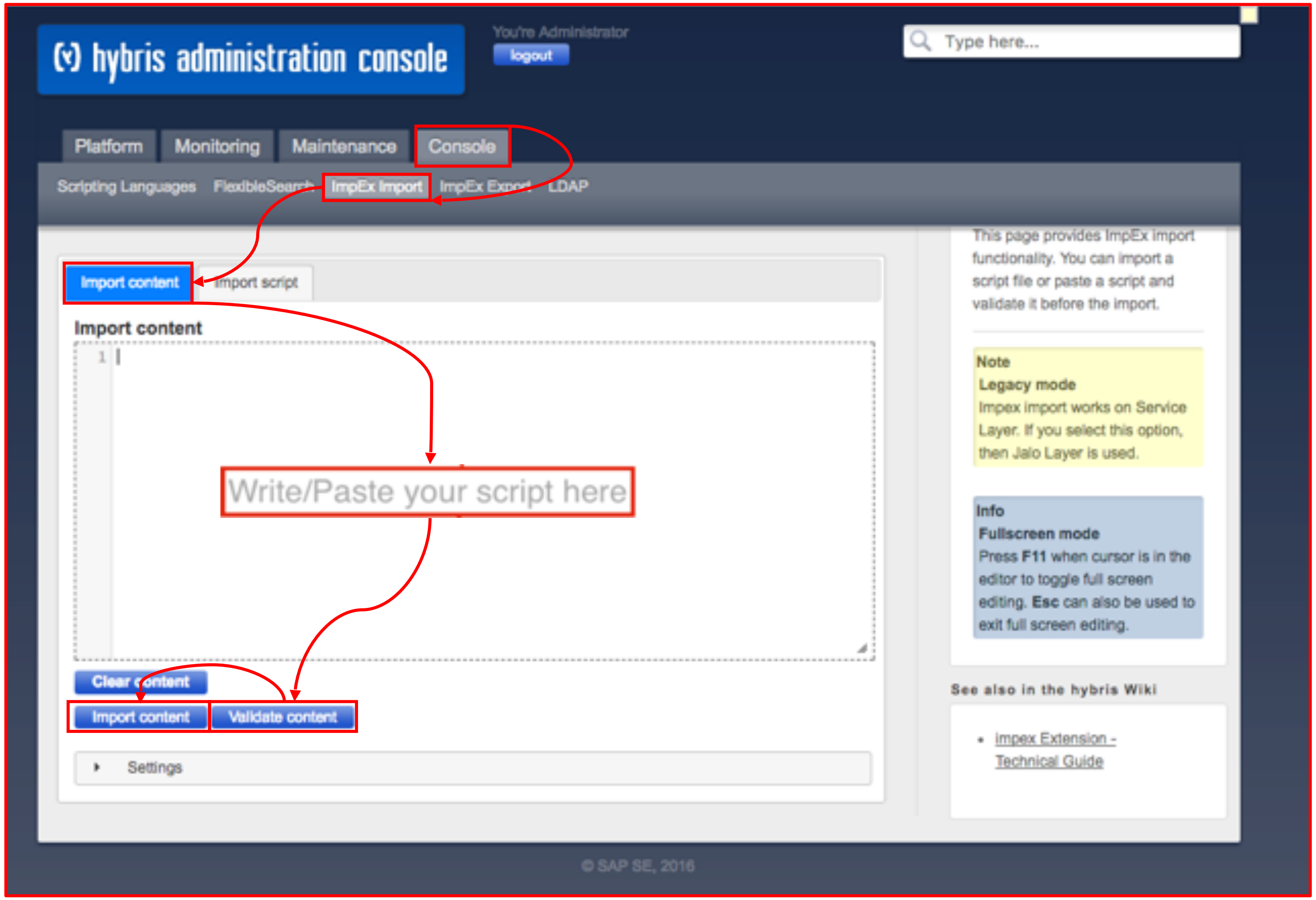
Where Can You Launch an Import?

- In the Hybris Administration Console
  - Test area for ImpEx scripts
  - Multiple files cannot be imported by a single ImpEx script
  - No external data is allowable
  - Limited configuration possibilities
- In the Backoffice
  - Create an `ImpExImportCronJob`
- Using the API
  - You can use the `ImportService`
- Using the Command Line
  - `ant importImpex -Dresource=/full/path/to/import.impex`

Where Can You Launch an Export?

- In the Hybris Administration Console
  - Test area for ImpEx scripts
- In the Backoffice
  - Select search results and export them using the context menu
  - Create an `ImpExExportCronJob`.
- Using the API
  - Use the `ExportService`
  - Create an `ImpExExportCronJob`

ImpEx Import in the HAC



This page provides ImpEx import functionality. You can import a script file or paste a script and validate it before the import.

Note
Legacy mode
Impex import works on Service Layer. If you select this option, then Jalo Layer is used.

Info
Fullscreen mode
Press **F11** when cursor is in the editor to toggle full screen editing. **Esc** can also be used to exit full screen editing.

See also in the hybris Wiki

- impex Extension - Technical Guide

© SAP SE, 2016

Scripting

Overview
Syntax and Examples
Invoking
Scripting
Distributed ImpEx

Scripting in ImpEx

- Prior to v5.2, ImpEx supported only beanshell scripting, so no need to specify language
  - Currently, script language should be specified with `%groovy%`, `%javascript%`, or `%beanshell%`
  - However, the old behavior (beanshell-only) is still the default

```
INSERT_UPDATE Currency;isocode[unique=true];conversion;digits;
#%groovy% aftereach: impex.info "$currentLineNumber ${line[1]}"
;GBP;1;2;
;EUR;1;2;
```

- Each line of script has its own context, meaning there is no common context shared by different lines of script code
- Scripts will only run if `"Enable code execution"` is selected
- To use Groovy or Javascript in ImpEx Import, set the global configuration property:
  - `impex.legacy.scripting=false`
- For HAC ImpEx Import Console: these can be set in the “settings” section.
- For a specific ImpEx Import Cron Job: (Backoffice: Administration tab)

Distributed ImpEx

Overview
Syntax and Examples
Invoking
Scripting
Distributed ImpEx

Overview

- Splits up ImpEx import work into separate batches, distributed across the cluster, which aims to handle scale large import tasks more efficiently
- Leverages the existing ImpEx framework to parse and analyze input and dump unresolved lines, and the TaskEngine to process single batches of data
- Works in 3 phases
  - Prepare and split phase: ImpEx file is read and split into batches
  - Single task execution phase: Task engine executes each batch individually, but in parallel
  - Finish phase: Clean up work

Regular ImpEx vs. Distributed ImpEx

Capability	Regular Impex	Distributed Impex
Servers utilized per import	single	whole cluster (can be limited to specific nodes or node groups)
Import data processed at once	one line	multiple lines (configurable as batch size)
Database transactions created	multiple transactions can be triggered for each line	one transaction for each batch
JDBC batch mode for similar data	no	yes
Which persistence layer can be used ?	Jalo, Model	Model
Triggered lookup queries	for each line	single query for all lines of a batch
Circular (missing) references resolved	yes (preserving unresolved lines and processing them in multiple round)	yes (preserving unresolved batches and processing them in multiple rounds)
Import can be aborted	no	yes (using the API – a UI is planned)

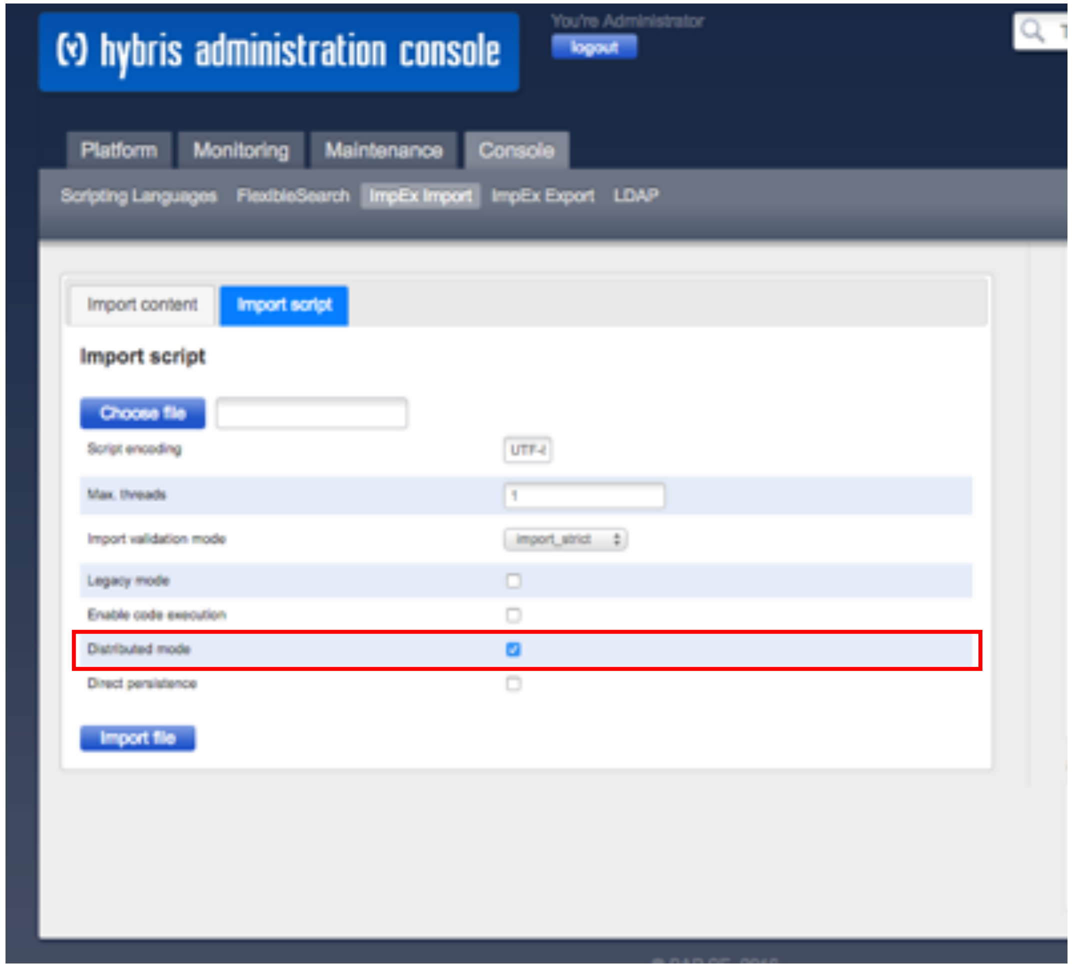
API enablement

- Enabling data import in the distributed mode programmatically works similarly as in classical ImpEx.
- For enabling it, `ImportConfig` API is used

```
final ImportConfig config = new ImportConfig();
config.setDistributedImpexEnabled(true);
```

Execute Distributed ImpEx in UI

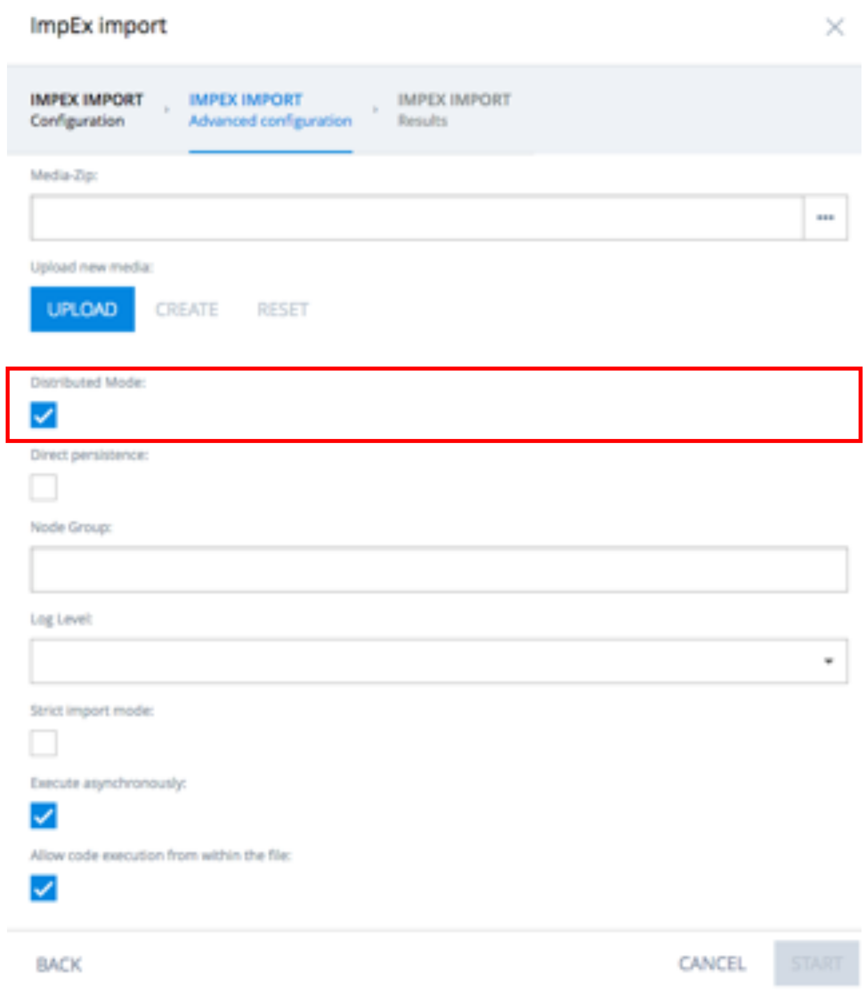
From HAC



Import script

Script encoding	UTF-8
Max. threads	1
Import validation mode	import_strict
Legacy mode	☐
Enable code execution	☐
Distributed mode	✓
Direct persistence	☐

From Backofice



Media-Zip:

Upload new media:

Distributed Mode: ✓

Direct persistence: ☐

Node Group:

Log Level:

Strict import mode: ☐

Execute asynchronously: ✓

Allow code execution from within the file: ✓

ImpEx is the principal tool for importing data into or exporting data from SAP Commerce

A **Business key** will be used to reference another data item

Defaults and Macros can be used to simplify an ImpEx script

A Translator is used to process special attributes and certain translators are available OOTB

- e.g. use the `ClassificationAttributeTranslator` to import values for classification features.

You can import / export data by using the HAC, an activation cronjob in Backoffice, or by invoking importService / exportService directly from your code

Distributed ImpEx was introduced to improve importing performance as of SAP Commerce 6.0

Exercise

ImpEx

Thank you.