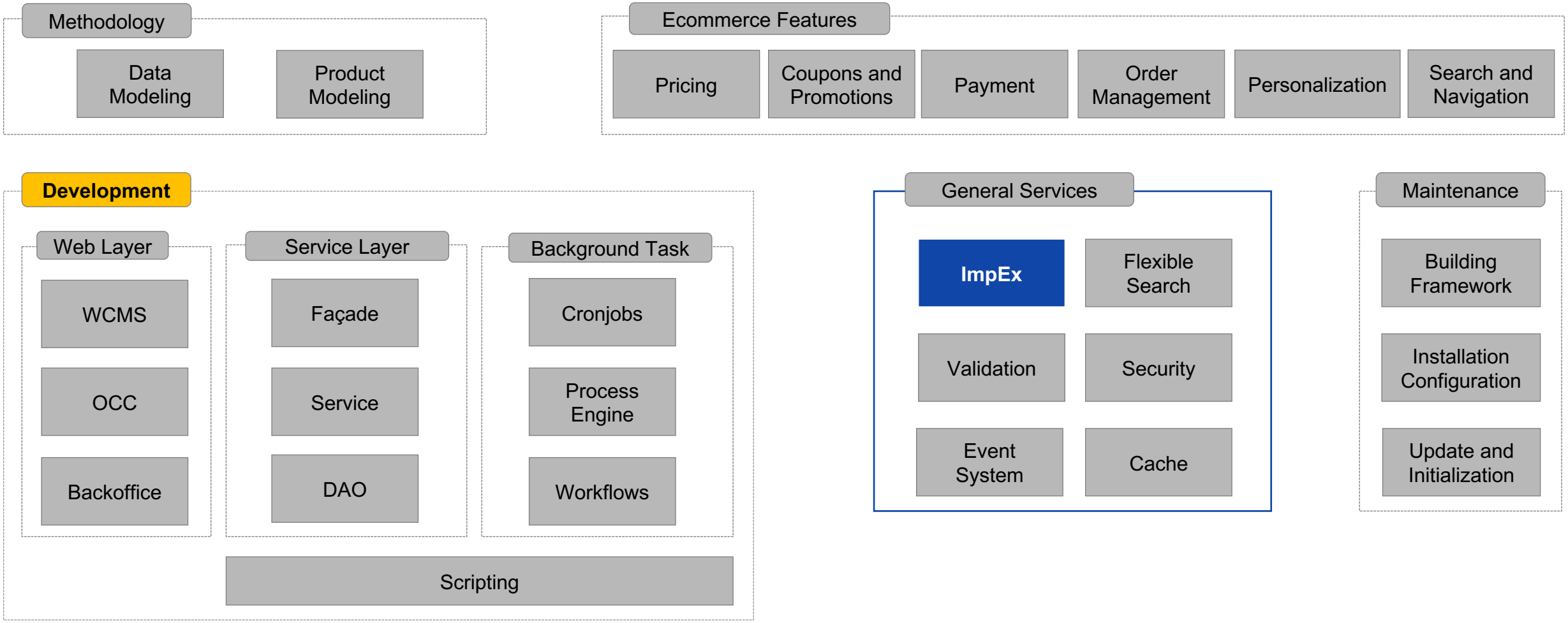
SAP Customer Experience

Impex

What we will cover in this topic



We will learn about:

- Overview Of ImpEx
- Syntax and Examples
- Invoking ImpEx
- Distributed ImpEx

The Context…

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

Overview

ImpEx – Overview

- ImpEx is an out-of-the-box import / export framework
- It's an interface between CSV files and the SAP Commerce Domain
  - 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 areas 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 core data or sample data (e.g., for system initialization)
  - to import test data into a testing system

ImpEx – Features

- ImpEx abstracts from database metadata (only refers to Commerce types and their attributes)
  - **No knowledge of underlying table or column names** (deployment)
  - **No foreign keys** (use “business keys,” which we will discuss in a moment)
- ImpEx simplifies imports
  - The order of the imported data **is ultimately irrelevant**! (Failed lines are retried)
  - Validation constraints may be disabled, to reduce overhead

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

- ImpEx concerns
  - no transactional imports (data lines cannot be grouped into transactions)
    (i.e., each individual data line commits independently, on success)
  - 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

Syntax Basics

- Header syntax:

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

INSERT         Product;   code; name[lang=en];
UPDATE         Car;       code[unique=true]; name[lang=en];
INSERT_UPDATE  Customer;  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;
```

? **What is the difference between `INSERT` and `INSERT_UPDATE`?**

- **`INSERT` creates a new entity**
- **`INSERT_UPDATE` uses an existing entity or creates a new one if one doesn't exist**

Before We Dive In…

In the Data Modeling Chapter, you learned that:

- Each persisted entity (item) in SAP Commerce has a system-generated key called the PK (for Persistence Key)
  - The PK is the identifier that the **persistence** layer uses to uniquely identify it across *all* types, e.g., in the Cache.
    It is used when any item holds a ***reference*** to any other item. For example, if an attribute of type Customer "points" to an Address item, the PK of the Address item is what is stored in that attribute.
  - A PK value can only be assigned by the Persistence Layer
  - Once a PK is assigned, it does not change, and it is not reused
    (...at least, not until the commerce cluster is initialized, resetting the PK generator)
- Data imported from / exported to other systems will have a business (natural, user-assigned) 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 (for a given type, within a catalog version)

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
```

- **The localized: *name* attribute requires *lang* keys en and fr to specify map's target elements**
- **The *country* attribute requires a *reference* (PK) to an item of another type (Country). Each target item is looked-up via Country's business key (isocode attribute), whose values are AQ and CN.**

Key points:

- The `code[unique=true]` marks the header attribute(s) that make-up the "key" attribute or "business key". ImpEx will search for product with code ***Maranello3*** before triggering import, to determine the target item, if any. (e.g., for UPDATE, there ***must*** be a unique, matching target item, whereas for INSERT there **must *not*** already be one). If more than one column is marked as unique, then ImpEx will consider the business key to be multi-column.
- A **localized:** attribute is really just a key-value *map*. A `[lang=??]` qualifier must be used to specify the target map *element* for the value provided. As this header example shows, multiple languages' data can be provided in the same data line, as long as each header attribute is sufficiently specified to make sense for the data provided.
- 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 Usually Specify the Business Key in ImpEx (instead of PK)

Direct reference to Country in Object Model

PromotionModel.java

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

CountryModel.java

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

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

promotions

PK	code	country	…
8796256	Maranello3	4592878	…

countries

PK	isocode	…
4592878	AQ	…

- In the Java domain:
  Java Object reference
- In the Commerce domain:
  Reference attrib = target item's PK
- In the DB domain:
  Foreign key = target row's PK
- PKs are often unknowable at the time the ImpEx is written (e.g., when used for *initialize*)
- ? How can we express reference in ImpEx without knowing the PKs?
- Use business keys!

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.

Scripting is covered in the live session series: "SAP Commerce Cloud – Additional Technical Essentials" details cf.

- 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

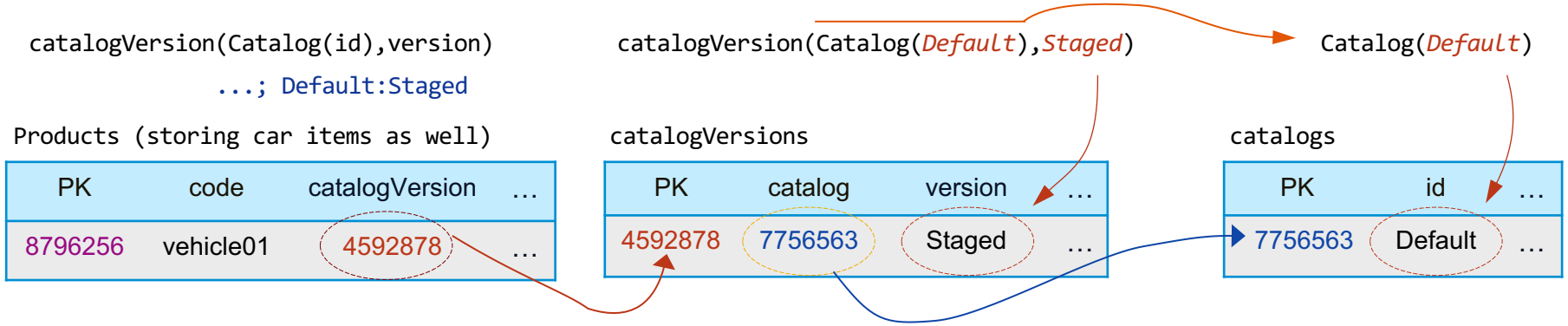
```
$catalogVersion=catalogVersion(catalog(id),version)[unique=true]
INSERT_UPDATE Car; code[unique=true]; name[lang=en]; unit(code); $catalogVersion
;DB5;Aston Martin DB5; pieces; Default:Staged
;ES1;Lotus Esprit S1; pieces; Default:Online
```

- This example uses a macro, which is substituted verbatim.
- A catalog-aware item like Car (subtype of Product) needs a composite business key to uniquely identify it, since multiple Car instances with the same code will exist across multiple 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 car 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.



Using Macros and Defaults

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

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

Every line here will produce the same end-result

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

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 *store* each newly created reference (PK) into the `CarRef` Map under their assigned keys: `db5` and `es1`

Here we *retrieve* the stored PKs (note the "PK lookup" parentheses) from the CarRef Map via their storage keys

In BIG ImpEx files, this greatly improves performance. The ImpEx processor stores each newly created PK in memory, thus eliminating a DB search. `CarRef` is the name of a Java *Map* whose *keys* are `db5` and `es1`, and *values* are the stored PKs.

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
```

Approve all Product items whose itemType property references ItemType whose code == 'Product'

- Date Format

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

Importing Classification Attributes

- A translator is necessary to import classification attributes
  - SAP Commerce provides the ClassificationAttributeTranslator
  - The translator needs the classification system and version
- For example, importing the classification attribute *trait* :

```
UPDATE Product;code[unique=true];
               @trait[system='ClassificationSystem',version='1.0',
                      translator=de.hybris.platform.catalog.jalo.classification.
                                 impex.ClassificationAttributeTranslator];
```

- SAP Commerce normally uses macros for this:

```
$sysName=ClassificationSystem
$sysVer=1.0
$translator=de.hybris...ClassificationAttributeTranslator
$clAttrModifiers=system=$sysName,version=$sysVer,translator=$translator

UPDATE Car;code[unique=true];@ads[$clAttrModifiers];@terms[$clAttrModifiers]
          ;car01            ;Coca-Cola,Pennzoil    ;5y loan
```

ads is a multi-value attribute, thus comma-separated

Where Can You Launch an Import?

- In the SAP Commerce Cloud Administration Console (HAC)
  - Test area for ImpEx scripts
  - ImpEx files can only be imported (executed) one-at-a-time
  - No external data is allowable
  - Limited configuration possibilities

- In the Backoffice (Create an `ImpExImportCronJob`)
  - System → Tools → Import
  - System → Background Processes → CronJobs (+) ImpEx Import CronJob

- Using the API
  - You can use the `ImportService, ImpExImportCronJob, Importer, etc.`

    More info: https://help.sap.com/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/aa417173fe4a4ba5a473c93eb730a417/2fb5a2a780c94325b4a48ff62b36ab23.html

- Using the Command Line
  - `ant importImpex -Dresource=/full/path/to/import.impex`

- Via (Cloud) Hot Folders
  - **covered in the live session series: "SAP Commerce Cloud – Additional Technical Essentials" details cf.**

Where Can You Launch an Export?

- In the SAP Commerce Cloud Administration **ImpEx Export** Console
  - Interactive editor for ImpEx scripts

- In the Backoffice Adminstration Cockpit (Create an `ImpExExportCronJob`)
  - System → Tools → Export
  - System → Background Processes → CronJobs (+) ImpEx Export CronJob

- Using the API
  - Use the `ExportService`
  - Create an `ImpExExportCronJob`
  - Check for further details: `Export API on //help.sap.com`

ImpEx Script For Export

- Specify the target file:

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

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

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

  - The system will generate an ImpEx file with the same header (but with data lines) 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%'"");"
```

- Hint: if you don't want to write the script yourself?
  - The **ScriptGenerator** from Backoffice Admin Cockpit can generate a script to export *all* items (all types).
    Backoffice Admin Cockpit → System → Tools → Script Generator

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
- More information can be found: [ImpEx Distributed Mode on //help.sap.com]

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)

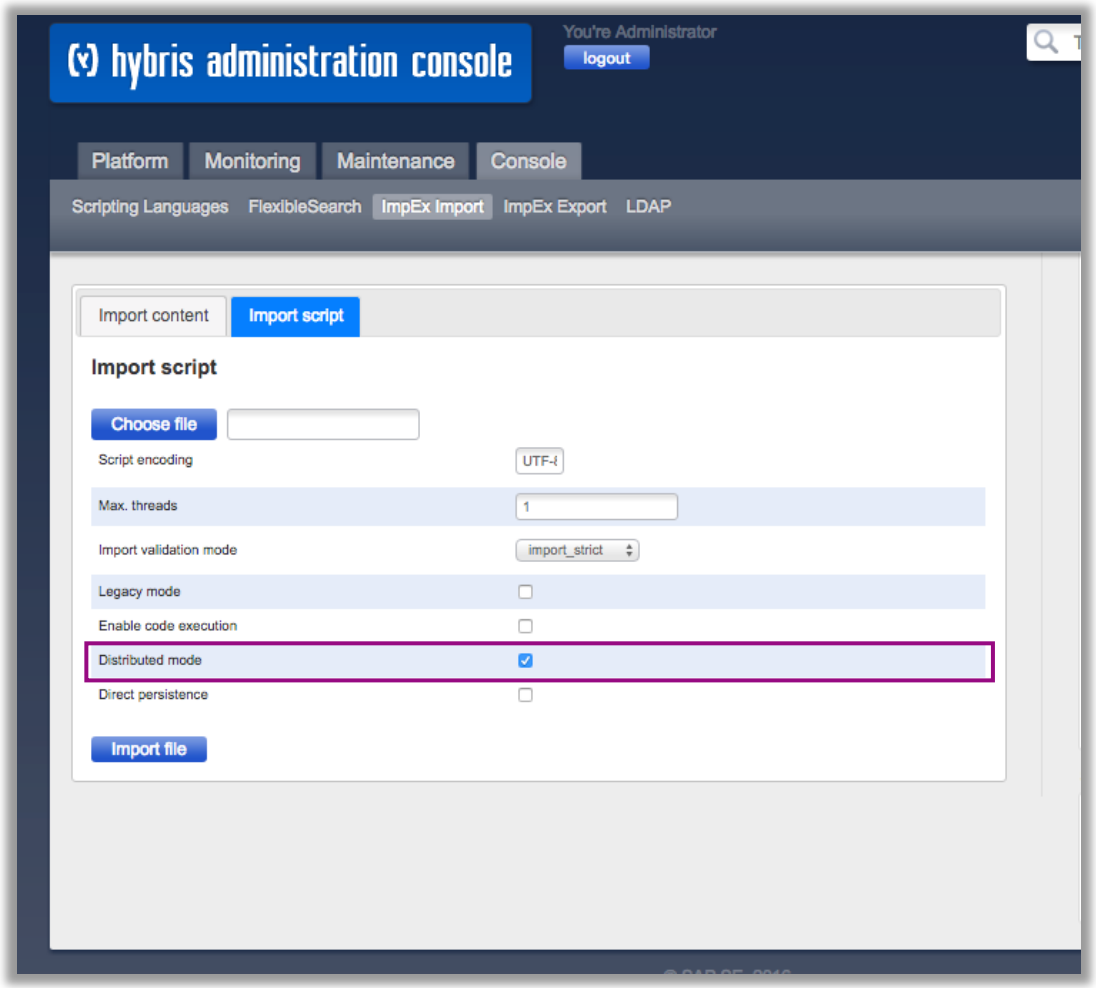
API enablement

- Enabling data import in the distributed mode programmatically works similarly as in classical ImpEx.
- To enable it, use the **`ImportConfig`** API

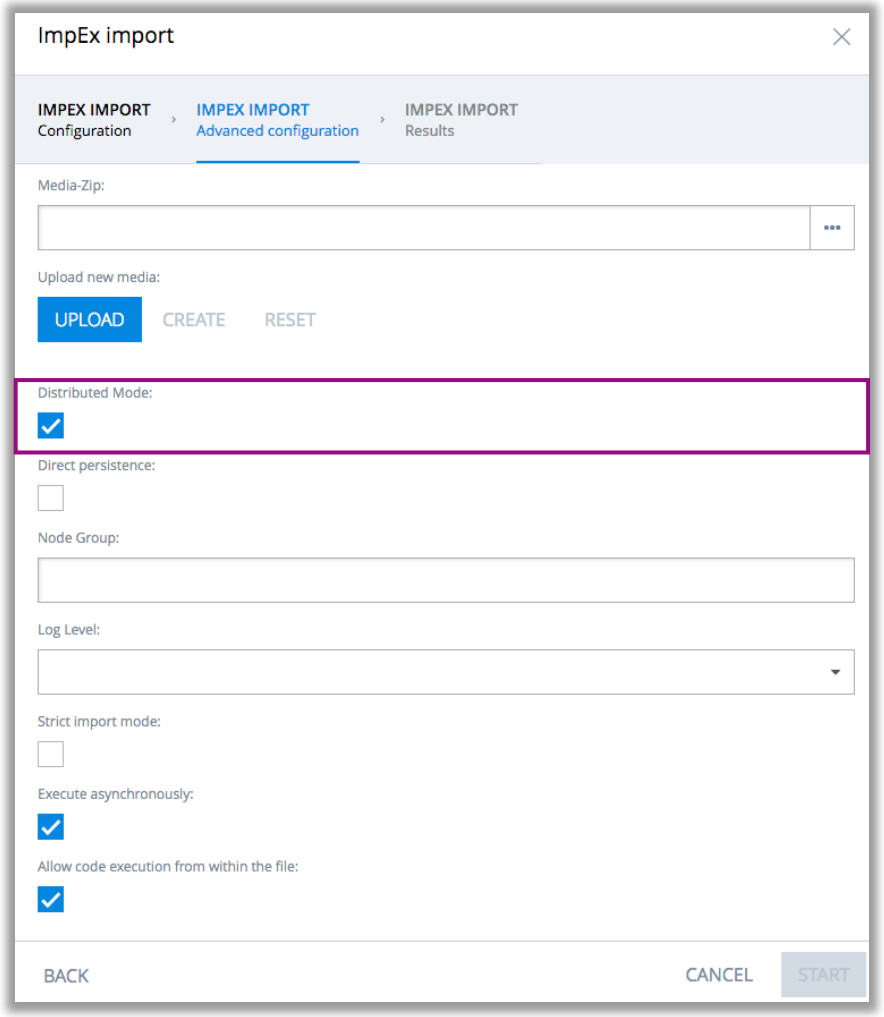
```
/*
  assuming we have an ImpExResource object that points to an import
  file on classpath
*/
final ImpExResource importFile;
final ImportConfig config = new ImportConfig();
config.setDistributedImpexEnabled(true);
config.setScript(importFile);

// perform import
final ImportResult importResult = importService.importData(config);
```

Execute Distributed ImpEx in UI

From HAC



From Backofice



Key Points

1. ImpEx is the principal tool for importing data into or exporting data from SAP Commerce.
2. A **Business key** will be used to reference another data item.
3. Defaults and Macros can be used to simplify an ImpEx script.
4. 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.
5. You can import / export data by using the HAC, an import/export cronjob in Backoffice, or by invoking importService / exportService directly from your code.
6. Distributed ImpEx was introduced to improve importing performance.

Impex Exercise

Thank you.