TechD·vision

Documentation Pacemaker v1.3.0

Welcome

PACEMAKER

CAUTION	As of version 3.8.0, the structure of the setup has changed considerably, and the previous configuration files are no longer be used. To avoid complex adjustments of the configuration, version 3.8.0 merged the configuration for all entities into one but dedicated overwriting of individual settings is now possible, e.g. for the log level.

Pacemaker Community Edition (CE) - provides components to build import solutions for Magento 2

Unlike other approaches, the framework is not built as a **Magento 2** extension. There are also no dependencies on other frameworks like Symfony, Laravel, or others.

Independent components are provided, which can be linked together as needed by adding them as component dependencies. Our fully functional CLI implementation is a good start to show you how the **Pacemaker Community Edition (CE)** Version can be used and customized to your needs.

You can find more information in our **Pacemaker Community Edition (CE)** Repository

- **Pacemaker Community Edition (CE)** Console Tool.

General system requirements and essentals

PHP Version

Compatible to PHP Version   >= 7.4

For further informations regarding **Magento** we recommend to check out the **Magento** specific system requirements and essentals.

TIP	To successfully run **Pacemaker** with **Magento** please check the requirements and essentials of the **Pacemaker Community Edition (CE)** , **Pacemaker Professional Edition (PE)** and **Pacemaker Enterprise Edition (EE)**

Community

Getting started

CAUTION	As of version 3.8.0, the structure of the setup has changed considerably, and the previous configuration files are no longer be used.
	To avoid complex adjustments of the configuration, version 3.8.0 merged the configuration for all entities into one but dedicated overwriting of individual settings is now possible, e.g. for the log level.

To install the **Magnento 2 Import Framework**, the Composer is necessary.

The framework itself is a set of components that provides import functionality for **Magento 2**. This repository, based on Symfony Console, uses the package Pacemaker Community Edition (CE) and provides a command-line tool with import functionality for **Magento 2** standard CSV files.

Before you start, keep in mind, that we've prepared a complete set of sample data. The sample data containing

- *attribute-sets*
- *attributes*
- *categories*
- *products*

are based on the original **Magento 2** sample data.

Please check out our sample data repository when you are going to test **Pacemaker Community Edition (CE)** or to find out, how the CSV files have to be structured.

NOTE	For **Magento** versions 2.2.x and 2.3.x, up from the **Pacemaker Community Edition (CE)** version, 3.6.0 can be used.

youtube video

Helpful notes

Reference resources

Magento resources

- Extension dev guide
- User guide - product attributes
- User guide - complex data
- User guide - customer attributes
- v2.3 Install guide
- start **MQ** consumers via cron

- Supervisor
- Supervisor configuration documentation
- specific system requirements and essentals
- **ERP/OMS**

Pacemaker resources

- Import Framework
- Import cli simple
- **Import Framework** - Magento 2 Import Framework
- Import cli simple
- **Import Framework** - Sample Data
- **Import Framework** - Product Import example CSV
- **Import Framework** - Product Import
- Shortcuts
- Media storage solution

Other resources

- Mysql reference 8.0
- Mageplaza - how to upload product videos in **Magento 2**
- Informit - pipelines
- Wikipedia - **SPMD**
- Jenkins - pipelines
- Gitlab - pipelines
- innodb_flush_log_at_trx_commit

PHP resources

- str_pad() function description in PHP manual

Requirements

PHP Version

Compatible to PHP Version   >= 7.4

Installation

CAUTION   As of version 3.8.0, the structure of the setup has changed considerably, and the previous configuration files are no longer be used.

> To avoid complex adjustments of the configuration, version 3.8.0 merged the configuration for all entities into one but dedicated overwriting of individual settings is now possible, e.g. for the log level.

Depending on the requirements, different installation options are available.

TIP | Install Guide for **Magento 2 CE/EE** + **Pacemaker Community Edition (CE)** + **example data** step by step

Install as Composer Project

To install the package as a new project, assuming the composer is available, open a console and enter

```
composer create-project techdivision/import-cli-simple --no-dev
```

It will clone the repository from the internal Gitlab and install **Pacemaker Community Edition (CE)**, that's all.

Install as Composer Library

The second option, and in most **Magento 2** projects the preferred way, will be the installation as a Composer library. For example, if you want to deliver it with your **Magento 2** project, please add

```
{
  "require": {
    "techdivision/import-cli-simple" : "3.8.x"
  }
}
```

to your **Magento 2** composer.json file. Then run

```
composer update
```

from your **Magento 2** root directory, and you're all set up.

Use as PHAR

The last, but for sure not the worst installation option, is to download the latest PHAR from our Github release page and make it executable, e.g. with wget

```
wget https://github.com/techdivision/import-cli-simple/releases/download/3.8.12/import-cli-
simple.phar \
  && sudo chmod +x import-cli-simple.phar
```

To install the PHAR in your actual **Magento 2** installation, move it to <MAGENTO-ROOT>/bin/import-cli-simple.phar or, to install it globally, to /usr/bin/import-cli-simple.phar. Now you ready to use it.

Add Missing Indexes

As the **Pacemaker Community Edition (CE)** functionality differs from the **Magento 2** standard, for performance reasons, it is necessary to add some essential indexes manually. To do that, open a MySQL command line, connect to your MySQL server

instance and enter the following SQL statement

```
ALTER TABLE `eav_attribute_option_value` ADD INDEX `EAV_ATTRIBUTE_OPTION_VALUE_VALUE`
(`value` ASC); \
  ALTER TABLE `catalog_product_entity_int` ADD INDEX `CATALOG_PRODUCT_ENTITY_INT_VALUE`
(`value` ASC); \
  ALTER TABLE `catalog_product_entity_varchar` ADD INDEX
`CATALOG_PRODUCT_ENTITY_VARCHAR_VALUE` (`value` ASC); \
  ALTER TABLE `catalog_product_entity_decimal` ADD INDEX
`CATALOG_PRODUCT_ENTITY_DECIMAL_VALUE` (`value` ASC); \
  ALTER TABLE `catalog_product_entity_datetime` ADD INDEX
`CATALOG_PRODUCT_ENTITY_DATETIME_VALUE` (`value` ASC); \
  ALTER TABLE `url_rewrite` ADD INDEX `URL_REWRITE_ENTITY_ID` (`entity_id` ASC); \
  ALTER TABLE `url_rewrite` ADD INDEX `URL_REWRIRE_ENTITY_TYPE_ENTITY_ID` (`entity_id` ASC,
`entity_type` ASC); \
  ALTER TABLE `catalog_product_entity_media_gallery` ADD INDEX
`CATALOG_PRODUCT_ENTITY_MEDIA_GALLERY_VALUE` (`value`);
```

When using the Magic 360 component, additional the following indexes need to be added

Use cases

- Install Magento with Magento 2 Community Edition (CE) including example data step by step = Install Guide for **Magento 2 CE/EE** + **Pacemaker Community Edition (CE)** + **example data** step by step

In this Quickstart Tutorial, you will find detailed information about Pacemaker - Simple Console Tool installation and usage.

To give you a better impression, what is possible with **Pacemaker Community Edition (CE)**, we are running an example that is based on a **Magento 2 CE/EE** Sample Data installation.

Follow the steps to delete, replace and add/update products + categories in a **Magento 2 CE/EE** installation with the Sample Data provided by **Magento 2 CE/EE**.

NOTE	- You can find the sources for **Magento 2 CE/EE** 2 on the **Magento 2 CE/EE** website - It is best practice to download the complete package directly from the **Magento 2 CE/EE** website - Under the precondition that you have a running **Magento 2 CE/EE** system, follow the instructions in the **Magento 2 CE/EE** guide to install your current **Magento 2 CE/EE** instance

- Step 1: Install Magento 2 CE/EE >= 2.3 using composer

- Step 2: Create database recommended by Magento 2 CE/EE without the official Magento 2 CE/EE example data as a blank installation

- Step 3: Set up a local host for your instance to access the installation via browser

- Step 4: [Optional!] - Create additional Websites/Stores/Storeviews if needed

- Step 5: Install Pacemaker and sample data

- Step 6: Create required folders

- Step 7: [Optional!] - add symlinks to the images if needed

- Step 8: Running the import

Step 1: Install Magento 2 CE/EE >= 2.3 using composer

- Requirements for a successful local **Magento 2 CE/EE** installation in the current version:
    - Composer
    - PHP >= 7.3
    - Local server to host a local instance
    - Local **HTTPS** support
    - If nessary, a **Magento Marketplace Account** to access the **Magento** Repository is needed as well
    - Follow the installation steps recommended by **Magento 2 CE/EE** to set up a **Magento 2 CE/EE** instance locally successfully

Step 2: Create database recommended by Magento 2 CE/EE without the official Magento 2 CE/EE example data as a blank installation

```
# navigate into your magento working directory
cd <magento-install-dir>

# create the magento database tables
bin/magento setup:install \
--admin-firstname Admin \
--admin-lastname Developer \
--admin-email johndoe@m241ce-pacemaker-exampledata.test \
--admin-user admin \
--admin-password admin123 \
--base-url https://m241ce-pacemaker-exampledata.test/ \
--db-host 127.0.0.1:3307 \
--db-name my-local-magento-instance \
--db-user root --db-password root --use-rewrites 1 \
--backend-frontname admin \
--currency EUR --timezone Europe/Berlin \
--cleanup-database
```

Step 3: Set up a local host for your instance to access the installation via browser

- Depending on your local development environment you use (Nginx…..), set up a locally accessible host
- e.g. https://m241ce-pacemaker-exampledata.test/

Step 4: [Optional!] - Create additional Websites/Stores/Storeviews if needed

- If required, now is the time to create all nessary Store Websites/Stores/Store View

Step 5: Install Pacemaker and sample data

- Download the Pacemaker sample **CSV** import files from the techdivision/import-cli-simple repository.

Add the repository with the sample data to your **Magento 2 CE/EE** installation by running the following command:

Step 6: Create required folders

<magento-install-dir>/pub/media/catalog if not exists

```
# navigate into your magento working directory
cd <magento-install-dir>

# create the directories catalog and product at once
mkdir pub/media/catalog

# or as shortcut command
composer require techdivision/import-cli-simple \
&& composer require techdivision/import-sample-data
```

The Pacemaker Community CLI well as the Sample Data files for your **Magento 2 CE/EE** version should now be available under the <magento-install-dir>/vendor/bin/techdivision/import-cli-simple and <magento-install-dir>/vendor/bin/techdivision/import-sample-data directories.

Step 7: [Optional!] - add symlinks to the images if needed

- Additionally, the product images can be linked in the Magento media directory

By default, the images are **not** copied to the media directory; they can be linked to make them visible in the frontend after importing the products. That can be done by creating symlinks to access the example data images

```
# navigate into your magento working directory
cd <magento-install-dir>

# create symlink to the product image files
ln -s vendor/techdivision/import-sample-data/generic/media/catalog/product \
pub/media/catalog/product
```

Step 8: Running the import

- Suppose this is the first import, which we expect, the default import directory <magento-install-dir>/var/importtexport must be created first
- All files with the sample data (attribute sets, attributes, categories, and products) need to be copied into this directory

The command for the import of the sample data plus the images looks like following:

```
# navigate into your magento working directory
cd <magento-install-dir>

# create the importexport directory
mkdir var/importexport

# copy all files to forlder importexport
cp vendor/techdivision/import-sample-data/generic/data/attributes-set/add-update/*.csv
var/importexport \
&& cp vendor/techdivision/import-sample-data/generic/data/attributes/add-update/*.csv
var/importexport \
&& cp vendor/techdivision/import-sample-data/generic/data/categories/add-update/*.csv
var/importexport \
&& cp vendor/techdivision/import-sample-data/generic/data/products/add-update/*.csv
var/importexport

# start import
vendor/bin/import-simple import:create:ok-file \
&& vendor/bin/import-simple import:attributes:set \
&& vendor/bin/import-simple import:attributes \
&& vendor/bin/import-simple import:categories \
&& vendor/bin/import-simple import:products
```

CAUTION   Only if a **ok** flag file is available in the same directory where the CSV files are located, the import process will start

The naming convention for the **ok** flag file **must** follow one of these naming conventions:

- <IMPORT-DIRECTORY>/<PREFIX>.ok
- <IMPORT-DIRECTORY>/<PREFIX>_<FILENAME>.ok
- <IMPORT-DIRECTORY>/<PREFIX>_<FILENAME>_<COUNTER>.ok

which results in one of

- import-cli-simple/projects/sample-data/tmp/magento-import.ok
- import-cli-simple/projects/sample-data/tmp/magento-import_20170203.ok
- import-cli-simple/projects/sample-data/tmp/magento-import_20170203_01.ok

In case we have got a Bunch , the flag file **must** contain the name of the CSV files that have to be imported within the next iterations.

If the flag file would be named <magento-install-dir>/var/importexport/product-import_20161021-161909.ok for example and contains the following lines:

- product-import_20161021-161909_01.csv
- product-import_20161021-161909_02.csv
- product-import_20161021-161909_03.csv

- product-import_20161021-161909_04.csv

The importer has to be invoked four times (because the example above is **no** Bunch), whereas, on each innovation, the next file will be imported and removed from the flag file.

Take a look in the subdirectories of <magento-install-dir>/vendor/bin/techdivision/import-sample-data/ for a working examples.

Step 9: Add Missing Indexes

As the **Pacemaker Community Edition (CE)** functionality differs from the **Magento 2** standard, for performance reasons, it is necessary to add some essential indexes manually.

To do that, open a MySQL command line or MySql-Tool of your choice (e.g. Sequel Ace), connect to your MySQL server instance and enter the following SQL statement

| TIP | When using the Magic 360 component, additional the following indexes need to be added |

```
ALTER TABLE `eav_attribute_option_value` ADD INDEX `EAV_ATTRIBUTE_OPTION_VALUE_VALUE`
(`value` ASC); \
ALTER TABLE `catalog_product_entity_int` ADD INDEX `CATALOG_PRODUCT_ENTITY_INT_VALUE`
(`value` ASC); \
ALTER TABLE `catalog_product_entity_varchar` ADD INDEX
`CATALOG_PRODUCT_ENTITY_VARCHAR_VALUE` (`value` ASC); \
ALTER TABLE `catalog_product_entity_decimal` ADD INDEX
`CATALOG_PRODUCT_ENTITY_DECIMAL_VALUE` (`value` ASC); \
ALTER TABLE `catalog_product_entity_datetime` ADD INDEX
`CATALOG_PRODUCT_ENTITY_DATETIME_VALUE` (`value` ASC); \
ALTER TABLE `url_rewrite` ADD INDEX `URL_REWRITE_ENTITY_ID` (`entity_id` ASC); \
ALTER TABLE `url_rewrite` ADD INDEX `URL_REWRIRE_ENTITY_TYPE_ENTITY_ID` (`entity_id` ASC,
`entity_type` ASC); \
ALTER TABLE `catalog_product_entity_media_gallery` ADD INDEX
`CATALOG_PRODUCT_ENTITY_MEDIA_GALLERY_VALUE` (`value`);
```

Step 10: Clean up

- To see the imported data, call following commands:

```
# navigate into your magento working directory
cd <magento-install-dir>


bin/magento indexer:reindex
bin/magento c:f


# or as one command
bin/magento indexer:reindex && bin/magento c:f
```

Final Result

- Following all steps, here the final result of a fully functional **Magento** Shop including **Pacemaker Community Edition (CE)** example data
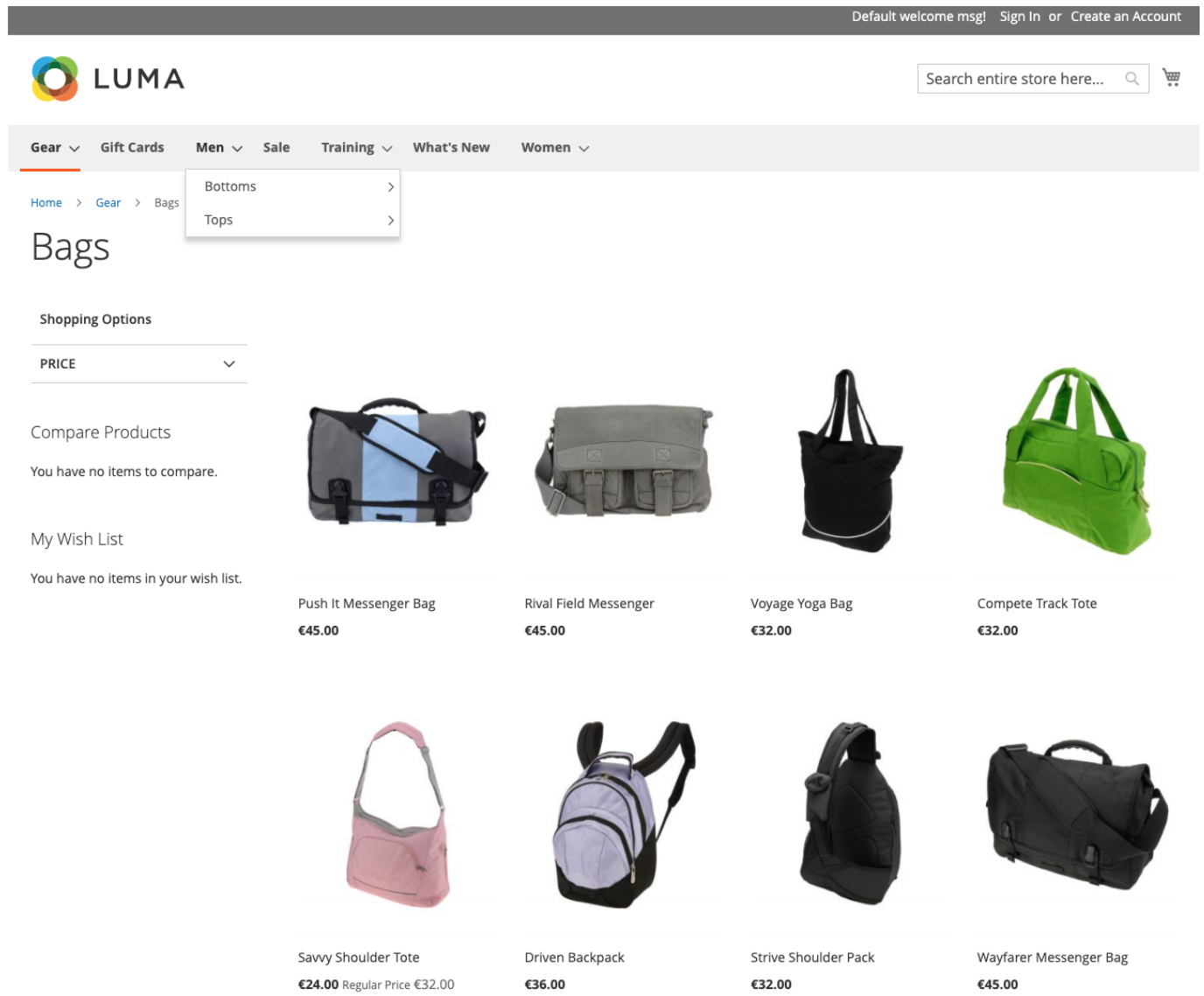
Usage

CAUTION

As of version 3.8.0, the structure of the setup has changed considerably, and the previous configuration files are no longer be used.

To avoid complex adjustments of the configuration, version 3.8.0 merged the configuration for all entities into one but dedicated overwriting of individual settings is now possible, e.g. for the log level.

After installation, the importer is ready-to-run.

When you're in the root of your **Magento 2** installation, you do **not** need to specify the configuration for the Workflow Engine. The importer parses the installation's configuration under app/etc/env.php and loads the **Magento** Edition/Version as well as

TechD•vision

the database configuration.

By default, the importer searches for CSV files in the directory var/importexport.

The files **must** have the prefix product-import, followed by a date/timestamp like 20180403-190920, a incremental number like 01 and the file suffix .csv.

Assuming, your CSV file var/importexport/product-import_20180403-190920_01.csv is ready to be imported and you're using the **PHAR**, you can start the importer with

```
bin/import-cli-simple.phar import:create:ok-file && bin/import-cli-simple.phar
import:products
```

The first command creates the mandatory .OK file that signals, that all import artifacts are on-place and the second command finally starts the import with the add-update operation on your **Magento 2** installation.

The import commands support an argument as well as several options.

Commands

Besides the import commands, several other more or fewer helper commands are available. The following commands for importing the entities are accessible:

Argument	Description	Format
import:categories	Starts importing categories	Customer + Customer Address Import
import:customers	Starts importing customers	Customer + Customer Address Import
import:customers:address	Starts importing customer addresses, expects that the customers are available	Customer + Customer Address Import
import:attributes:set	Starts importing attribute sets and their groups	Attribute Set + Group Import
import:attributes	Starts importing attributes, expects that the referenced attribute sets + groups are available	Attribute Import
import:products	Starts the product import, expects that the referenced attributes, and the attribute sets and groups, are available	Product Import
import:products:inventory	Starts importing product inventory, expects that the products are available	Product Import
import:products:inventory:msi	Starts importing product MSI inventory, expects that the products are available	Product Import
import:products:price	Starts importing product prices, expects that the products are available	Product Import
import:products:price:tier	Starts importing product tier prices, expects that the products are available	Product Import // Tier Price
import:products:url	Starts importing product URLs, expects that the products are available	Product Import // URL Rewrites

By default, if no other source directory has been configured, either as command-line option or in the configuration file, all

commands are searching for the CSV files and the matching OK file in the var/importexport directory of your **Magento** installation.

Arguments

For the commands described above, the following configuration arguments are available:

Argument	Description	Default value
shortcut	Specify the shortcut name to execute, either one of validating, convert, add-update, replace or delete (for further information look at the next section)	n/a

Shortcuts

In contrast to the **Magento 2** standard import functionality, up from version 3.8.0, **Pacemaker Community Edition (CE)** will provide two additional import shortcuts (formerly operations):

Shortcut	Description
validate	Performs an explicit validation of the data in the respective CSV file (available for **all** entities)
convert	Extracts the attributes or categories from given CSV files with products (available for **attributes** and **categories**)
add-update	New product data is added to the existing product data for the existing entries in the database. All fields except SKU can be updated. New tax classes that are specified in the import data are created automatically. New **SKUs** that are specified in the import file are created automatically.
replace	The existing product data is replaced with new data. If an SKU in the import data matches the **SKU** of an existing entity, all fields, including the **SKU** are deleted, and a new record is created using the CSV data. An error occurs if the CSV file references an SKU that does not exist in the database.
delete	Any entities in the import data that already exist in the database are deleted from the database. Delete ignores all columns in the import data, except for **SKU**. You can disregard all other attributes in the data. An error occurs if the CSV file references an **SKU** that does not exist in the database.

IMPORTANT	Exercise caution when replacing data because the existing product data will be completely cleared, and all references in the system will be lost.

Options

The following configuration options are available:

Option	Description	Default value
--serial	Specify the unique identifier of this import process which will also be the name of the temporary import directory	Some UUID
--configuration	Specify the pathname to the configuration file to use	n/a
--custom-configuration-dir	Specify the path to the custom configuration directory containing snippets to override the default values with	n/a
--pid-filename	The explicit PID filename to use	<system-temp-dir>/importer.pid
--system-name	The system name to be used (will added to the mail subject, if mails are configured)	The hostname
--installation-dir	The **Magento** installation directory to which the files has to be imported	The actual working directory
--source-dir	The directory that has to be watched for new files	n/a
--target-dir	The target directory with the files that has been imported	n/a
--archive-dir	The directory with the archived files that has been imported	n/a
--archive-artefacts	The flag to activate the artefact archiving functionality	true
--clear-artefacts	The flag whether or not the import artefacts have to be cleared	true
--magento-edition	The **Magento** edition to be used, either one of **CE** or **EE**	n/a
--magento-version	The **Magento** version to be used, e.g. 2.1.2	n/a
--use-db-id	The ID of the database to use, if not specified, the database with the default flag will be used	n/a
--db-pdo-dsn	The DSN used to connect to the **Magento** database where the data has to be imported, e.g. mysql:host=127.0.0.1;dbname=magento	n/a
--db-username	The username used to connect to the **Magento** database	n/a
--db-password	The password used to connect to the **Magento** database	n/a

TechD•vision

Option	Description	Default value
--db-table-prefix	The table prefix used by the **Magento** database	n/a
--debug-mode	The flag to activate the debug mode	false
--log-level	The log level to use (see Monolog documentation for further information)	info
--single-transaction	The flag to wrap the import process into a single transaction	false
--params	A JSON encoded string that'll be merged with the parameter from the configuration file (has to be in the same format)	n/a
--params-file	The path to a file with the JSON encoded parameter that will be merged with the parameter from the configuration file (has to be in the same format)	n/a
--cache-enabled	Whether or not the cache functionality for cache with the type cache.configurable should be enabled	false
--move-files-prefix	The prefix of the files that should be imported and moved to the temporary directory of the import	Defaults to the prefix of the first plugin subject

Besides the configuration option, all options can and **should** be defined in the configuration file.
The command-line options should only be used to override these values in some circumstances.

> **TIP**
>
> If the configuration option has **not** been specified, the system tries to locate the **Magento** Edition, based on the specified installation-dir option.
>
> If the installation-dir option **IS** specified explicitly, and the directory is a valid **Magento** root directory, the application tries to load database credentials from the app/etc/env.php script, so it is **not** necessary to specify a database configuration, nor in the configuration file or as a command-line parameter.

Debug mode

The debug mode provides a more detailed logging output (including PHP version, a list with activated extensions and a check if XDebug is enabled), by automatically setting the Monolog log level to LogLevel::DEBUG if **not** overwritten with the command-line option --log-level.

Additionally, it ignores

- Product category relations that not exists

- product images that are **not** available

- product images that can **not** be cleaned-up, e.g., the files are **not** available anymore

- product links (related, upsell, crosssell, etc.) for SKUs which are **not** available anymore

- product tier prices that can **not** be deleted, e.g. they are **not** available anymore

- product URL rewrites that can **not** be created

- configurable products for SKUs which are **not** available or available more than one time

- the archive plug-in is **not** able to create the archive directory, e.g. invalid permissions

- missing option values

- mission option values plug-in will send a CSV file with the disappeared option values to the configured mail address

but logs these issues as warnings to the console.

When the debug mode has been enabled, missing attribute option values will **not** throw an exception, instead they will be logged and put on an internal stack. If the MissingOptionValuesPlugin has been enabled, and put on an internal stack. If the MissingOptionValuesPlugin has been enabled, a CSV file with the missing option values will be created in the temporary import folder. If the plug-in configuration has enabled a Swift Mailer, the CSV file will be sent to the given mail addresses.

That will help developers test imports with partially invalid CSV files that do **not** break data consistency.

CAUTION	To improve performance, up from version 3.6.0, metadata that allows retracing above artifacts where the field data has been originated, is only available in debug mode. In production mode, exceptions will now only contain the name and line number of the actual artifact.

Parallel imports

To avoid unwanted behaviour, only one import process can be started at a time. To ensure that only one process is running, a **PID** file in the system's temporary directory (sys_get_temp_dir()) is created, which contains the **UUID** of the actual import process. After the import process has been finished, the file will be removed, and a new process can start. = Usage PHAR

Using **Pacemaker Community Edition (CE)** as PHAR usually has no impact.

As with version 3.8.0, where no dedicated configuration file is available, there is no need to reference a configuration file from within the **PHAR** anymore. = Explaining Bunches

The Pacemaker is able to handle Bunches.

In general, this functionality will only make sense in a multithreaded or multiprocessing environment where the bunches can be imported in parallel.

In this case, it should only give the developer an idea of how a multiprocess functionality can be implemented.

IMPORTANT	**A bunch is a CSV file, which is only a part of a complete import**

- It does not matter what kind of data a bunch contains, as the importer handles the data in the necessary order

- The first step is to import all simple products found in a bunch

- After that, information such as the created entity **IDs** is released in connection with the imported **SKUs**, which are necessary to import all other product data (bunches, configurables, images, related, etc.).

- It is possible to import this data step by step, but each step also in parallel.

Split an import into multiple bunches; the bundled files *MUST* follow the following pattern:

- The prefix has to equal, e.g. product-import

- The prefix has to be followed by an underscore (_)

- A random number of alphanumeric characters has to follow.
- These characters have also to be followed by an underscore (_)
- Finally, each bunch **must** have a sequential number, followed by a .csv

For example, the following CSV files that contain the product sample data will be imported as a bunch:

- var/importexport/product-import_20170203-1234_01.csv
- var/importexport/product-import_20170203-1234_02.csv
- var/importexport/product-import_20170203-1234_03.csv
- var/importexport/product-import_20170203-1234_04.csv

When starting the import process by invoking the appropriate command, those files will be imported as one file.

CAUTION	It is **not** necessary to gather the importer four times

Configuration

CAUTION	As of version 3.8.0, the structure of the setup has changed considerably, and the previous configuration files are no longer be used.
	To avoid complex adjustments of the configuration, version 3.8.0 merged the configuration for all entities into one but dedicated overwriting of individual settings is now possible, e.g. for the log level.

If **no** configuration file is specified, it will be loaded and merged from individual parts delivered with the respective repositories.

In the case of the import:products, only the operations, specified by the given shortcut, e.g. add-update, will be used and executed.

By default, the configuration contains whether database configuration nor an image directory. The command-line options can specify the database configuration, it is necessary to provide a custom configuration snipped that activates the flags "copy-images":true, "clean-up-media-gallery": true" and "clean-up-empty-image-columns": true and contains the paths to the image files.

This snippet **requires** the **JSON** format, and can be named with any random name, e.g. images.json and has to be placed in the usage

By default, the value for the custom-configuration-dir is <magento-install-directory>/app/etc/configuration.

Configuration snippets

All configuration snippets **must** be in **JSON** format and have to be located in the custom configuration dir.

In general, nearly all available configuration options/arguments can and **should** be defined in those snippets, instead of passing them as command-line options.

The possibility to override configuration values on the command-line should only be used during development or in individual cases, e.g. when the values will be loaded from a third-party system.

The structure is separated into a general configuration section, the database configuration, the logger configuration, and the configuration for the available operations.

General configuration

The general configuration contains necessary metadata like the **Magento** edition and version, which will be needed when **Pacemaker Community Edition (CE)** will be invoked from a directory other than the **Magento** installation directory and without the --installation-dir option.

Also, it can be used to specify default values, if you want to change the default operation from add-update to replace for example.

The change that general metadata we recommend to place a snipped, e.g., with the name <custom-configuration-dir>/configuration.json in the custom configuration directory.

```
{
  "magento-edition": "CE",
  "magento-version": "2.3.1",
  "operation-name" : "replace",
  "table-prefix": "test_",
  "installation-dir" : "/var/www/magento"
}
```

Global parameter

Global parameter in the configuration file enable developers to pass specific configuration values from the configuration file itself. To serve the same purpose, the command-line (using the --params option) or an additional file (using the --params-file option) through their import logic, e.g., project-specific observers can be used.

In a configuration snippet

It's possible to create a snippet with params, e.g. <custom-configuration-dir>/params.json which contains values like

```
{
  "params": {
    "my-website-country-mapping": {
      "DE": [ "de_DE", "de_AT", "de_CH" ],
      "EN": [ "en_US", "en_UK" ]
    }
  }
}
```

These parameter can then be used wherever access to the configuration object is available, e.g. in an observer like

```
namespace My\Project;

use TechDivision\Import\Observers\AbstractObserver;

/**
 * A custom observer implementation.
 */
class MyObserver extends AbstractObserver
```

```
{

    /**
     * Return's the global param with the passed name.
     *
     * @param string $name         The name of the param to return
     * @param mixed  $defaultValue The default value if the param doesn't exist
     *
     * @return string The requested param
     * @throws \Exception Is thrown, if the requested param is not available
     */
    public function getGlobalParam($name, $defaultValue = null)
    {
        return $this->getSubject()->getConfiguration()->getConfiguration()->getParam($name,
$defaultValue);
    }

    /**
     * Will be invoked by the action on the events the listener has been registered.
     *
     * @param \TechDivision\Import\Subjects\SubjectInterface $subject The subject instance
     *
     * @return array The modified row
     */
    public function handle(SubjectInterface $subject)
    {

        // load the params from the configuration
        $myWebsiteMapping = $this->getGlobalParam('my-website-country-mapping');

        // do something with the configuration value
    }
}
```

As command-line option

Besides the params that can be defined in the configuration file itself, it is possible to specify additional params using the command-line as an option

```
bin/import-cli-simple.phar import:products \
    --params='{ "params": { "my-website-country-mapping": { "DE": [ "de_LI" ] } } }'
```

which will append the value de_LI to the DE param of the my-website-country-mapping.

As file, defined as command-line option

Besides the possibility of specifying the params directly as a command-line option, it is also possible to specify a path to a file containing the JSON encoded params. The file **requires** the following format.

```
{
  "params": {
    "my-website-country-mapping": {
      "DE": [ "de_LI" ]
    }
  }
}
```

IMPORTANT   Values from the configuration file will be overwritten with values delivered by command-line use, which again will be overwritten with the values from an additional file that has been specified with the --params-file option.

Extend Pacemaker Community Edition (CE) with additional libraries

In more complex projects, it'll we possible, that additional libraries are necessary. As the **Pacemaker Community Edition (CE)** Console Tool uses a Symfony DI container. It is required to register the other library by adding it to the configuration file. Depending on how the **Pacemaker Community Edition (CE)** Console Tool has been installed, there are two options.

CAUTION   If you write an extension library, do **not** forget to provide the **Symfony DI** configuration.

Extension libraries

Assuming that the **Pacemaker Community Edition (CE)** Console Tool has been installed as Composer library, together with a **Magento 2** installation, the simplest way to register an additional extension is to add a snippet, e.g. <custom-configuration-dir>/extension-libraries.json that contains the name of the library, like

```
{
  "extension-libraries" : [
    "techdivision/import-product-magic360"
  ]
}
```

NOTE   This is **only** possible if the extension library uses the **same** Composer autoloader as **Pacemaker Community Edition (CE)** Console Tool does.

As a natural outcome, if you're using the **PHAR** version of **Pacemaker Community Edition (CE)**, it is **required** use the additional-vendor-dirs directive

Additional vendor directories

Suppose that the **Pacemaker Community Edition (CE)** Console Tool **PHAR** archive will be used. The Composer class loader of the additional library vendor directory must be added by a snippet, e.g. <custom-configuration-dir>/additional-vendor-dirs.json, which contains the following content

```
{
  "additional-vendor-dirs" : [
    {
      "vendor-dir": "vendor",
```

```
      "relative": true,
      "libraries": [
        "techdivision/import-adapter-custom"
      ]
    }
  ]
}
```

Events

Beside the configured worklfow with Plugins, Subjects, Observers and Callbacks, addtional events are available to add custom functionality.

Event Name	Description	Since Version
app.set.up	Is triggered before the import will be processed.	
app.tear.down	Is triggered after the import has been processed.	
app.process.transaction.start	Is triggered before the application start's the transaction.	
app.process.transaction.success	Is triggered after the application has the transaction committed successfully.	
app.process.transaction.failure	Is triggered after the application rollbacked the transaction.	
app.process.transaction.finished	Is triggered after the application transaction has been finished (either it has been successful or not).	3.8.0
subject.artefact.process.start	Is triggered before an import artifact will be processed.	
subject.artefact.process.success	Is triggered when an import artifact has successfully been processed.	
subject.artefact.process.failure	Is triggered when an import artifact can not be processed.	
subject.artefact.row.process.start	Is triggered when an import artifact has successfully been processed.	
subject.artefact.row.process.success	Is triggered when an import artifact has successfully been processed.	
subject.artefact.header.row.process.start	Is triggered before an import artifact's header row will be processed.	3.8.0
subject.artefact.header.row.process.start	Is triggered when an import artifact's header row has successfully been processed.	3.8.0
plugin.process.start	Is triggered before the plugin's process() method will be processed.	3.4.0
plugin.process.success	Is triggered after a plugin's process() method has been processed.	3.4.0
plugin.process.failure	Is triggered when an exception has been thrown during the plugin's process() method is processed.	3.4.0
plugin.export.start	Is triggered before a plugin's export() method will be processed.	3.8.0
plugin.export.success	Is triggered after a plugin's export() method has been processed.	3.8.0
plugin.export.failure	Is triggered when an exception has been thrown during the plugin's export() method is processed.	3.8.0
subject.import.start	Is triggered before a subject's import() method will be processed.	3.4.0

Event Name	Description	Since Version
subject.import.success	Is triggered after a subject's import() method has been processed.	3.4.0
subject.import.failure	Is triggered when an exception has been thrown during the subject's import() is processed.	3.4.0
subject.export.start	Is triggered before a subject's export() method will be processed.	3.4.0
subject.export.success	Is triggered after a subject's export() method has been processed.	3.4.0
subject.export.failure	Is triggered when an exception has been thrown during the subject's export() is processed.	3.4.0

Besides the possibility of registering global events, up with version 3.4.0, it is possible to register events on a plugin and subject level. It avoids execution of events that only provide functionality for a dedicated plugin or subject, so keep in mind that they will **not** be fired for every plugin or every subject.

Plugin level

The listeners will only be executed before, after, or on the plugin's failure, for the configured event. Configuration in a snippet, e.g. <customer-installation-dir>/operations.json, can look like

```json
{
  "operations": {
    "general": {
      "catalog_product_tier_price": {
        "add-update": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "listeners": [
                {
                  "plugin.process.success": [
                    "import_product_tier_price.listener.delete.obsolete.tier_prices"
                  ]
                }
              ],
              "subjects": [ ... ]
            }
          }
        }
      }
    }
  }
}
```

Subject level

As subjects are responsible for importing **and** exporting artifacts, events for both steps have been added.

The listeners will only be executed before, after, or on the subject's failure, for the configured event.

Configuration in a relevant snippet, e.g. <customer-installation-dir>/operations.json, can look like

```
{
  "operations": {
    "general": {
      "catalog_product_tier_price": {
        "add-update": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "subjects": [
                {
                  "id": "import_product_tier_price.subject.tier_price",
                  "listeners": [
                    {
                      "subject.import.success": [
                        "import_product.listener.register.sku.to.pk.mapping"
                      ]
                    }
                  ],
                  "observers": [ ... ]
                }
              ]
            }
          }
        }
      }
    }
  }
}
```

Default listeners

By default, **Pacemaker Community Edition (CE)** comes with several listeners registered.

```
{
  "listeners": [
    {
      "app.set.up": [
        "import.listener.render.ansi.art",
        "import.listener.render.operation.info",
        "import.listener.render.mysql.info",
        "import.listener.render.debug.info",
        "import.listener.initialize.registry"
      ],
      "app.process.transaction.success": [
        "import.listener.finalize.registry",
        "import.listener.archive",
```

```
      "import.listener.clear.artefacts",
      "import.listener.clear.directories",
      "import.listener.operation.report"
    ],
    "app.process.transaction.failure": [
      "import.listener.finalize.registry",
      "import.listener.render.validations"
    ],
    "app.tear.down": [
      "import.listener.import.history",
      "import.listener.clear.registry"
    ]
  }
 ]
}
```

The first one, named import.listener.render.ansi.art renders the nice ASCII art when invoking the CLI. The second and third parties are mandatory as they are responsible for initializing the registry of the actual import.

NOTE	As a solution partner, feel free to replace the **ASCII** art renderer with one of your choices, e.g. rendering your company logo.

Database

The configuration allows the registration of multiple databases, for example in a snippet <custom-configuration-dir>/databases.json, which can look like

```
{
  "databases": [
    {
      "id": "local",
      "default": false,
      "pdo-dsn": "mysql:host=127.0.0.1;dbname=appserver_magento2_ee212",
      "username": "your-username",
      "password": "your-password"
    },
    {
      "id": "remote",
      "default": true,
      "pdo-dsn": "mysql:host=127.0.0.130;dbname=appserver_magento2_ee212",
      "username": "your-username",
      "password": "your-password"
    }
  ]
}
```

Depending on the command-line option --use-db-id and the specified value, the database with the given ID will be used. If the command-line options are **not** specified, the one with the flag "default": true will be used.

If not found, the first configured database will be used.

If a value for the command-line option --db-pdo-dsn has been specified, the --use-db-id option will be ignored, and the given DSN value will be used for database connection instead. Additionally, the credentials, by using the --db-username and --db-password options also need to be specified.

Loggers

Pacemaker Community Edition (CE) uses Monolog to provide the basic logging functionality. Therefore, at least one logger instance is necessary. By default, if no logger has been configured, a system logger will be installed, which writes log messages to the error log that has been set in the php.ini file of the used PHP installation.

To add additional loggers, e.g. in case you want to send mails if an exception has been thrown, the configuration can be extended, e.g. with a snippet <custom-configuration-dir>/loggers.json which looks like

```json
{
  "loggers": {
    "mail": {
      "id": "import.logger.factory.monolog",
      "channel-name": "logger/mail",
      "handlers": [
        {
          "id": "import.logger.factory.handler.swift",
          "formatter": {
            "id": "import.logger.factory.formatter.line",
            "params": {
              "format": "[%datetime%] %channel%.%level_name%: %message% %context% %extra%",
              "date-format": "Y-m-d H:i:s",
              "allow-inline-line-breaks": true,
              "ignore-empty-context-and-extra": true
            }
          },
          "params": {
            "log-level": "error",
            "bubble": false
          },
          "swift-mailer" : {
            "id" : "import.logger.factory.transport.swift.smtp",
            "params" : {
              "to" : "tw@techdivision.com",
              "from" : "pacemaker@techdivision.com",
              "subject": "Something Went Wrong",
              "content-type" : "text/plain"
            },
            "transport" : {
              "params" : {
                "smtp-host" : "mail.techdivision.com",
                "smtp-port" : 25
              }
            }
          }
```

```
            }
        }
    ]
  }
}
```

You can also use a handler for Native Mail Sender, which takes the settings of the PHP configuration.

```
{
    ...,
    "handlers": [
        {
            "id": "import.logger.factory.handler.native.mail.log",
            "formatter": {
                "id": "import.logger.factory.formatter.line",
                "params": {
                    "format": "[%datetime%] %channel%.%level_name%: %message% %context%
%extra%",

                    "date-format": "Y-m-d H:i:s",
                    "allow-inline-line-breaks": true,
                    "ignore-empty-context-and-extra": true
                }
            },
            "params": {
                "to": [
                    "tw@techdivision.com"
                ],
                "subject": "Something Went Wrong",
                "from": "pacemaker@techdivision.com",
                "level": "error",
                "bubble": true,
                "maxColumnWidth": 500
            }
        }
    ],
    ...
}
```

This will add a mail logger and set the default log level to **error**.

Aliases

Aliases can be used to override classes with custom functionality, e.g. provided by a project-specific library. For example, if a custom adapter for caching is used, the default class behind the **Symfony DI** configuration of the cache.adapter can be replaced by the customer DI identifier `import_parallel .cache .adapter.redis` in a snippet, e.g. <custom-installation-dir>/aliases.json.

```
{
```

```
  "aliases": [
    {
      "id": "cache.adapter",
      "target": "import_parallel.cache.adapter.redis"
    }
  ]
}
```

| CAUTION | Overriding classes can be dangerous and should only be done when you aware of what you're doing. |

Images

When you want to copy images from a source directory to the **Magento** pub/media directory, you can add a snippet, e.g. <custom-configuration-dir>/operations.json, that overrides the default operation.

In case images for the products has to be copied from directory var/importexport/media/wysiwyg to the appropriate target directory pub/media/catalog/product the copy-images flag has to be set to true as well as the values for the media-directory (which is the target directory), and the images-file-directory (which is the source directory) has to be also specified.

- With the option "override-images": true|false you are able to configure how to deal with images during an import that already exists in the target directory.
- Is the option set to override-images": false, the existing file is left as it is, but the filename of the new file increments with a counter, and the latest version is assigned to the product.
- If the option is override-images": true, the existing image will be overwritten.

Example Configuration:

```
{
  "operations": {
    "ce": {
      "catalog_product": {
        "add-update": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "subjects": [
                {
                  "id": "import_product.subject.bunch",
                  "file-resolver": {
                    "prefix": "product-import"
                  },
                  "params": {
                    "copy-images": true,
                    "override-images": false,
                    "media-directory" : "pub/media/catalog/product",
                    "images-file-directory" : "var/importexport/media/wysiwyg",
                    "clean-up-media-gallery": true,
                    "clean-up-empty-image-columns": true,
                    "clean-up-website-product-relations": true,
```

```
                    "clean-up-category-product-relations": true,
                    "clean-up-empty-columns": [
                      "special_price",
                      "special_price_from_date",
                      "special_price_to_date"
                    ]
                  },
                  "observers": [
                    {
                      "import": [
                        "import_product.observer.composite.base.add_update"
                      ]
                    }
                  ]
                }
              ]
            }
          }
        }
      }
    }
  }
}
```

In the CSV file, the path to the images has to start with a / like /m/b/mb01-blue-0.jpg. Pay attension to the example files in the repository techdivision/import-sample-data;

	When you copy images, you **must** create resized versions of them. Therefore, you can invoke the command bin/magento catalog:images:resize on the CLI. If you missed that, the images will be rendered in the admin area, but **not** on the frontend!
CAUTION	Please also keep in mind that it won't usually make sense to copy the images during the import process, as this will slow down the performance significantly.
	It is strongly recommended to copy the pictures to the appropriate folder in your **Magento** installation and only importing the path to the images.

Header mappings

In some cases, it can be convenient to map column names to the appropriate attributes. It's a built-in feature and can simply be configured by adding a snippet, e.g. <custom-configuration-dir>/header-mappings.json that contains an array with header mappings like

```
{
  "header-mappings": {
    "catalog_product": {
      "my_sku_column": "sku",
      "my_qty_column": "qty",
      ...,
    }
```

```
    }
}
```

In the example above, the column name my_sku_column will automatically be mapped to the mandatory column SKU. It happens **before** any columns will be processed and allows vendors to configure any column to the appropriate attribute without custom development.

NOTE	Keep in mind that the header mappings are valid for all operations within the configuration file

Product links

Pacemaker Community Edition (CE) provides the functionality to import product links as well as their positions. _related_, _upsell_, and _crosssell_ links are supported.

As **Pacemaker Community Edition (CE)** also supports the import of _grouped_ products, which are nothing else than the additional link type super, it is possible to import the position of the products that are linked to a grouped product.

To import this link type, only a comma (,) separated list of the linked products must be specified in the appropriate column

- column related_skus for related products that'll be rendered on the product detail page
- column upsell_skus for upsell products that'll be rendered on the product detail page
- column crosssell_skus for cross-sell products that'll be rendered on the shopping cart
- column associated_skus for products that will be part of a grouped product, which will also be rendered on the product detail page

As example, the columns to import those link types may look like this

sku	...	related_skus	upsell_skus	crosssell_skus	associated_skus	...
24-MB01		24-MB02,24-MB03	24-WG03,24-WG04	24-SB01,24-SB02	24-AS01,24-AS02	

Link Positions

Besides the **SKUs** of the linked product itself, it is also possible to specify the related products

sku	...	related_position	upsell_position	crosssell_position	associated_position	...
24-MB01		1,2	2,1		1,2	

The position also has to be a comma (,) separated list, but it contains numbers instead of the SKUs. The numbers finally are the position the linked product will be rendered in the GUI.

CAUTION	Please be aware that the positions are **not** mandatory, and the columns can be empty. If so, the **Pacemaker Community Edition (CE)** creates the place itself, based on the order of the given **SKUs**. If positions are changed in the **Magento** backend, they will be **overwritten** within the next import process.

Magento 2 CE < 2.1.6

Magento 2 CE supports positions for product links, as well as **Magento 2 EE**. By default, up to version 2.1.6, importing product positions is **not** possible in the CE, because of the database of the CE lacks missing rows in the catalog_product_link_attribute table.

In case that the rows are not available, the positions, defined in the CSV file's columns are ignored.

- related_position

- crosssell_position

- upsell_position

To enable importing positions, add the following rows the **Magento 2 CE** database

```
INSERT INTO
      `catalog_product_link_attribute` (
          `link_type_id`,
          `product_link_attribute_code`,
          `data_type`
      )
   VALUES
      (1,'position','int'),
      (4,'position','int'),
      (5,'position','int');
```

IMPORTANT | Make sure that the values are **not** already available before adding them.

Dynamic option creation

Up from version 3.8.0, missing product option values for the admin store will be created by default.

To disable this, override the default add-update shortcut (defined in the configuration snipped of the techdivision/import-product library), by adding a snippet, e.g. <custom-configuration-dir>/shortcuts.json and remove the eav_attribute related operations.

```
{
  "shortcuts": {
    "ce": {
      "catalog_product": {
        "add-update": [
          ...
          "general/eav_attribute/convert",
          "general/eav_attribute/add-update.options",
          "general/eav_attribute/add-update.option-values",
          "general/eav_attribute/add-update.swatch-values",
          ...
        ]
      }
    }
```

```
    }
  }
```

a file outcome should that looks like you can see below

```
{
  "shortcuts": {
    "ce": {
      "catalog_product": {
        "add-update": [
          "general/general/global-data",
          "general/general/move-files",
          "general/catalog_product/collect-data",
          "general/catalog_category/convert",
          "ce/catalog_category/sort",
          "ce/catalog_category/add-update",
          "ce/catalog_category/add-update.path",
          "ce/catalog_category/add-update.url-rewrite",
          "general/catalog_category/children-count",
          "general/catalog_product/validate",
          "ce/catalog_product/add-update",
          "ce/catalog_product/add-update.variants",
          "ce/catalog_product/add-update.bundles",
          "ce/catalog_product/add-update.links",
          "ce/catalog_product/add-update.grouped",
          "ce/catalog_product/add-update.media",
          "general/catalog_product/add-update.msi",
          "general/catalog_product/add-update.url-rewrites"
        ]
      }
    }
  }
}
```

CAUTION

Please be aware that it is **not** possible to create translations for the other stores other than the admin store itself.

When creating option values dynamically, the option values are taken from the column additional_attributes always have to be the admin store's values.

Translations can be done in the **Magento** backend or with the attribute import

Dynamic category creation

Up from version 3.8.0, missing categories in the product CSV file will be created by default, but only for the admin store.

To disable this, override the default add-update shortcut (defined in the configuration snipped of the techdivision/import-product library), by adding a snippet, e.g. <custom-configuration-dir>/shortcuts.json and remove the catalog_category related operations.

```
{
  "shortcuts": {
    "ce": {
      "catalog_product": {
        "add-update": [
          ...
          "general/catalog_category/convert",
          "ce/catalog_category/sort",
          "ce/catalog_category/add-update",
          "ce/catalog_category/add-update.path",
          "ce/catalog_category/add-update.url-rewrite",
          "general/catalog_category/children-count",
          ...
        ]
      }
    }
  }
}
```

a file outcome should that looks like you can see below

```
{
  "shortcuts": {
    "ce": {
      "catalog_product": {
        "add-update": [
          "general/general/global-data",
          "general/general/move-files",
          "general/catalog_product/collect-data",
          "general/eav_attribute/convert",
          "general/eav_attribute/add-update.options",
          "general/eav_attribute/add-update.option-values",
          "general/eav_attribute/add-update.swatch-values",
          "general/catalog_product/validate",
          "ce/catalog_product/add-update",
          "ce/catalog_product/add-update.variants",
          "ce/catalog_product/add-update.bundles",
          "ce/catalog_product/add-update.links",
          "ce/catalog_product/add-update.grouped",
          "ce/catalog_product/add-update.media",
          "general/catalog_product/add-update.msi",
          "general/catalog_product/add-update.url-rewrites"
        ]
      }
    }
  }
}
```

CAUTION

Please be aware that it is **not** possible to create categories for the other stores other than the admin store itself.

When creating categories dynamically, and the names are taken from the column categories, always **must** be the admin store's values.

Translations can be done in the **Magento** backend or with the category import .

Delta import

Besides the full import functionality, **Pacemaker Community Edition (CE)** also supports delta import functionality.

The delta import supports **SKUs** that is **not** part of one of the actual CSV files like listed below

- all link types (up-sell, cross-sell + related)
- variant products
- grouped products
- bundle products

In contrast to a full import, which requires the following invocation of the appropriate indexers, after a delta import, the delta indexing functionality takes care for the necessary index updates.

NOTE

The delta indexer can not be started on the command-line. Instead, it'll be initiated by the default **Magento CRON** job.

Clean-up

CAUTION

As deleting data can reduce performance significantly, the clean-up functionality should be used carefully!

Pacemaker Community Edition (CE) provides a **clean-up** functionality that offers the possibility to remove values for empty columns or relations that are no longer part of the CSV file.

It is beneficial when using the add-update operation because it will only be added or updated in general if already available.

Product import

Clean-Up functionality for the product import is **activated** by default and available for following listing below and as well as for all product attributes (clean-up-empty-columns) which needs additional configuration.

- Images (clean-up-empty-image-columns)
- Media Gallery (clean-up-media-gallery)
- Category Relations (clean-up-category-product-relations)
- Website Relations (clean-up-website-product-relations)
- Tier Prices (clean-up-tier-prices)
- URL Rewrites (clean-up-url-rewrites)

To activate the **clean-up** functionality for product attributes, as well as additional attributes, the columns that has to be **cleaned** need to be specified in the clean-up-empty-columns array like "clean-up-empty-columns": ["activity", "erin_recommends"].

NOTE	Remember that columns specified under 'additional_attributes' must be **cleaned** and provided with empty values, e.g. if the following columns 'activity' and 'in_recommends' must be **cleaned**, the column 'additional_attributes' requires the value '"activity=,erin_recommends="'.

To deactivate the **clean-up** functionality, add a snippet, e.g. <custom-configuration-dir>/operations.json and override the corresponding clean-up-* flags like

```
{
  "operations": {
    "general": {
      "catalog_product": {
        "add-update.url-rewrites": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "subjects": [
                {
                  "id": "import_product_url_rewrite.subject.url.rewrite",
                  "file-resolver": {
                    "prefix": "url-rewrite"
                  },
                  "params": {
                    "clean-up-url-rewrites": false
                  },
                  "observers": [
                    {
                      "import": [
                        "import_product_url_rewrite.observer.url.rewrite.update"
                      ]
                    }
                  ]
                }
              ]
            }
          }
        }
      },
      "catalog_product_tier_price": {
        "add-update": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "listeners": [
                {
                  "plugin.process.success": [
                    "import_product_tier_price.listener.delete.obsolete.tier_prices"
```

```
                    ]
                  }
                ],
                "params": {
                  "clean-up-tier-prices": false
                },
                "subjects": [
                  {
                    "id": "import_product_tier_price.subject.tier_price",
                    "listeners": [
                      {
                        "subject.import.success": [
                          "import_product.listener.register.sku.to.pk.mapping"
                        ]
                      }
                    ],
                    "file-resolver": {
                      "prefix": "product-import-tier-price"
                    },
                    "observers": [
                      {
                        "import": [
                          "import_product_tier_price.observer.tier_price.update"
                        ]
                      }
                    ]
                  }
                ]
              }
            }
          }
        },
        "ce": {
          "catalog_product": {
            "add-update": {
              "plugins": {
                "subject": {
                  "id": "import.plugin.subject",
                  "subjects": [
                    {
                      "id": "import_product.subject.bunch",
                      "file-resolver": {
                        "prefix": "product-import"
                      },
                      "params": {
                        "copy-images": false,
                        "clean-up-media-gallery": false,
                        "clean-up-empty-image-columns": false,
```

```
                    "clean-up-website-product-relations": false,
                    "clean-up-category-product-relations": false,
                    "clean-up-empty-columns": []
                },
                "observers": [
                  {
                    "import": [
                      "import_product.observer.composite.base.add_update"
                    ]
                  }
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

Images/Media gallery

In most cases, it is beneficial to delete images that are no longer in the CSV files from the database.

The image type, e.g. thumbnail as an attribute, links to the product, and the name and the position are stored in separate tables.

To clean-up both of them, it is necessary to set the flags clean-up-empty-image-columns and clean-up-media-gallery to true, to the default values.

If you're not confident what you're doing, both flags should have the same value.

Category relations

In many cases, product category relations don't change at all or change, not really often.

In case the product category relations change frequently, and it'll be necessary to update them with the add-update operation, this can be done by setting the flag clean-up-category-product-relations to true.

As the product category relation is not only persisted in a column, this relation can **not** be cleaned by adding a column name to the array clean-up-empty-columns.

Website relations

In many cases, product website relations don't change at all or change, not often. In case the product website relations change frequently, and it'll be necessary to update them with the add-update operation, this can be done by setting the flag clean-up-website-product-relations to true.

As the product website relation is not only persisted in a column, this relation can **not** be cleaned by adding a column name to the array clean-up-empty-columns.

Tier prices

Prices nearly always need to be up-to-date, so if tier prices are imported, the clean-up functionality should be activated by setting the flag clean-up-tier-prices to true.

URL rewrites

Cleaning up URL rewrites will be necessary in most cases and should be activated by setting the flag clean-up-url-rewrites to true,

Categories

As the category functionality in **Magento** is less complicated, the **clean-up** only provides one flag for the

- URL Rewrites (clean-up-url-rewrites)

As for the product import, the flags have to be configured on a subject level like

```
{
  "operations": {
    "ce": {
      "catalog_category": {
        "add-update.url-rewrite": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "subjects": [
                {
                  "id": "import_category.subject.bunch",
                  "file-resolver": {
                    "prefix": "category-url-rewrite"
                  },
                  "params": {
                    "clean-up-url-rewrites": false
                  },
                  "observers": [
                    {
                      "import": [
                        "import_category.observer.url.rewrite.update"
                      ]
                    }
                  ]
                }
              ]
            }
          }
        }
      }
    }
  }
}
```

```
}
```

Category url rewrites

Cleaning-Up URL rewrites will be necessary in most cases. To **deactivate** it, set the flag clean-up-url-rewrites to false. =
Validation

Up with version 3.8.0, validation for all entity types will activate by default.

Especially in case of large CSV files, let us suppose > 100 MB, validation can slow down the import process massively.
Therefore, the validation is highly customizable and can, if necessary, completely be switched off.

Switch-Off validaton

If you don't want your CSV files validated, you are able to override the appropriate shortcuts with a snippet, e.g.
etc/configuration/shortcuts.json.

For example, if you want to remove the validation from the add-update process of your product import, delete the operation
general/catalog_product/validate from the appropriate shortcut.

Finally, the snippet has to look like the following code:

```
{
  "shortcuts": {
    "ce": {
      "add-update": [
        "general/general/global-data",
        "general/general/move-files",
        "general/catalog_product/collect-data",
        "general/eav_attribute/convert",
        "general/eav_attribute/add-update.options",
        "general/eav_attribute/add-update.option-values",
        "general/eav_attribute/add-update.swatch-values",
        "general/catalog_category/convert",
        "ce/catalog_category/sort",
        "ce/catalog_category/add-update",
        "ce/catalog_category/add-update.path",
        "ce/catalog_category/add-update.url-rewrite",
        "general/catalog_category/children-count",
        "ce/catalog_product/add-update",
        "ce/catalog_product/add-update.variants",
        "ce/catalog_product/add-update.bundles",
        "ce/catalog_product/add-update.links",
        "ce/catalog_product/add-update.grouped",
        "ce/catalog_product/add-update.media",
        "general/catalog_product/add-update.msi",
        "general/catalog_product/add-update.url-rewrites"
      ]
    }
  }
```

```
    }
}
```

Custom validations

Use for each entity type a snippet, that declares the available operations in the corresponding repository configuration folder etc/configuration/operations.json.

Each of these files contains the configuration for the validation operation.

By overriding this definition, e.g. with a custom snippet like custom-configuration-dir>/operations.json, you've full control what will get validated and how the validation works.

In general, the validation operation is based on a validator subject, an observer, some listeners, and various callbacks.

Finally, the validations are implemented as callbacks that allow you to register one or more validation callbacks for each column.

Pacemaker Community Edition (CE) comes with some specialized callbacks that only can be used for the corresponding columns and a custom regex validator that can be used to integrate custom, regex-based validations.

To go into more details, let's take a look at the validation operation of the product import, which is declared in the operations.json of the techdivision/import-product repository.

To make it easier to understand, we've removed the unnecessary parts for you in this example:

```
{
  "operations": {
    "general": {
      "catalog_product": {
        "validate": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "subjects": [
                {
                  ...
                    "params" : {
                      "custom-validations" : {
                        "sku" :  [ "/.+/" ],
                        "product_type": [ "simple", "virtual", "configurable", "bundle",
"grouped", "giftcard" ],
                        "visibility": [ "Not Visible Individually", "Catalog", "Search",
"Catalog, Search" ]
                      }
                    },
                    "callbacks": [
                      {
                        "sku": [ "import.callback.custom.regex.validator" ],
                        "store_view_code": [ "import.callback.store.view.code.validator" ],
                        "attribute_set_code": [ "import.callback.attribute.set.name.validator"
```

```
                ],
                                "product_type": [ "import.callback.custom.array.validator" ],
                                "tax_class_id": [ "import_product.callback.validator.tax.class" ],
                                "product_websites": [ "import.callback.store.website.validator" ],
                                "visibility": [ "import.callback.visibility.validator" ],
                                "related_skus": [ "import_product.callback.validator.link" ],
                                "upsell_skus": [ "import_product.callback.validator.link" ],
                                "crosssell_skus": [ "import_product.callback.validator.link" ],
                                "created_at" : [ "import.callback.validator.datetime" ],
                                "updated_at" : [ "import.callback.validator.datetime" ],
                                "special_price_to_date" : [ "import.callback.validator.datetime" ],
                                "special_price_from_date" : [ "import.callback.validator.datetime" ],
                                "custom_design_to" : [ "import.callback.validator.datetime" ],
                                "custom_design_from" : [ "import.callback.validator.datetime" ],
                                "new_to_date" : [ "import.callback.validator.datetime" ],
                                "new_from_date" : [ "import.callback.validator.datetime" ],
                                "price" : [ "import.callback.validator.number" ],
                                "special_price" : [ "import.callback.validator.number" ],
                                "map_price" : [ "import.callback.validator.number" ],
                                "msrp_price" : [ "import.callback.validator.number" ],
                                "qty" : [ "import.callback.validator.number" ]
                        }
                ]
            }
        ]
      }
    }
   }
  }
 }
}
```

NOTE	you face a problem with a validator or don't need it, because your file doesn't contain the column, feel free to remove the corresponding callback from your overriding snippet.

Regex validator

The regex validator, used to validate the SKU, uses a regular expression to check the corresponding column's value.

For the SKU, the regular expression results in /.+/ what allows any character with at least one char.

If you want to enable digitals only, you can change the regular expression in the params/custom-validations array from /.+/ to /\+d/.

TIP	Our recommendation for a regex builder is regex 101, which will help you a lot to find the right expression for your needs.

Custom array validator

If you don't want to use a regular expression, but you have a list of allowed values, the custom array validator will be a good decision. By default, we use it to validate the product type.

To add your own product type, simply extend the list in the params/custom-validations array, e.g. to:

- "simple", "virtual", "configurable", "bundle", "grouped", "giftcard", "my_product_type". = Default column values

Sometimes it can be helpful if default values can be stored for columns not contained in the CSV file, e.g. if there are only simple products and the column product_type would be redundant.

From version 3.8.0 on, a snippet can be used to define default values for columns not contained in the CSV file.

The following fragment must have the following format:

```
{
  "default-values": {
    "catalog_product": {
      "my-column": "my-column-value"
    }
  }
}
```

CAUTION	Be aware that the default value will only be used, in case the CSV file does **not** contain the column.
	If the column is available, then always the column value will be used, even if the column is empty.

Media files

In most cases, the import of media files will be complicated, as the amount of data that has to be moved can quickly be large. **Pacemaker** offers an excellent solution to handle that comprehensively.

In general, that implies that **Pacemaker** doesn't take care, that the images will be moved from the directory, where they have been uploaded, to the magento pub/media/catalog/product directory.

Instead, the upload directory will be provided as a symlink to **Magento's** media directory.

Directory structure

The proven directory structure looks like the following one, whereas this one also contains the necessary directories for the 360° images, which may also be of interest in many cases.

```
/
 --var/
   --www/
     --sftp/
       --import/
         |--media/
         |   --images/
```

```
|      |--00490826/
|      |   |--general/
|      |   |   |--base_image.jpg
|      |   |   --small_image.jpg
|      |   |--additional/
|      |   |   |--additional_image_01.jpg
|      |   |   --additional_image_02.jpg
|      |    --360/
|      |       |--10584391-SPK-001--_01.JPG
|      |       |--10584391-SPK-001--_02.JPG
|      |       |--10584391-SPK-001--_03.JPG
|      |       |--10584391-SPK-001--_04.JPG
|      |       |--10584391-SPK-001--_05.JPG
|      |        --10584391-SPK-001--_06.JPG
|       --00490794/
|          |--general/
|          |   |--base_image.jpg
|          |    --small_image.jpg
|          |--additional/
|          |   |--additional_image_01.jpg
|          |    --additional_image_02.jpg
|           --360/
|              |--10584352-SAK-001-G_01.JPG
|              |--10584352-SAK-001-G_02.JPG
|              |--10584352-SAK-001-G_03.JPG
|              |--10584352-SAK-001-G_04.JPG
|              |--10584352-SAK-001-G_05.JPG
|               --10584352-SAK-001-G_06.JPG
 --data/
   |--category-import_20191204-140200_01.csv
   |--category-import_20191204-140200_01.ok
   |--product-import_20191204-140200_01.csv
    --product-import_20191204-140200_01.ok
```

WARNING

Magento requires exactly a two-level folder structure to make the image rendering work. After the **SKU**, exactly one additional folder that finally contains the image files, e.g. the folder general, **must** be available.

It's because **Magento** creates the image cache on-the-fly, based on the method \Magento\MediaStorage\App\Media::getOriginalImage() (if this has not already been done with the bin/magento catalog:image:resize command) which uses a regular expression that cuts off everything **before** those three levels.

It leads to the issue of the path to the original file under the directory pub/media/catalog/product will not be found, and instead of the image the placeholder will be rendered.

Path in CSV files

The path to the images in the CSV files has to be relative to the **Magento** pub/media directory. Additionally, they have to start with a slash.

For example, the files in the structure above **must** have the following format in the CSV file product-import_20191204-140200_01.csv

sku	base_image	small_image	is_360	images_360
00490826	/00490826/general/base_image.jpg	/00490826/general/small_image.jpg	Yes	/00490826/360
00490794	/00490794/general/base_image.jpg	/00490794/general/small_image.jpg	Yes	/00490794/360

Symlinks

When the directory structure has been created, symlinks can be created as well. It can be done by following command

```
ln -s <magento-install-dir>/var/pacemaker/import /var/www/sftp/import/data \
  && ln -s <magento-install-dir>/pub/media/catalog/product /var/www/sftp/import/media/images \
  && ln -s <magento-install-dir>/pub/media/magic360 /var/www/sftp/import/media/images
```

Import

Besides the automatic import by the appropriate pipeline, it is also possible to import the data manually.

For debugging purposes, this can be useful, as in some cases, error messages will be rendered on the CLI, and finding errors can then be more comfortable.

```
cd <magento-install-dir> \
  && rm var/pacemaker/import/*.csv \
  && cp <sample-data-dir>/* var/pacemaker/import \
  && vendor/bin/pacemaker import:products add-update \
    --serial=import \
    --source-dir=var/pacemaker \
    --target-dir=var/pacemaker \
    --magento-version=2.3.4 \
    --magento-edition=EE
```

Getting started

CAUTION	As of version 3.8.0, the structure of the setup has changed considerably, and the previous configuration files are no longer be used.
	To avoid complex adjustments of the configuration, version 3.8.0 merged the configuration for all entities into one but dedicated overwriting of individual settings is now possible, e.g. for the log level.

To install the **Magnento 2 Import Framework**, the Composer is necessary.

The framework itself is a set of components that provides import functionality for **Magento 2**. This repository, based on Symfony Console, uses the package Pacemaker Community Edition (CE) and provides a command-line tool with import functionality for **Magento 2** standard CSV files.

Before you start, keep in mind, that we've prepared a complete set of sample data. The sample data containing

- *attribute-sets*
- *attributes*
- *categories*
- *products*

are based on the original **Magento 2** sample data.

Please check out our sample data repository when you are going to test **Pacemaker Community Edition (CE)** or to find out, how the CSV files have to be structured.

| NOTE | For **Magento** versions 2.2.x and 2.3.x, up from the **Pacemaker Community Edition (CE)** version, 3.6.0 can be used. |

youtube video

File Structure

Attribute sets groups

Pacemaker Community Edition (CE) provides an attribute import and the appropriate command import:attributes:set therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated MSI import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.
- The default <PREFIX> is attribute-set-import, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01.
- As a result, the filename is look like attribute-set-import_20190608-114344_01.csv.
- Additionally, the appropriate .ok file is needed.

CAUTION	• There is a fixed structure for the bunch import, which must be followed.
	• The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv.
	• The name structure follows a predefined filename structure that is mandatory for the Bunch import.

Unique Identifier

- The unique identifier for the attribute set import is the attribute set name.
- The attribute set name is a mandatory field available in the column attribute_set_name on **every** row of the **CSV** file.

Columns

The CSV file with the attribute sets for the **Magento 2 CE/EE** consists of the following columns, whereas each attribute set can have multiple attribute groups:

Column Name	Type	Example	Description
attribute_set_name	varchar	Bag	The unique attribute set name.
based_on	varchar	Default	The name of the attribute set to copy the attributes from.
entity_type_code	varchar	catalog_product	The entity type code the attribute has to be bound to.
sort_order	integer	1	The sort order of the attribute set.
attribute_group_name	varchar	Product Details	The name of the attribute group.
attribute_group_code	varchar	product-details	The unique code of the attribute group.
attribute_group_tab_group_code	varchar	basic	The tag group code of the attribute group.
attribute_group_sort_order	int	10	The sort order for the attribute group in the backend/frontend.
default_id	int	1	The default ID of the attribute group.

Attributes

Pacemaker Community Edition (CE) provides an attribute import and the appropriate command import:attributes therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated **MSI** import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.
- The default <PREFIX> is attribute-import, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01.
- As a result, the filename is look like attribute-import_20190608-114344_01.csv.
- Additionally, the appropriate .ok file is needed.

CAUTION	• There is a fixed structure for the bunch import, which must be followed.
	• The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv.
	• The name structure follows a predefined filename structure that is mandatory for the Bunch import.

Unique identifier

- The unique identifier for the attribute import is the attribute code.
- The attribute code is a mandatory field that has been available in the column attribute_code on **every** row of the CSV file.

Columns

The CSV file with the attributes for the **Magento 2 CE/EE** consists of the following columns:

Column Name	Type	Example	Description
store_view_code	varchar	default	The specific store view(s) for attribute translations. If blank, the row provides the data for the admin store view.
attribute_set_name	varchar	Default	Assigns the attribute to a specific attribute set.
attribute_group_name	varchar	Product Details	Assigns the attribute to a specific attribute group.
entity_type_code	varchar	catalog_product	The entity type code, the attribute is bound to.
attribute_code	varchar	my_attribute	The unique attribue code
sort_order	int	1	The sort order for the attributes in the backend/frontend.
attribute_option_values	text	option-01, option-02	A comma (,) separated list with the attribute options and their values.
attribute_option_swatch	text	type=1,value=#000000|type=1,value=#d646d6	A pipe (|) separated list with the attribute option swatch configuration. If the attribute is swatch type (see additional_data column).
attribute_option_sort_order	text	1,2,3	The sort order for the attribute options in the backend/frontend.
attribute_model	varchar	Magento\Catalog\Model\ResourceModel\Eav\Attribute	The model to load the EAV data from (**must** be the **FQCN** of an available class).
backend_model	varchar	Magento\Eav\Model\Entity\Attribute\Backend\Datetime	The model to convert the user input (**must** be the **FQCN** of an available class).
backend_table	varchar	my_table_name	The table to store the user input.
frontend_model	varchar	Magento\Eav\Model\Entity\Attribute\Frontend\Datetime	The model to validate the user input (**must** be the **FQCN** of an available class).
frontend_input	varchar	boolean	The frontend input type (**must** be one of hidden, boolean, text, select, date, textarea, text, multiselect, price, weight, media_image, or gallery).
frontend_label	varchar	My Custom Label	The label for the attribute in the admin store.
frontend_class	varchar	hidden-for-virtual	A CSS class name.
source_model	varchar	Magento\Eav\Model\Entity\Attribute\Source\Table	The model to load the available values (**must** be the **FQCN** of an available class).
frontend_input_renderer	varchar	Magento\Rma\Block\Adminhtml\Product\Renderer	Frontend input renderer (**must** be the **FQCN** of an available class).
apply_to	varchar	simple, virtual, bundle, downloadable, configurable	A comma (,) separated list with product types; the attribute applies to

Column Name	Type	Example	Description
backend_type	varchar	int	The attributes backend type (**must** be one of text, varchar, decimal, int, datetime or static).
is_required	int	0	Whether or not the attribute value is mandatory.
is_user_defined	int	0	Whether or not, the attribute is user-defined.
default_value	text	2	The attributes default value (has to depend on the attribute configuration).
is_unique	int	0	Whether or not the attribute **must** contain a unique value in the configured context.
note	varchar	My custom note	A custom note for the attribute (will not be rendered anywhere).
is_global	int	1	Whether or not the attribute is global and can be used to create configurable products.
is_visible	int	1	Whether or not, the attribute is visible in the backend.
is_searchable	int	0	Whether or not, the attribute is searchable in frontend.
is_filterable	int	0	Whether or not, the attribute will be available in the filter navigation.
is_comparable	int	0	Whether or not the attribute can be used in product comparison.
is_visible_on_front	int	0	Whether or not the attribute is visible on the catalog pages in a storefront.
is_html_allowed_on_front	int	0	Whether or not the attribute value allows us to contain HTML code in a storefront.
is_used_for_price_rules	int	0	Whether or not the attribute can be used to create catalog price rules.
is_filterable_in_search	int	0	Whether or not the attribute values can be used to filter the search results.
used_in_product_listing	int	0	Whether or not the attribute values will be rendered in the catalog.
used_for_sort_by	int	0	Whether or not the attribute values can be used to sort the catalog.
is_visible_in_advanced_search	int	0	Whether or not, the attribute is visible in the advanced search
position	int	99	Position of the attribute in the layered navigation block.
is_wysiwyg_enabled	int	0	Whether or not, the WYSIWYG editor has to be enabled to edit the attribute's value.
is_used_for_promo_rules	int	0	Whether or not, the attribute can be used to create promo rules.

Column Name	Type	Example	Description
is_required_in_admin_store	int	0	Whether or not, the attribute **must** has a value in the admin store.
is_used_in_grid	int	0	Is used in the grid.
is_visible_in_grid	int	0	Is visible in the grid.
is_filterable_in_grid	int	0	Is filterable in the grid.
search_weight	float	1	Search weight.
additional_data	text	swatch_input_type=visual, update_product_preview_image=0, us`e_product_image_for_swatch=0`	A comma (,) separated list with additional attributes, which are necessary to create Visual Swatch or Text Swatch attributes.

Categories

Pacemaker Community Edition (CE) provides a category import and the appropriate command import:categories therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated **MSI** import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.
- The default <PREFIX> is category-import, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01.
- The results can be found in a filename like category-import_20190608-114344_01.csv.
- Additionally, the appropriate .ok file is needed.

CAUTION	- There is a fixed structure for the bunch import, which must be followed. - The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv. - The name structure follows a predefined filename structure that is mandatory for the Bunch import.

Unique identifier

For the product import, the **SKU** is the unique identifier of a product.

In the case of the categories, this is a bit more complicated as **Magento** uses a string of the imploded category IDs, separated by a slash (/), e.g. 1/2/120/1502.

It is not possible to e.g. the **PIM** system is aware of those **IDs**, the category instead needs a string with the admin store category names instead of the **ID** and expects it in the column *path*.

The results in values look like Default Category/Women/Tops/Hoodies & Sweatshirts. A common problem can be that one of those category names itself contains a slash (/). In that case, the category name must be enclosed with the default attaching

character, e.g., the double apostrophes (").

The additional attributes column of the product import is necessary to make sure that the path can be extracted, and the category will be added to the correct parent node.

The value for a category path that contains slashes must look like

```
"Default Category/Dachdecker- & Spenglerarbeiten/""Kehl-/Traufenanschlüsse &
-Belüftungen""/""Trauf-/Lüftungsrollen"""
```

Columns

The CSV file with the categories for the **Magento 2 CE/EE** consists of the following columns.

Column Name	Type	Mandatory	Example	Description
store_view_code	varchar	yes	e.g. en_US or read more about additional scopes	The specific store view(s) where the category is available. If blank, the category is available at the default store view.
attribute_set_code	varchar	yes	default	Assigns the product to a specific attribute set or product template, according to product type. Once the product is created, the attribute set cannot be changed.
path	varchar	yes	Default Category/MyCategory	The complete category path, including the root category.
name	varchar	yes	My Category	The category name appears the naviagtion, and is the name that customers use to identify the category.
is_active	int	yes	1	Enables or disables the category.
is_anchor	int	yes	1	If the category is anchor, the category's products as well as the products of the subcategories will be listed.
include_in_menu	int	yes	1	Specifies if the category will be included in the menu or not.
use_name_in_product_search	int	yes	1	If the category name is used for fulltext search on products
display_mode	varchar	yes	Products only	One of "Products only", "Static block only" or "Static block and products"
url_key	varchar	yes	my-category	The category's unique URL key
description	text	no	Some longer text here	The category description, that'll be rendered on the category page

Column Name	Type	Mandatory	Example	Description
image_path	varchar	no	images/categories/my-category.png	The absolute or relative path to a category image file
meta_title	varchar	no	My Category Name	The category's title that'll be rendered in the category page's <title> tag
meta_keywords	text	no	Category Name, Keyword 1, Keyword 2	The category's meta keywords that'll be rendered in the category page's <meta name="keywords"></meta>+++` tag
meta_description	text	no	A good Description with SEO relevant content	The category's meta description that'll be rendered in the category page's +<meta name="description">`</meta> tag
landing_page	int	no	2	The ID of a CMS block that has to be rendered in the category page
position	int	no	10	The category's position in the navigation
custom_design	varchar	no	Magneto Blank	The custom design name used to display the catgory
custom_design_from	datetime	no	10/24/16, 12:36 PM	The start date for the scheduled design update
custom_design_to	datetime	no	10/24/16, 12:36 PM	The end date for the scheduled design update
page_layout	varchar	no	1 column	A custom page layout used to display the category, one of 1 column, 2 columns with left bar, 2 columns with right bar, 3 columns, Empty
custom_layout_update	text	no	<referenceContainer name="catalog.leftnav" remove="true"> </referenceContainer>	A custom page layout update in XML format
available_sort_by	text	no	Position,Name	The comma separated product list sortings for the catgory
default_sort_by	varchar	no	Position	The default product list sorting for the category
custom_apply_to_products	int	no	1	If set to 1, the design will also be applied to the products listed in the category
custom_use_parent_settings	int	no	1	Overrides the custom design settings with the default one's

Column Name	Type	Mandatory	Example	Description
filter_price_range	decimal	no	100.00	The layered navigation price steps
created_at	varchar	no	10/24/16, 12:36 PM	The category's creation date
updated_at	varchar	no	10/24/16, 12:36 PM	The date when the category has been updated
additional_attributes	text	no	custom_attribute_01=a-value,custom_attribute_02=value-01|value-02	A comma separated list with additional attributes (the attributes **must** already be available)

Products

Pacemaker Community Edition (CE) provides a product import and the appropriate command import:products therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated **MSI** import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.
- The default <PREFIX> is product-import, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01.
- The results in a filename like product-import_20190608-114344_01.csv.
- Additionally, the appropriate .ok file is needed.

CAUTION	- There is a fixed structure for the bunch import, which must be followed. - The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv. - The name structure follows a predefined filename structure that is mandatory for the Bunch import.

Unique identifier

- The unique identifier for the product import is the **SKU**.
- The **SKU** is a mandatory field that has to be available in the column sku on **every** row of the **CSV** file.

Compatibility & extensions

The structure for the Product import is nearly **100 % compatible** with the **Magento 2 CSV** structure.

Please check out the **Magento 2** documentation for a more detailed description of the **CSV** file structure.

Especially for the product import, **Magento** extends the default **CSV** format for some individual cases called complex data, which provides some data serialization to extend the default CSV format.

Magento describes this on a dedicated page on their website. Besides the topics related to that page by **Magento** itself, there is a column with additional attributes that also uses a sophisticated data format like provided additional attributes.

In addition to the **Magento 2 CSV** structure, it is possible to add additional columns to allow tier prices and **MSI** stock data to be part of the product import.

Both columns contain serialized data like the provided additional attributes column does.

During the product import, **Pacemaker Community Edition (CE)** extracts the data from the columns and creates separate CSV files.

These CSV files are 1:1 the format described for MSI and Tier Prices import in the corresponding sections.

After the products are imported, **Pacemaker Community Edition (CE)** imports the tier prices as well as the **MSI** data. = Additional attributes

The column additional_attributes allows import values for all attributes that are not part of the default **CSV** format, including the user-defined columns.

CAUTION	Remember **always** that for an attribute of type [Yes/No|Select|Multi-select|Dropdown|Text field|Visual field] the value is **always** the admin store's value and **never** one of the scopes. The rendered value **is** the one by the selected store view, either in the back or front; it is also the case for all other attribute types.

Enclosing, Escaping, and Delimiter

As the default multiple field separator, which separates the key-value pairs with the option values to be imported, is a comma (,) it is necessary to enclose the **complete** value of the column with double apostrophes (").

Additionally, if only **one** of the attribute option values also contains a comma, e. g. the value Laptop Sleeve, 15 inches, the **complete** key-value pair additionally has to be enclosed with double apostrophes (").

It leads to unusual constellations, as those double apostrophes (") has to be escaped with double apostrophes (") again see

sku	…	additional_attributes	…
MB-2401		"color=black,""features=Audio Pocket|Laptop Sleeve, 15 inches"""	

In case the values of additional attributes contain a double apostrophe ("), e. g. like Laptop "Sleeve" those have to be escaped with a backslash or a double apostrophe (") itself like

sku	…	additional_attributes	…
MB-2401		"color=black,features=Audio Pocket|Laptop ""Sleeve"""	

It will also be possible that the values of additional attributes contain double apostrophes (") as well as commas (,). Those have to be enclosed and escaped like

sku	…	additional_attributes	…
MB-2401		"color=black,""features=Audio Pocket|Laptop ""Sleeve"", 15 inches"""	

CAUTION	Please keep in mind that when an escape character, which defaults to backslash (\), is used to escape the delimiter, e. g. Laptop \"Sleeve\" the delimiter will **not** be removed automatically, instead it will be stored in the database.

Multiselect Attributes

In the case of Multiselect Attributes, more than one value for an attribute is specified. Those values have to be separated by character, which by default is the pipe (|).

For example, if the Multiselect Attribute activity should have the values gym and hiking, trails selected, the column additional_attributes **must** look like

sku	…	additional_attributes	…
MB-2401		"""activity=gym|hiking, trails"",""features=Audio Pocket|Laptop Sleeve, 15 inches"""	

CAUTION The Multiselect Attributes are the only values that use the multiple value delimiter.

Configuration

Beside the general **CSV** configuration on subject level **Pacemaker Community Edition (CE)** allows to override the default values, which are a comma (,) for the multiple fields and a pipe (|) for the numerous value separator.

In contrast to the general **CSV** configuration, this has to be done on the global level and is therefore valid for all operations, subjects, and observers.

It is possible to override the default configuration by adding either one of or both multiple-field-delimiter and multiple-value-delimiter to the configuration file like

```
{
  "magento-edition": "CE",
  "magento-version": "2.3.2",
  "operation-name" : "add-update",
  "installation-dir" : "/var/www/magento",
  "multiple-field-delimiter" : ",",
  "multiple-value-delimiter" : "|",
  "databases" : [ ... ],
  "loggers" : [ ... ],
  "operations" : { ... }
}
```

Product scopes

The import functionality provides a possibility to import values on different scopes.

Therefore, all scope specific values have to be added on a separate row for each scope, which represents a store view, like

sku	store_view_code	attribute_set_code	product_type	name	description	url_key	…
MB-2401		Default	simple	Duffle Bag	This the default description.	duffle-bag	
MB-2401	de_DE	Default		Reisetasche	Das ist die deutsche Beschreibung.	reisetasche	

sku	store_view_code	attribute_set_code	product_type	name	description	url_key	…
MB-2401	fr_FR	Default		Sac Marin	C'est la description par défaut.	sac-marin	
MB-2401	es_ES	Default		Bolsa de Lona	Esta es la descripción por defecto.	bolsa-de-lona	

CAUTION	The column store_view_code **must** contains the appropriate **Magento** store view code and the column attribute_set_code **must** have a valid attribute set code.

Translateable Values

CAUTION	Not all attributes can have different values per scope.

- In general, all product attributes that have been created with scope Store View can be translated.
- To determine which attributes have the scope Store View open the Backend and go to the overview with the product attributes by clicking on the navigation path menu:Stores[Attributes > Products].

Select the filter Store View and press btn:[enter].

The result should look like

Most of the default attributes have a dedicated column in the **CSV** file and can be translated by adding the appropriate value to a row with the proper store view scope. But **all** attributes that have been user-defined, either in the backend or by a developer, do not.

To find out which columns are user-defined, use your preferred SQL editor and enter a **SQL** statement like
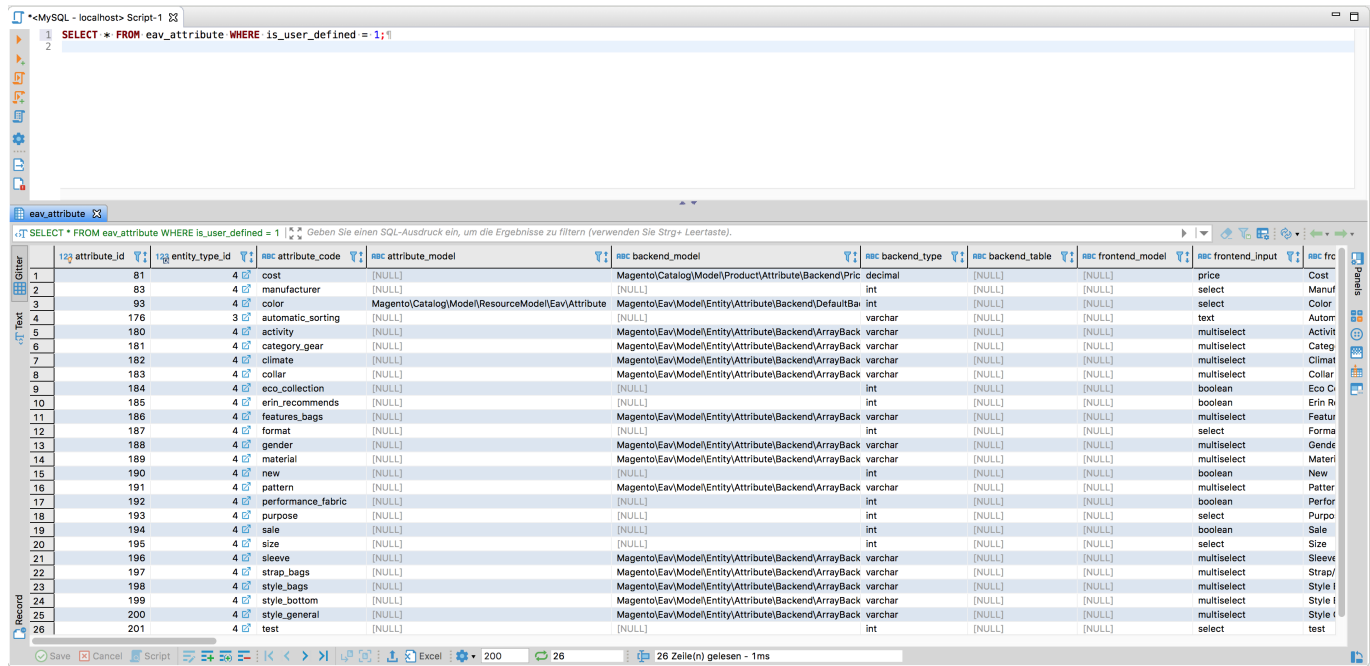
It should give you a list of all user-defined attributes.

All user-defined attributes can be imported by adding the appropriate key-value pair to the additional_attributes column to make the import as flexible as possible.

The example below shows a part of a **CSV** file with scope specific values for the columns *name*, *description*, and *test*

sku	store_view_code	name	description	additional_attributes	…
MB-2401		Duffle Bag	This the default description.	"test=english,activity=Gym|Hiking"	
MB-2401	de_DE	Reisetasche	Das ist die deutsche Beschreibung.	"test=german"	
MB-2401	fr_FR	Sac Marin	C'est la description par défaut.	"test=french"	
MB-2401	es_ES	Bolsa de Lona	Esta es la descripción por defecto.	"test=spain"	

For more details about Additional Attributes read the dedicated section for additional attributes

Product links

Pacemaker Community Edition (CE) provides the functionality to import product links as well as their positions.

- **related**, **upsell**, and **crosssell** links are supported

As **Pacemaker Community Edition (CE)** also supports the import of **grouped** products, which are nothing else than the additional link type super, it is possible to import the products linked to a **grouped** product.

Only a comma (,) separated list of the **linked** products has to be specified in the appropriate column to import those link types.

- column related_skus for related products that will be rendered on the product detail page
- column upsell_skus for upsell products that will be rendered on the product detail page
- column crosssell_skus for crosssell products that will be rendered on the shopping cart
- column associated_skus for products that will be part of a grouped product, which will also be rendered on the product detail page

For example, the columns to import those link types may look like

sku	…	related_skus	upsell_skus	crosssell_skus	associated_skus	…
24-MB01		24-MB02,24-MB03	24-WG03,24-WG04	24-SB01,24-SB02	24-AS01,24-AS02	

Link Positions

Besides the **SKUs** of the linked product itself, it is also possible to specify the position of the related products

sku	…	related_position	upsell_position	crosssell_position	associated_position	…
24-MB01		1,2	2,1		1,2	

The position also has to be a comma (,) separated list, but it contains numbers instead of the **SKUs**.

The numbers are the position where it will render the linked product in the **GUI**.

CAUTION	Please be aware that the positions of the columns are **not** mandatory, and the columns can be empty. If so, the **Pacemaker Community Edition (CE)** creates the position itself, based on the given **SKUs** order. If the positioning will be changed in the **Magento** backend, they will be **overritten** within the next import process

Magento 2 Community Edition (CE) < 2.1.6

Magento 2 Community Edition (CE) supports positioning for product links, as well as **Magento 2 Enterprise Edition (EE)**.

By default, up to version 2.1.6, importing product positions is **not** possible in the **Magento 2 Community Edition (CE)**, because the database of the **Magento 2 Community Edition (CE)** lack's of missing rows in the catalog_product_link_attribute table.

In case the rows are not available, the import will ignore positions defined in the **CSV** file's columns.

- related_position
- crosssell_position

- upsell_position

To enable importing positions, add the following rows the **Magento 2 Community Edition (CE)** database.

```
INSERT INTO
        `catalog_product_link_attribute` (
            `link_type_id`,
            `product_link_attribute_code`,
            `data_type`
        )
    VALUES
        (1,'position','int'),
        (4,'position','int'),
        (5,'position','int');
```

CAUTION	Make sure that the values are **not** already available before adding them.

MSI

Pacemaker Community Edition (CE) provides a dedicated **MSI** import and the appropriate command import:products:inventory:msi therefore.

TIP	You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated **MSI** import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv

- The default <PREFIX> is product-import-inventory-msi, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01

- As a result, the filename is look like product-import-inventory-msi_20190608-114344_01.csv

- Additionally the appropriate .ok file is needed

CAUTION	- There is a fixed structure for the bunch import, which must be followed.
	- The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv.
	- The name structure follows a predefined filename structure that is mandatory for the Bunch import.

The **CSV** file with the **Magento 2 CE/EE** attributes consists of the following columns:

Column Name	Type	Example	Description
sku	varchar	24-MB01	The product SKU to import the inventory source item for
source_code	varchar	default	The code for the source of the products.
status	integer	1	The stock status of the product for this source.
quantity	decimal	98.0	The amount of the product for this source.

Use cases

Import without MSI:

- **CSV** requires the following columns: sku, qty, is_in_stock
- Inventory Import Step must be called with the command import:products:inventory
- The default file name is: product-import-inventory_<xx>.csv

If **MSI** should be imported with the full product import, read the appropriate documentation how this can be archived.

Import with MSI:

CSV requires the following columns	sku, quantity, status, source_code
The inventory import step must be called with the command	import:products:inventory:msi
The default file name is	product-import-inventory-msi_<xx>.csv

To use MSI with the default product import, two steps are necessary.

Step 1: Add Additional Column

Add the column inventory_source_items also to import MSI stock data with the product import.

For example, the column **must** contain the data in the following structure, e. g.

```
source_code=default,quantity=10.0,status=1|source_code=inventory-01,quantity=5.0,status=1
```

As for the tier prices, the column with the **MSI** inventory source items supports the same format. Our sample data comes with an example of how the file should look.

Step 2: Extend Configuration

The second step is to add the subject that processes the **MSI** to your configuration file.

An example configuration file for the **Pacemaker Community Edition (CE)** is part of the **Pacemaker Community Edition (CE)** command-line tool.

The configuration for the appropriate operations has to be extended with

- The **subject** import_product_msi.subject.bunch with the **observer** import_product_msi.observer.inventory.source.item.update for the **add-update** and the **observer** import_product_msi.observer.clear.inventory.source.item and import_product_msi.observer.inventory.source.item for the replace operation
- The **observer** import_product_msi.observer.product.source.item that has to be added **after** the first **observer** of the first import_product.subject.bunch subject
- The **observers** import_product_msi.observer.product.source.item and import_product.observer.composite.base.delete **before** the import_product.observer.composite.base.delete of the **subject** import_product.subject.bunch of the delete operation
- The **subject** import_product_msi.subject.bunch with the **observer** import_product_msi.observer.clear.inventory.source.item of the delete operation

For the delete operation, the configuration has to look like following code

```json
{
 ...,
 "operations": [
   {
      "name" : "delete",
      "plugins" : [,
        {
          "id": "import.plugin.subject",
          "subjects": [
            ...,
            {
              "id": "import_product.subject.bunch",
              ...,
              "observers": [
                {
                  "import": [
                    "import_product.observer.last.entity.id",
                    "import_product_msi.observer.product.source.item",
                    "import_product.observer.composite.base.delete"
                  ]
                }
              ]
            },
            ...,
            {
              "id": "import_product_msi.subject.bunch",
              "identifier": "files",
              "file-resolver": {
                "prefix": "inventory-msi"
              },
              "observers": [
                {
                  "import": [
                    "import_product_msi.observer.clear.inventory.source.item"
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
    ...,
   ]
 }
}
```

For the replace operation, the configuration has to look like

```
{
  ...,
  "operations": [
      {
      "name" : "replace",
      "plugins" : [
        ...,
        {
          "id": "import.plugin.subject",
          "subjects": [
            {
              "id": "import_product.subject.bunch",
              ...,
              "observers": [
                {
                  "import": [
                    "import_product.observer.composite.base.replace",
                    "import_product_msi.observer.product.source.item"
                  ]
                }
              ]
            },
            ...,
            {
              "id": "import_product_msi.subject.bunch",
              "identifier": "files",
              "file-resolver": {
                "prefix": "inventory-msi"
              },
              "observers": [
                {
                  "import": [
                    "import_product_msi.observer.clear.inventory.source.item",
                    "import_product_msi.observer.inventory.source.item"
                  ]
                }
              ]
            }
          ]
        }
        ...,
      ]
    }
  ]
}
```

For the add-update operation, the configuration has to look like

```
{
```

```
...,
"operations": [
  {
      "name" : "add-update",
      "plugins" : [,
        {
          "id": "import.plugin.subject",
          "subjects": [
            ...,
            {
              "id": "import_product.subject.bunch",
              ...,
              "observers": [
                {
                  "import": [
                    "import_product.observer.composite.base.add_update",
                    "import_product_msi.observer.product.source.item"
                  ]
                }
              ]
            },
            ...,
            {
              "id": "import_product_msi.subject.bunch",
              "identifier": "files",
              "file-resolver": {
                "prefix": "inventory-msi"
              },
              "observers": [
                {
                  "import": [
                    "import_product_msi.observer.inventory.source.item.update"
                  ]
                }
              ]
            }
          ]
        }
      ...,
      ]
    }
  ]
}
```

Tier prices

- Tier price import only

- Multi Websites/Stores import of products including tier prices

- Extend Configuration

Tier price import only

Pacemaker Community Edition (CE) provides a dedicated tier price import and the appropriate command import:products:price:tier therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated tier price import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.
- The default <PREFIX> is product-import-tier-price, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01.
- As a result, the filename is look like product-import-tier price_20190608-114344_01.csv.
- Additionally, the appropriate .ok file is needed.

CAUTION	• There is a fixed structure for the bunch import, which must be followed. • The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv. • The name structure follows a predefined filename structure that is mandatory for the Bunch import.

The **CSV** file with the attributes for the **Magento 2 CE/EE** consists of the following columns:

Column Name	Type	Example	Description
sku	varchar	24-MB01	The product **SKU** to import the tier price for the required Product
tier_price_website	varchar	All Websites	The **website code** to import the tier price for • All Websites as default for all websites or • <website_code>
tier_price_customer_group	varchar	ALL GROUPS	The customer group for which the tiered price get imported, or 'ALL GROUPS' for all customer groups
tier_price_qty	decimal	20.0	The quantity the tier price is valid from
tier_price	decimal	98.0	The tier price itself
tier_price_value_type	varchar	Fixed	The value type can either be • Fixed|Discount

If the tier prices are imported with the full product import, read the description below.

Multi Websites/Stores import of products including tier prices

To perform a successful multi websites/stores product import, the field tier_prices must be added to the product import data **CSV** with the data field structure description as followed

- **Admin row:**
    - **Add additional column**
        - The first step is to add the column tier_prices to the **CSV** file with the product data.

⬜ When importing tier_prices to multiple store views, the tier_prices field must be populated with the following given format and separated by a pipe (|)

⬜ No list element escaping required

⬜ Example store view 1: qty=5,price=51.000000,value_type=fixed,website=All Websites,customer_group=ALL GROUPS

⬜ Example store view 2: qty=15,price=3.00,value_type=fixed,website=additional_base,customer_group=General

⬜ Example store view 3: qty=10,price=47.750000,value_type=discount,website=base,customer_group=ALL GROUPS

Example tier_prices field result. The data sets are seperated by pipes (|)

```
qty=5,price=51.000000,value_type=fixed,website=All Websites,customer_group=ALL
GROUPS|qty=15,price=3.00,value_type=fixed,website=additional_base,customer_group=General|qty
=10,price=47.750000,value_type=discount,website=base,customer_group=ALL GROUPS
```

- There is no limitation in how many rows of tier prices the column contains. THe pipe (|) will separate each row with tier prices, whereas each column of a row including the attribute_code to value pairs will be separated by the common comma (,).

- The attribute_code to value pairs itself will use the = char for separation.

- Our sample data comes with an example of how the file should look.

*Table 1. Field **tier_prices** value structure*

Attribute	Example	Description
qty	5	The number entered determines when a tier_price takes effect
price	51.000000	The value entered under price takes effect when the min. count specified under qty is reached or greater
value_type	fixed	The value type is defind by a choice of 2 Values * [fixed|discount]
website	All Websites	The website value is defind by the unique Website Code [website_code]. The Value All Websites is predefined by **Magento**. *Figure 1. Magento Backend > Stores > All Stores ⬜ add the **Web Site Code** to the attribute website=additional_base*
customer_group	ALL GROUPS	The Customer Group defines the associated User Group * **[ALL GROUPS|General|Retailer|NOT LOGGED IN|Wholesale]**
CAUTION		- The actual website_code for each additional store view, the store view code is entered at the argument website=<WEBSITE CODE> - No entry in store_view_code field

- **Store row**

  - In the field store_view_code for each additional store view, except the admin row, the field must be filled with the respective **store view code** (default,de_de, en_us,…)

  - No entry in the field tier_prices is required for the following **store code** related additional product values

    if you want to track the import more carefully in the directory importexport, feel free to use following additional params to import products

    - --archive-artefacts=false --clear-artefacts=false

Web Site	Store	Store View
Additional Website (Code: additional_base)	Additional Store (Code: additional_store)	Additional Store View (Code: additional_base_view)
Main Website (Code: base)	Main Website Store (Code: main_website_store)	Default Store View (Code: default)
Main Website (Code: base)	Main Website Store (Code: main_website_store)	Germany (Code: de_de)
Main Website (Code: base)	Main Website Store (Code: main_website_store)	United States (Code: en_us)

*Figure 2. Add the **Store View Code** to the field store_view_code in your import **CSV** file (Example: de_de, en_us, …..)*

```
vendor/bin/import-simple import:products --archive-artefacts=false --clear
-artefacts=false
```

TIP   the additional arguments make sure that the imported files and logs are not removed after a import. You will find all processed files after a import in a newly created folder named as timestamp
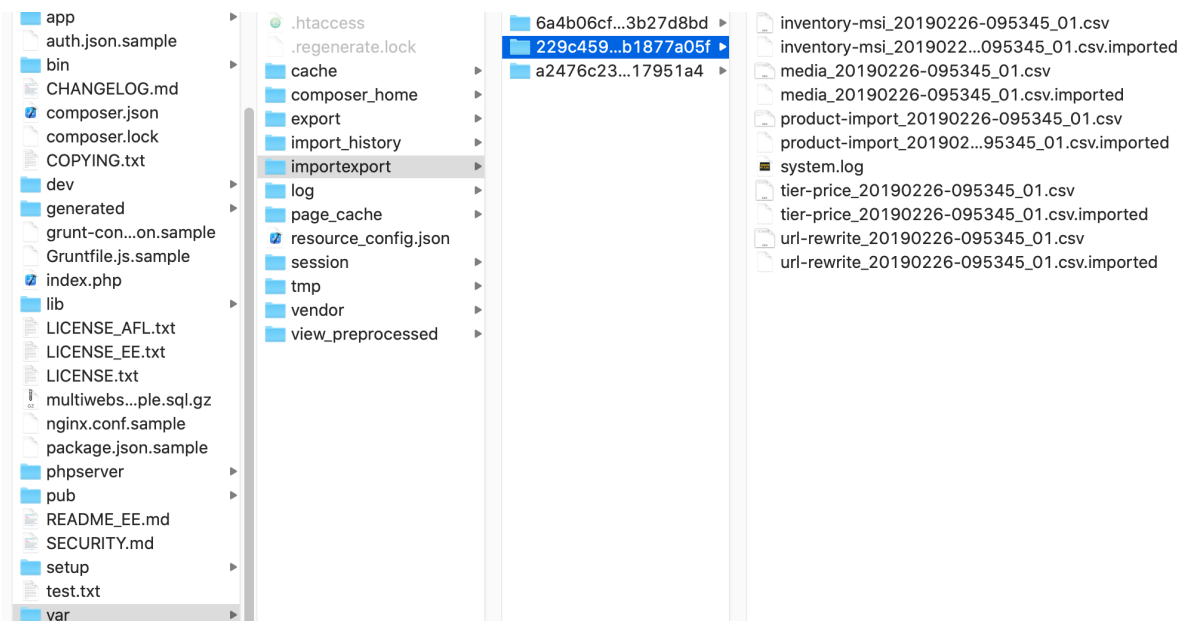
Figure 3. Using the parameters --archive-artefacts=false --clear-artefacts=false a new directory will be created with the current timestamp as folder name

	sku	store_view_code	attribute_set_code	product_type	cate
1	sku	store_view_code	attribute_set_code	product_type	cate
2	24-UB02	*<null>*	Bag	simple	Defa
3	24-UB02	default	Bag	simple	*<nul*
4	24-UB02	de_de	Bag	simple	*<nul*
5	24-UB02	en_us	Bag	simple	*<nul*
6	24-UB02	additional_base_view	Bag	simple	*<nul*

Figure 4. Add the **Store View Code** to the field store_view_code in your import **CSV** file (Example: de_de, en_us, …..)

configurable…	tier_prices
<null>	qty=5,price=51.000000,value_type=fixed,website=All Websites,customer_group=AL…
<null>	<null>
<null>	*<null>*
<null>	*<null>*
<null>	*<null>*

Figure 5. Add the **Multi-Store Attributes** to the field tier_prices as described in the **admin row**

TIP	For testing purpose only: If you want to check the result of a multi-store product import in the **Magento** backend, please make sure in advance that the following default options are set: • menu:Stores[Configuration > Catalog /Catalog > Price [Catalog Price Scope ▯ Website]] • menu:Stores[Configuration > General > Web > Url Options [Add Store Code to Urls ▯ Yes]]

Extend Configuration

The second step is to add the listener, the subjects, and the observers that process the tier prices to your configuration file.

An example configuration file for the **Pacemaker Community Edition (CE)** is part of the **Pacemaker Community Edition (CE)** command-line tool.

The configuration for the appropriate operations has to be extended with

- The subject import_product_tier_price.subject.tier_price with the observer import_product_tier_price.observer.tier_price.update for the add-update and the observer import_product_tier_price.observer.tier_price for the replace operation

- The observer import_product_tier_price.observer.product.tier_price has to be added **after** the first observer of the first import_product.subject.bunch subject

- A listener import_product_tier_price.listener.delete.obsolete.tier_prices for the event plugin.process.success on subject level (only for add-update operation)

- Param clean-up-tier-prices on the subject level either with the value **true** or **false**, which decides whether or not tier-prices should be cleaned-up (only for add-update operation)

CAUTION	If the clean-up functionality is activated with the param clean-up-tier-prices set to true, all tier prices that are **not** anymore part of the **SKUs** in the **CSV** file will be **removed**.

For the replace operation, the configuration has to look like

```json
{
  ...,
  "operations": [
      {
        "name" : "replace",
        "plugins" : [
          ...,
          {
            "id": "import.plugin.subject",
            "subjects": [
              {
                "id": "import_product.subject.bunch",
                ...,
                "observers": [
                  {
                    "import": [
                      "import_product.observer.composite.base.replace",
                      "import_product_tier_price.observer.product.tier_price"
                    ]
                  }
                ]
              },
              ...,
              {
                "id": "import_product_tier_price.subject.tier_price",
                "identifier": "files",
                "file-resolver": {
                  "prefix": "tier-price"
                },
                "observers": [
                  {
                    "import": [
                      "import_product_tier_price.observer.tier_price"
                    ]
                  }
                ]
              }
            ]
          }
          ...,
        ]
      }
    ]
}
```

For the add-update operation, the configuration has to look like

```json
{
  ...,
```

```
"operations": [
  {
    "name" : "add-update",
    "plugins" : [,
      {
        "id": "import.plugin.subject",
        "listeners" : [
          {
            "plugin.process.success" : [
              "import_product_tier_price.listener.delete.obsolete.tier_prices"
            ]
          }
        ],
        "params" : [
          {
            "clean-up-tier-prices" : true
          }
        ],
        "subjects": [
          ...,
          {
            "id": "import_product.subject.bunch",
            ...,
            "observers": [
              {
                "import": [
                  "import_product.observer.composite.base.add_update",
                  "import_product_tier_price.observer.product.tier_price"
                ]
              }
            ]
          },
          ...,
          {
            "id": "import_product_tier_price.subject.tier_price",
            "file-resolver": {
              "prefix": "tier-price"
            },
            "observers": [
              {
                "import": [
                  "import_product_tier_price.observer.tier_price.update"
                ]
              }
            ]
          }
        ]
      }
    ...,
```

```
        ]
    }
  ]
```

For both operations, it has to be be addeded after the import_product_url_rewrite.subject.url.rewrite. The delete operation configuration does not need any customizations as the tier prices will be cleaned up by their foreign keys. = Customer + Customer Address Import

Pacemaker Community Edition (CE) comes with a customer and customer address import and the appropriate commands import:customers and import:customers:address therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated **MSI** import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.

- The default <PREFIX> for the customer import is customer-import and for the customer address import customer-address-import.

- The <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a incremental number with two digits starting with 01.

- As a result, the filename is look like customer-import_20190608-114344_01.csv for customers and customer-address-import_20190608-114344_01.csv for customer addresses.

- Additionally the appropriate .ok file is needed.

CAUTION	• There is a fixed structure for the bunch import, which must be followed. • The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv. • The name structure follows a predefined filename structure that is mandatory for the Bunch import.

The customer and customer address import structure is **100 % compatible** with the **Magento 2 CSV** structure.

Read the appropriate **Magento 2** documentation for a more detailed description of the **CSV** file structure. = Product url rewrites

Pacemaker Community Edition (CE) comes with a dedicated product URL rewrite import and the appropriate command import:products:url therefore.

You can find more information about how to invoke the command in the usage section.

- The filename for the dedicated product URL rewrite import **must** match the pattern <PREFIX>_<FILENAME>_<COUNTER>.csv.

- The default <PREFIX> is url-rewrite, the <FILENAME> is a combination of date and time like 20190608-114344, and the <COUNTER> is a consecutive number with two digits starting with 01.

- As a result, the filename is look like url-rewrite_20190608-114344_01.csv.

- Additionally the appropriate .ok file is needed.

CAUTION	• There is a fixed structure for the bunch import, which must be followed. • The three parts to be specified are each separated by an underscore (_) and result in a filename like <PREFIX>_<FILENAME>_<COUNTER>.csv. • The name structure follows a predefined filename structure that is mandatory for the Bunch import.

The **CSV** file with the attributes for the **Magento 2 CE/EE** consists of the following columns:

Column Name	Type	Example	Description
sku	varchar	24-MB01	The product SKU to import the tier price for
store_view_ code	varchar	default	The specific store view(s) for attribute translations. If blank, the row provides the data for the admin store view.
categories	varchar	Default Category/Gear,Default Category/Gear/Bags	The comma (,) separated list with related categories, URL rewrites has to be created for
product_we bsites	varchar	base	The comma (,) separated list with website codes, URL rewrites has to be created for
visibility	varchar	Catalog, Search	The products visibility (URL rewrites will **not** be created for value Not Visible Individually)
url_key	varchar	joust-duffle-bag	The **URL** key used to create the URL rewrite

By default, the product **URL** rewrites will be imported with the full product import.

Read the appropriate documentation for a more detailed description.

Components & Concept

Components

- **Pacemaker Community Edition (CE)** is a set of components that enables developers to build import services for **Magento 2**.
- **Pacemaker Community Edition (CE)** comes with a group of core components that provides the functionality to import products, categories, and attributes.
- With the **Pacemaker Community Edition (CE)** Console Tool, we also provide a reference implementation for applications that uses **Pacemaker Community Edition (CE)**. = Core components

This page lists the main components that provide **Magento 2** import core functionality to import products, categories, and attributes.

General (Independent from Edition)

- import - A core library supporting generic **Magento 2** import functionality
- import-cli - **CLI** command implementation used by the implementing **CLI** applications
- import-app-simple - Generic application implementation that uses **Symfony Console + DI** as well as **Pacemaker Community Edition (CE)** to provide **Magento 2 CE/EE** import functionality
- import-configuration-jms - A JMS based **Pacemaker Community Edition (CE)** configuration implementation

Components for Pacemaker Community Edition (CE)

These are the **Pacemaker Community Edition (CE)** core components for the **Magento 2 Community Edition (CE)**.

- import-product - Provides product import functionality
- import-product-url-rewrite - Provides **Product URL Rewrite** import functionality
- import-product-bundle - Provides **Bundle Product** Import functionality
- import-product-link - Provides *Product Relatio*n import functionality
- import-product-media - Provides **Product Image Import** functionality
- import-product-variant - Provides **Configurable Product Import** functionality
- import-product-grouped - Provides **Grouped Product Import** functionality
- import-product-msi - Provides **Product MSI Import** functionality
- import-product-tier-price - Provides **Product Tier Price Import** functionality
- import-category - Provides **Category Import** functionality
- import-attribute - Provides **Attribute Import** functionality
- import-customer - Provides **Customer Import** functionality
- import-customer-address - Provides **Customer Address Import** functionality
- import-converter - Provides **generic converter** functionality
- import-converter-product-category - Provides the functionality to convert a **CSV** file with category from a **CSV** file with products
- import-converter-product-attribute - Provides the functionality to convert a **CSV** file with attribute option values from a **CSV** file with products.

TIP	Like import-attribute, several components will also work with the **Pacemaker Enterprise Edition (EE)**, so there is not separate implementation.

Components for Pacemaker Enterprise Edition (EE)

These are the **Pacemaker Community Edition (CE)** core components for the **Magento 2 Enterprise Edition (EE)**.

- import-ee - Provides Core Import functionality for **Magento 2 Enterprise Edition (EE)**
- import-product-ee - Provides Product Import functionality for **Magento 2 Enterprise Edition (EE)**
- import-product-bundle-ee - Provides Bundle Product Import functionality for **Magento 2 Enterprise Edition (EE)**
- import-product-link-ee - Provides Product Import Relation functionality for **Magento 2 Enterprise Edition (EE)**
- import-product-media-ee - Provides Product Import Image functionality for **Magento 2 Enterprise Edition (EE)**
- import-product-variant-ee - Provides Configurable Product Import functionality for **Magento 2 Enterprise Edition (EE)**
- import-product-grouped-ee - Provides Grouped Product Import functionality for **Magento 2 Enterprise Edition (EE)**
- import-category-ee - Provides Category Import functionality for **Magento 2 Enterprise Edition (EE)**
- import-converter-ee - Provides generic **Magento 2 Enterprise Edition (EE)** converter functionality

3rd party components

Whenever possible, we try to implement components for as many 3rd party components as possible.

- import-product-magic360 - Provides import functionality for the Magictoolbox Magic360 Extension = Applications

Applications are **Pacemaker Community Edition (CE)** implementations to make the import functionality available, e.g. on **CLI**.

- import-cli-simple - This is the **Pacemaker Community Edition (CE) Console Tool** implementation that uses **Pacemaker Community Edition (CE)** to provide **Magento 2 CE/EE** import functionality = Framework

The primary purpose of the **Pacemaker Community Edition (CE)**, if you will not use it indirectly with the **Pacemaker Community Edition (CE) Console Tool**, is to support you to build your own **Magento 2** import service.

Therefore, we chose a component-based architecture.

Suppose you want to implement your custom component to import another **Magento 2** entity, e.g. customers, you can and should follow these guidelines. = Dependency injection

The **Pacemaker Community Edition (CE)** components do not care about **DI**, but they can be tied together using **DI**.

For the **Pacemaker Community Edition (CE) Console Tool**, that we've implemented as a reference application, we used the Symfony DI Container to compose the application.

To make a start as simple as possible, we recommend using **Symfony** and **Symfony DI** when starting writing your component or application.

Therefore each core library provides the necessary **Symfony DI** configuration files in the directory symfony/Resources/config/services.xml.

To get an impression on how **Symfony DI** can be used to composer your application, look at the reference application.

Pacemaker Community Edition (CE) Console Tool, which parses the library files on start-up, depending on the used **Magento Edition**, the appropriate load, initialize, and inject the necessary classes.

> **NOTE** | For configuration the **Symfony** service **IDs** will be used instead of the real class names.

Component structure

In general, there is no restriction for the folder structure of your component.

As **Pacemaker Community Edition (CE)** components usually will be installed by **Composer**, at least your component structure should be **PSR-0** or **PSR-4** compatible.

As usual, every component will be delivered with its configuration (the necessary files for **DI**, for example); the structure should also support the separation of source code and configuration files.

Pretending you will start to implement your first component, the recommended component structure looks like this.

```
composer.json
    src/
        Actions/
            Processors/
        Assemblers
        Callbacks
        Observers
        Repositories
            CacheWarmers/
        Services
```

```
        Subjects
        Utils
    tests/
    symfony/
            DependencyInjection/
            Resources/
                config/
                    service.xml
            Utils/
                DependencyInjectionKeys.php
        ImportBundle.php
```

The registry

As **Pacemaker Community Edition (CE)** is built to run in single and multithreaded/multiprocess environments, it needs a possibility to share the data between the components in a single process and between threads and processes.

Whenever data has to be shared, the registry will be the right place.

The default implementation only works in a single-threaded environment, where it does not need to take care of concurrency issues that will occur in multithreaded or multiprocess ones.

Further Thoughts regard multithreading and multiprocessing

As the registry implementation comes with the core components only works in a single process, probably the choice in most use cases. Depending on the environment **Pacemaker Community Edition (CE)** will be used, it will be necessary to implement a registry processor implementation that fits these environments' needs.

To give you an example of what kind of changes will be necessary for a multithreaded environment, e.g. like appserver.io we will implement the method mergeAttributesRecursive().

A thread safe version of this method would look like this:

```
namespace TechDivision\Import\Services;

class RegistryProcessor implements RegistryProcessorInterface
{

    // other methods still go here

    /**
     * This method merges the passed attributes with an array that
     * has already been added under the given key.
     *
     * If no value will be found under the passed key, the attributes
     * will be registered.
     *
     * @param mixed $key        The key of the attributes that have to be merged with the
passed ones
     * @param array $attributes The attributes that have to be merged with the existing ones
```

```
     *
     * @return void
     * @throws \Exception Is thrown, if the already registered value is no array
     * @link http://php.net/array_replace_recursive
     */
    public function mergeAttributesRecursive($key, array $attributes)
    {

        // merge the passed attributes in a thread-safe manner
        return $this->synchronized(function ($k, $a) {
            // if the key not exists, simply add the new attributes
            if (!isset($this->attributes[$k])) {
                $this->attributes[$k] = $a;
                return;
            }
            // if the key exists and the value is an array, merge it with the passed array
            if (isset($this->attributes[$k]) && is_array($this->attributes[$k])) {
                $this->attributes[$k] = array_replace_recursive($this->attributes[$k], $a);
                return;
            }
        }, $key, $attributes);

        // throw an exception if the key exists, but the found value is not of type array
        throw new \Exception(sprintf('Can\'t merge attributes, because value for key %s
already exists, but is not of type array', $key));
    }
}
```

Operations

An operation reflects a step on an import command like the add-update and combines the necessary functionality like a container.

It allows you to add a custom configuration for an existing operation or to add a new one.

When do I need an operation?

You will need an operation if you want to integrate your custom implementation e.g., based on a plugin, subjects, or observers. A process allows you to combine these classes based on your needs.

| CAUTION | In general, you will need an operation when you want to create a new functionality combined out of plugins, subjects, and observers. |

How to implement an operation?

An operation can not be implemented as it is a set of plugins that will be executed in the configured order.

Up since version **3.8.0**, most operations only have **one** plugin, as the chaining of operations and the order in which they have to be executed can be defined in the shortcuts.

In general, you can configure a process that matches your requirements, like:

```
{
  "operations" : {
    "general": {
      "catalog_product": {
        "my-operation": {
          "plugins" : [
            {
              "id": "import.plugin.subject",
              "subjects" : [
                {
                  "id": "my.subject.id",
                  "file-resolver": {
                    "prefix": "my-prefix"
                  }
                }
              ]
            }
          ]
        }
      }
    }
  }
}
```

The operations have to be defined under the **Magento Edition** they should be available and the entity type they are dedicated to.

If the **Magento Edition** is irrelevant, you can use general, otherwise, ce (for the community) or ee (for commerce) are supported values. = Plugins

- Depending on the required functionality, it will be necessary to implement a plugin.
- A plugin is the highest level when starting a component.
- It will not be required to implement a plugin, as the subject or observer level can cover the necessary needs in many cases.

When do I need a plugin?

It would help if consider implementing a plugin in either one of these cases.

- You want to deal with all artifacts of an import, and you want to do something with the files before they are processed or after they have been processed, e.g. like the artifact plugin that compresses all the import artifacts into a **ZIP** and moves it to a configurable folder
- You want to load some data from the database or the filesystem that will be needed later on in your subjects or observers, during the files will be processed, e.g. like the global data plugin that loads the available attributes and adds them to the registry
- You need to pre-initialize something before the primary import process starts, e.g. like the cache warmer plugin that warms the registered repository

- You need to do something after the import has been finished, e.g. like the missing options plugin that sends a list with missing option values, a configurable receiver

CAUTION   In general, you need a plugin if you want to interact before or after the primary import step, or you have to implement some business logic that needs access to **all important artifacts** at the same time.

How to implement a plugin?

A good example is the TechDivision\Import\Plugins\GlobalDataPlugin that is part of the **Pacemaker Community Edition (CE)** core.

In general, you do not have to write the plugin from scratch; instead, extend the TechDivision\Import\Plugins\AbstractPlugin class that implements the TechDivision\Import\Plugins\PluginInterface which **must** be implemented by every plugin.

The interface defines the setPluginConfiguration() method, which expects the plugin configuration with the optional parameters.

```
{
  "id": "import.plugin.global.data"
}
```

And the process() method that will have to implement the main plugin's functionality.

The TechDivision\Import\Plugins\GlobalDataPlugin loads the global data, e.g. entity types from the import processor and injects it into the registry.

It makes it available by all following plugins, subjects, and observers and helps to avoid unnecessary database access.

```
namespace TechDivision\Import\Plugins;

use TechDivision\Import\Utils\RegistryKeys;

class GlobalDataPlugin extends AbstractPlugin
{
    /**
     * Process the plugin functionality.
     *
     * @return void
     * @throws \Exception Is thrown, if the plugin can not be processed
     */
    public function process()
    {

        // load the global data from the import processor
        $globalData = $this->getImportProcessor()->getGlobalData();

        // add the status with the global data
        $this->getRegistryProcessor()->mergeAttributesRecursive(
            RegistryKeys::STATUS,
            array(RegistryKeys::GLOBAL_DATA => $globalData)
```

```
        );
    }
}
```

OOTB Plug-Ins

The standard plugins are part of the **Pacemaker Community Edition (CE)** core and can be used **OOTB**.

When it is necessary to implement your components, you will go fine in most cases, if you do not implement your plugin, but instead use the SubjectPlugin and provide your functionality in the form of subjects, observers, and services.

Cache Warmer

Has been removed up with version **3.8.0**

Global Data

Load's the global data necessary for the import process from the database and adds it to the registry so that every plugin can access it. You can find a more detailed explanation of the plugin in the section above.

The configuration has to look like the following:

```
{
  "id": "import.plugin.global.data"
}
```

Subject

It is the plugin that does the main work by invoking the registered subjects and their registered observers and callbacks.

Using **Pacemaker Community Edition (CE)** as a framework, this plugin will be a good entry point to add your custom functionality.

In most cases, it will be enough to write a subject and observers that provides the necessary functionality and will be invoked by this plugin.

The plugin configuration is

```
{
  "id": "import.plugin.subject",
  "subjects": [ ... ]
}
```

You can always register as many subjects and observers as necessary.

Missing Option Values

This plugin provides the extended functionality to track whether an attribute option value, referenced in a **CSV** import file, is available or not, depending on debug mode enabled or not.

If the debug mode is **not** enabled, an exception will be thrown immediately. Otherwise, each missing attribute option value will

be written to the **CSV** file missing-option-values.csv will be stored in the temporary import directory and optionally sent to the specified mail recipients.

The configuration of the plugin can look like

```
{
  "id": "import.plugin.missing.option.values",
  "swift-mailer" : {
    "id" : "import.logger.factory.transport.swift.smtp",
    "params" : {
      "to" : "tw@techdivision.com",
      "from" : "pacemaker@techdivision.com",
      "subject": "Something Went Wrong",
      "content-type" : "text/plain"
    },
    "transport" : {
      "params" : {
        "smtp-host" : "mail.techdivision.com",
        "smtp-port" : 25
      }
    }
  }
}
```

> **TIP**   The swift-mailer configuration node is optional.

Only if the **Swift Mailer** configuration is available, the **CSV** file will be sent to the specified recipients.

Subjects

I am pretending that you do not need to implement an entirely new functionality where a plug-in makes sense, e.g. your want to invoke a method of the **Magento RESTFul API** after the product import has been finished; in most cases, a combination of one or more subject(s) with some observers can fully cover the requirements of a new use case.

This chapter will discuss when it makes sense to implement a subject and how it could be done.

When do I need a subject?

The TechDivision\Import\Plugins\SubjectPlugin implements the observer pattern and is the first choice for most use cases.

These plug-ins allow you to register an unlimited number of subjects, whereas each of them can have an unlimited number of observers.

In case a new file has to be imported is found, the TechDivision\Import\Plugins\SubjectPlugin invokes the process() method of **all** registered subjects on this file in the order they have been registered.

The topic itself processes the file's content by invoking **ALL** registered observers for **each** row of the given file.

You can imagine this as something like a chain that can be configured by a workflow engine. This approach allows you to process nearly every file by reading the file contents row by row.

So, usually, you need to implement a subject when

- You want to process an import file, independent which format it has
- You want to share data between observers or pass the data to the following subjects
- You need to customize the file parsing, e.g. not parsing a single row, but a bunch of rows

TIP In general, you need a subject when you want to do something with a **file**.

How to implement a subject?

In general, you should consider extending TechDivision\Import\Subjects\AbstractSubject or one of its subclasses, e.g. TechDivision\Import\Subjects\AbstractEavSubject if you want to implement the import for another **EAV** entity.

It would at least help if you implemented the interface TechDivision\Import\Subjects\SubjectInterface which is the minimum requirement for a subject implementation.

Let's implement a standard requirement, where you need a subject that allows one of its observers to load a product with the SKU found in the import file, adding the SKU to entity_id mapping to the subject and finally pass the mappings to the next subject.

```
namespace TechDivision\Import\Product\Subjects;

use TechDivision\Import\Utils\RegistryKeys;
use TechDivision\Import\Subjects\AbstractSubject;

class MySubject extends AbstractSubject
{

    /**
     * The SKU to entity_id mappings we want to pass to the next subject.
     *
     * @var array
     */
    protedted $skuEntityIdMapping = array();

    /**
     * Intializes the previously loaded global data for exactly one bunch.
     *
     * @param string $serial The serial of the actual import
     *
     * @return void
     */
    public function setUp($serial)
    {

        // load the status of the actual import
        $status = $this->getRegistryProcessor()->getAttribute($serial);

        // load the SKU => entity_id mappings from further subjects
```

```php
        $this->skuEntityIdMapping =
$status[RegistryKeys::GLOBAL_DATA][RegistryKeys::SKU_ENTITY_ID_MAPPING];

        // invoke the parent method
        parent::setUp($serial);
    }

    /**
     * Clean up the global data after importing the bunch.
     *
     * @param string $serial The serial of the actual import
     *
     * @return void
     */
    public function tearDown($serial)
    {

        // load the registry processor and add the SKU => entity_id mappings
        $this->getRegistryProcessor()->mergeAttributesRecursive(
            $serial,
            array(
                RegistryKeys::SKU_ENTITY_ID_MAPPING => $this->skuEntityIdMapping
            )
        );

        // invoke the parent method
        parent::tearDown($serial);
    }

    /**
     * Add the passed SKU => entity ID mapping.
     *
     * @param string  $sku      The SKU to map
     * @param integer $entityId The entity ID to be mapped
     *
     * @return void
     */
    public function addSkuEntityIdMapping($sku, $entityId)
    {
        $this->skuEntityIdMapping[$sku] = $entityId;
    }
}
```

The subject provides the addSkuEntityIdMappping() method, which will be invoked by the observer with the **ID**
import_product.observer.my.observer that loads the product by the SKU found in the import file.

Additionally, it implements the methods setUp() and tearDown(), that'll be invoked automatically before and after the import
file has been processed.

These methods allow us to load/add data from/to the TechDivision\Import\Services\RegistryProcessor, which acts as a data

container for the whole import process passing it from one subject to next.

To make your subject accessible for the **Workflow Engine**, you first have to define it in the **Symfony DI** configuration file symfony/Resources/config/services.xml of your component.

Depending on your namespace, this would look like

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">
    <services>
        <service id="import_product.subject.my"
class="TechDivision\Import\Product\Subjects\MySubject" shared="false">
            <argument type="service" id="import.processor.registry"/>
            <argument type="service" id="import.generator.core.config.data.uid"/>
            <argument type="service" id="loggers"/>
            <argument type="service" id="import.events.emitter"/>
        </service>
        <service id="import_product.observer.my"
class="TechDivision\Import\Product\Observers\MyObserver">
            <argument type="service" id="import_product.processor.product.bunch"/>
        </service>
    </services>
</container>
```

The subject from above can finally be added to the **Workflow Engine** with the following configuration.

```json
{
    "id": "import.plugin.subject",
    "subjects": [
      {
        "id": "import_product.subject.my",
        "identifier": "files",
          "file-resolver": {
            "prefix": "product-import"
          },
        "observers": [
          "import_product.observer.my"
        ]
      }
    ]
}
```

Instead we will implement the observer with the **ID** import_product.observer.my.observer in the next chapter regarding Observers . = Observers

In the previous chapter Subjects , we have discussed when a subject is needed and how it can be implemented.

When do I need an observer?

As a subject itself usually will **not** implement the main import business logic, this will be the observers' responsibility. So it would help if you implemented an observer when

- You want to load data from the database based on values you found in the import file
- You want to load data to make it available for the following observers or subjects
- You want to persist data assembled from values you found in a row of an import file

| TIP | In general, you need an observer when you want to do something with a **row**. |

How to implement an observer?

The TechDivision\Import\Observers\AbstractObserver will be a perfect choice to be extended by our first observer implementation. At least an observer has to implement the interface TechDivision\Import\Observers\ObserverInterface.

Our example observer tries to load the product with the SKU found in the actual row and adds the SKU to entity_id mapping to the subject.

```php
namespace TechDivision\Import\Product\Observers;

use TechDivision\Import\Utils\RegistryKeys;
use TechDivision\Import\Subjects\SubjectInterface;
use TechDivision\Import\Observers\AbstractObserver;

class MyObserver extends AbstractObserver
{

    /**
     * The product bunch processor instance.
     *
     * @var \TechDivision\Import\Product\Services\ProductBunchProcessorInterface
     */
    protected $processor;

    /**
     * Initialize the observer with the passed product bunch processor instance.
     *
     * @param \TechDivision\Import\Product\Services\ProductBunchProcessorInterface
$productBunchProcessor The product bunch processor instance
     */
    public function __construct(ProductBunchProcessorInterface $processor)
    {
        $this->processor = $processor;
    }

    /**
     * Will be invoked by the subject once for every row.
     *
```

```php
     * @param \TechDivision\Import\Subjects\SubjectInterface $subject The subject instance
     *
     * @return array The modified row
     * @see \TechDivision\Import\Observers\ObserverInterface::handle()
     */
    public function handle(SubjectInterface $subject)
    {

        // initialize the row
        $this->setSubject($subject);
        $this->setRow($subject->getRow());

        // try to load the product with the SKU of the actual row and store the entity ID =>
SKU mapping in the subject
        if ($product = $this->processor->loadProduct($this->getValue(ColumnKeys::SKU)) {
            $this->subject->addSkuEntityIdMapping($product[MemberNames::ENTITY_ID],
$product[MemberNames::SKU]);
        } else {
            throw new \Exception(sprintf('Can\'t load product with SKU "%s"', $sku));
        }

        // return the processed row
        return $this->getRow();
    }
}
```

Callbacks

When do I need a callback?

Callbacks can be used to transform values found in the **CSV** file into the necessary types that need to be stored in the database.

For example, the default **Magento 2 CSV** format allows the values

- Catalog
- Search
- Catalog, Search
- Not Visible Individually

For the column visibility. These values can not be stored in the appropriate database column, as this expects integer values.

Therefore, a callback can transform the string into the correct integer value; in this case, the class TechDivision\\Import\\Product\\Callbacks\\VisibilityCallback.

The necessary callbacks to transform the **Magento 2** standard attributes found in the **CSV** file are already defined by default.

When a new, user-defined attribute will be added, e.g. with a setup script, the **Pacemaker Community Edition (CE)** tries to find the best matching callback, depending on the attribute's frontend_input value. **Pacemaker Community Edition (CE)** comes

with callbacks for

- select
- multiselect
- boolean

frontend_input types. Callbacks for other input types will be part of upcoming versions and can always be implemented by the developers using **Pacemaker Community Edition (CE)** in their project.

TIP	In general, you need a callback if you want to do something with the **VALUE** on a specific column in each row of a **CSV** file. Please be aware that a custom callback will **replace** the default callback and **not** be appended!

How to implement a callback?

To implement a callback, you can extend the abstract class TechDivision\Import\Callbacks\AbstractCallback.

The callback's handle() method expects the invoking observer as a parameter that provides the method getAttributeValue that you access to the value you want to process.

The callback **must** returns the processed value to allow the following callbacks also to process it.

```
namespace TechDivision\Import\Product\Callbacks;

use TechDivision\Import\Observers\AttributeCodeAndValueAwareObserverInterface;

/**
 * A callback implementation that converts the passed visibility.
 *
 * @author    Tim Wagner <t.wagner@techdivision.com>
 * @copyright 2016 TechDivision GmbH <info@techdivision.com>
 * @license   http://opensource.org/licenses/osl-3.0.php Open Software License (OSL 3.0)
 * @link      https://github.com/techdivision/import-product
 * @link      http://www.techdivision.com
 */
class VisibilityCallback extends AbstractCallback
{

    /**
     * Will be invoked by a observer it has been registered for.
     *
     * @param
\TechDivision\Import\Observers\AttributeCodeAndValueAwareObserverInterface|null $observer
The observer
     *
     * @return mixed The modified value
     */
    public function handle(AttributeCodeAndValueAwareObserverInterface $observer = null)
    {
```

```
        // set the observer
        $this->setObserver($observer);

        // replace the passed attribute value into the visibility ID
        return $this->getSubject()->getVisibilityIdByValue($observer->getAttributeValue());
    }
}
```

To register a callback, it has to be added to the array with the callbacks of a subject, like

```
"callbacks": [
  {
    "visibility": [
      "import_product.callback.visibility"
    ]
  }
]
```

In our case visibility, the callback key is the column name; the callback has to be invoked.

Therefore, the callback will be invoked on every column visibility for each row of the processed **CSV** file. = How to extend

The **Pacemaker** import functionality is designed to work standalone.

Up from version 3.8.0, it is possible to use the **Magento** code directory <magento-install-dir>/app/code/ as well to extend the **Pacemaker** import functionality without the requirement to deploy it as a composer library.

Override existing classes

In some cases, it will be necessary to override a default class of the **Pacemaker** import library.

For example if additional attributes have been added to a **non-EAV** entity or the import should keep going if a website that has been referenced in the **CSV** file is not available in the **Magento** instance.

- A minimum **Magento** module has to be created.
- In every project that uses the **Pacemaker** module, it will be named Import, e.g. MyProject\Import.

The directory structure should be simular to:

```
<magento-install-dir>/
 --app/
    --code/
       --MyProject/
          --Import/
             |--registration.php
              --etc/
                  --di.xml
                  --config.xml
                  --module.xml
```

```
        --adminhtml/
            --system.xml
```

As the **Pacemaker** import functionality is entirely based on the **Symfony Framework**, as a bear minimum a **DI** configuration, that enables us to override the default class **must** be available.

The configuration file **must** be located in the directory <magento-install-dir>/app/code/MyProject/Import/symfony/Resources/config/services.xml.

```
<magento-install-dir>/
 --app/
    --code/
       --MyProject/
          --Import/
             |--registration.php
             |--etc/
             |    --di.xml
             |    --config.xml
             |    --module.xml
             |    --adminhtml/
             |       --system.xml
             |--Observers/
             |    --ProductWebsiteObserver.php
              --symfony/
                 --Resources/
                    --config/
                       --services.xml
```

The simple **DI** configuration we use to override the original observer requires the following content:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">
    <services>
        <service
            id="import_product.observer.product.website"
            class="MyProject\Import\Product\Observers\ProductWebsiteObserver"/>
    </services>
</container>
```

The corresponding observer can extend the class that has to be overwritten, or may completely implement the custom business logic.

If the initial class is extended, this could look similar to the following code:

```php
namespace MyProject\Import\Product\Observers;
```

```
/**
 * Observer that creates/updates the product's website relations.
 */
class ProductWebsiteObserver extends
\TechDivision\Import\Product\Observers\ProductWebsiteObserver
{

    /**
     * Process the observer's business logic.
     *
     * @return array The processed row
     */
    protected function process()
    {
        // custom code here
    }
}
```

Add custom functionality

A custom functionality can be added in the same way.

If functionality has to be extended, it will also be necessary to add the workflow engine configuration.

The workflow engine configuration can either be done by using snippets that are located in the global configuration directory under <magento-install-dir>/app/etc/configuration or in the configuration directory of the module, which will be app/code/MyProject/Import/etc.

| NOTE | Everything else is **not** different to writing an extension that will be deployed in the **Magento** vendor directory. |

Register the module

To make the module available for usage in the **Pacemaker** import functionality, it must be registered in the workflow engine configuration.

To achieve this, add a snippet like <magento-install-dir>/app/etc/configuration/additional-vendor-dirs.json which has the following content:

```
{
    "additional-vendor-dirs": [
        {
            "vendor-dir": "app/code",
            "libraries": [
                "MyProject/Import"
            ]
        }
    ]
}
```

Best practices example

FAQ

- Composer runs into auth issues on my Mac OS X machine.
- The performance on the production/stating system is worse than on my local machine.
- Timestamp Detection does not work.
- General

Composer runs into auth issues on my Mac OS X machine.

Question

As Solution Partner or Customer, you received a ext12345 username together with a token. It gives you access via **Composer** to the necessary libraries.

These credentials have to be added in your auth.json, which can, for example, be in the source directory of your project like src/auth.json.

Example:

```
{
    "http-basic": {
        "gitlab.met.tdintern.de": {
            "username": "ext00000",
            "password": "asaZIjkUIollKSADnrlm"
        }
    }
}
```

Whenever **Composer** is invoked, these credentials will be used for the **HTTP** download, **Composer** will do.

In some cases, you will receive a message from **Composer** that you will not have the necessary access rights to install **Pacemaker** or one of its packages.

Anwser

MacOS saves the credentials in the keychain on the host level (https://gitlab.met.tdintern.de).

When invoking **Composer** the first time, and it doesn't matter from which project, the first host entry from the keychain will be used and not the auth.json anymore.

It may lead to the problem that the Pacemaker libraries can not be loaded anymore because of missing access.

In the case of **Pacemaker**, it will be necessary to disable the caching of the system credentials for **HTTPS** calls.

Therefore execute the following commands to remove the credential helper from the **GIT** configuration

```
git config --local --unset-all credential.helper \
   && git config --global --unset-all credential.helper \
```

```
   && git config --system --unset-all credential.helper
```

After that, the credential helper has to re-initialized empty (because of any other tool like **xCode** for example may also use it) with the following command

```
git config --global --add credential.helper "" && composer clear-cache
```

Finally, search in the keychain tool for met and delete the entry gitlab.met.tdintern.de.

If the keychain has been deactivated, in the future **GIT** should **always** use the credentials from the auth.json from your project or the global one.

The performance on the production/stating system is worse than on my local machine.

Question

The performance between your local and any other system differs significantly. What can be the reason?

Anwser

Probably the **MySQL** transaction log has different settings.

The option innodb_flush_log_at_trx_commit by default has the value 1.

It means, that the transaction log will be written by MySQL after each commit.

This option controls the balance between strict **ACID** compliance for commit operations and higher performance that is possible when commit-related **I/O** operations are rearranged and done in batches.

Setting this value to 2, you can achieve better performance, but then you could lose transactions in a crash.

Possible values are

0	write and flush once per second
1	write and flush at each commit
2	write at commit, flush once per second

For example, switching this value from 1 to 2 the import performance improves from 02:06:10 to 00:03:29 h which is for sure significant

ID	Pipeline	Created	Finished	Duration
219	xxx_import_catalog	Oct 24, 2019 2:47:04 PM	Oct 24, 2019 4:53:14 PM	02:06:10 h
221	xxx_import_catalog	Oct 24, 2019 5:02:02 PM	Oct 24, 2019 5:05:31 PM	00:03:29 h

Timestamp Detection does not work.

Question

I'm using the Pacemaker Professional Edition (PE) > 3.8.0, when I try to activate the timestamp detection with --use -timestamp=true, everything seems to work fine but the performance is at the same level as when i activate the change-set detection with --use-change-set=true.

The log file does not contain any error messages, as well as the console.

Anwser

It may result out of the problem that the date format in your **CSV** file is different from the default format and the necessary date format has not been configured in the configuration.

Pretending you are using **Magento Commerce**, to change the date format to Y-m-d H:i:s, create a snippet named <magent-install-dir>/app/etc/configuration/operations.json that contains the following content

```json
{
  "operations": {
    "ee": {
      "catalog_product": {
        "validate": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "listeners": [
                {
                  "plugin.process.start": [
                    "import.listener.reset.loader.eav.attribute.option.value"
                  ],
                  "plugin.process.success": [
                    "import.listener.stop.validation"
                  ]
                }
              ],
              "subjects": [
                {
                  "id": "import.subject.validator",
                  "create-imported-file": false,
                  "date-converter": {
                    "source-date-format": "Y-m-d H:i:s"
                  },
                  "file-resolver": {
                    "prefix": "product-import"
                  },
                  "listeners": [
                    {
                      "subject.artefact.header.row.process.start": [
                        "import.listener.validate.header.row"
                      ]
                    }
                  ],
```

```json
                        "params": {
                          "custom-validations": {
                            "sku": [
                              "/.+/"
                            ],
                            "product_type": [
                              "simple",
                              "virtual",
                              "configurable",
                              "bundle",
                              "grouped",
                              "giftcard",
                              "designyourown"
                            ],
                            "visibility": [
                              "Not Visible Individually",
                              "Catalog",
                              "Search",
                              "Catalog, Search"
                            ]
                          }
                        },
                        "observers": [
                          {
                            "import": [
                              "import_product.observer.composite.base.validate"
                            ]
                          }
                        ],
                        "callbacks": [
                          {
                            "sku": [
                              "import.callback.custom.regex.validator"
                            ],
                            "store_view_code": [
                              "import.callback.store.view.code.validator"
                            ],
                            "attribute_set_code": [
                              "import.callback.attribute.set.name.validator"
                            ],
                            "product_type": [
                              "import.callback.custom.array.validator"
                            ],
                            "tax_class_id": [
                              "import_product.callback.validator.tax.class"
                            ],
                            "product_websites": [
                              "import.callback.store.website.validator"
                            ],
```

```
                    "visibility": [
                      "import.callback.visibility.validator"
                    ],
                    "related_skus": [
                      "import_product.callback.validator.link"
                    ],
                    "upsell_skus": [
                      "import_product.callback.validator.link"
                    ],
                    "crosssell_skus": [
                      "import_product.callback.validator.link"
                    ],
                    "created_at": [
                      "import.callback.validator.datetime"
                    ],
                    "updated_at": [
                      "import.callback.validator.datetime"
                    ],
                    "special_price_to_date": [
                      "import.callback.validator.datetime"
                    ],
                    "special_price_from_date": [
                      "import.callback.validator.datetime"
                    ],
                    "custom_design_to": [
                      "import.callback.validator.datetime"
                    ],
                    "custom_design_from": [
                      "import.callback.validator.datetime"
                    ],
                    "new_to_date": [
                      "import.callback.validator.datetime"
                    ],
                    "new_from_date": [
                      "import.callback.validator.datetime"
                    ],
                    "price": [
                      "import.callback.validator.number"
                    ],
                    "special_price": [
                      "import.callback.validator.number"
                    ],
                    "map_price": [
                      "import.callback.validator.number"
                    ],
                    "msrp_price": [
                      "import.callback.validator.number"
                    ],
                    "qty": [
```

```json
                          "import.callback.validator.number"
                        ],
                        "is_returnable": [
                          "import_product_ee.callback.rma.validator"
                        ]
                      }
                    ]
                  }
                ]
              }
            }
          },
          "replace": {
            "plugins": {
              "subject": {
                "id": "import.plugin.subject",
                "subjects": [
                  {
                    "id": "import_product_ee.subject.bunch",
                    "date-converter": {
                      "source-date-format": "Y-m-d H:i:s"
                    },
                    "file-resolver": {
                      "prefix": "product-import"
                    },
                    "params": {
                      "copy-images": false
                    },
                    "observers": [
                      {
                        "import": [
                          "import_product_ee.observer.composite.base.replace"
                        ]
                      }
                    ]
                  }
                ]
              }
            }
          },
          "add-update": {
            "plugins": {
              "subject": {
                "id": "import.plugin.subject",
                "subjects": [
                  {
                    "id": "import_product_ee.subject.bunch",
                    "date-converter": {
                      "source-date-format": "Y-m-d H:i:s"
```

```
                        },
                        "file-resolver": {
                          "prefix": "product-import"
                        },
                        "params": {
                          "copy-images": false,
                          "clean-up-media-gallery": true,
                          "clean-up-empty-image-columns": true,
                          "clean-up-website-product-relations": true,
                          "clean-up-category-product-relations": true,
                          "clean-up-empty-columns": [
                            "special_price",
                            "special_price_from_date",
                            "special_price_to_date"
                          ]
                        },
                        "observers": [
                          {
                            "import": [
                              "import_product_ee.observer.composite.base.add_update"
                            ]
                          }
                        ]
                      }
                    ]
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

General

Question

General

Anwer

Field contents should be enclosed in double-quotes (").

Professional

Getting started

Pacemaker is an extension **bundle (1)** for **Magento 2**.

- It is a new way to organize your background processes inside your shop system.
- **Pacemaker** uses the pipeline pattern, which is familiar to most developers from tools like Gitlab, Jenkins, or Travis.
- Developers are using these tools to manage complex processes like building and deployment of software.
- With **Pacemaker** it is possible to use this powerful pattern also in your eCommerce infrastructure.
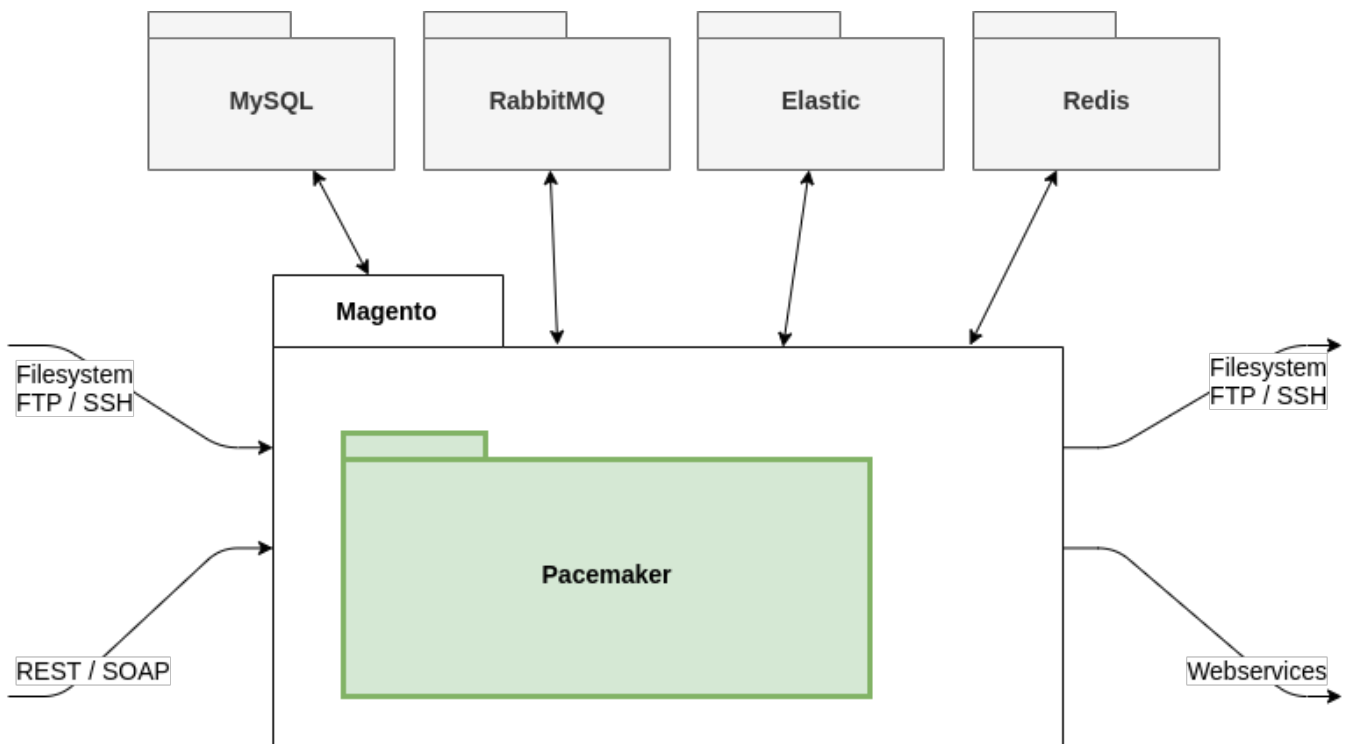
Pacemaker Context

Figure 6. Pacemaker Context View

Pacemaker has access to all existing interfaces of **Magento** as well as the possibility to introduce new connections to foreign or local systems.

Pacemaker Components

Pacemaker consists of many modules. In the current version, these modules can be grouped into four components.

For a better understanding of **Pacemaker**, it is necessary to know that each of these components is a solution for different types of problems.

Each component has its own dependencies, and whenever you introduce new functionality to **Pacemaker**, you need to identify the right component to integrate your code.
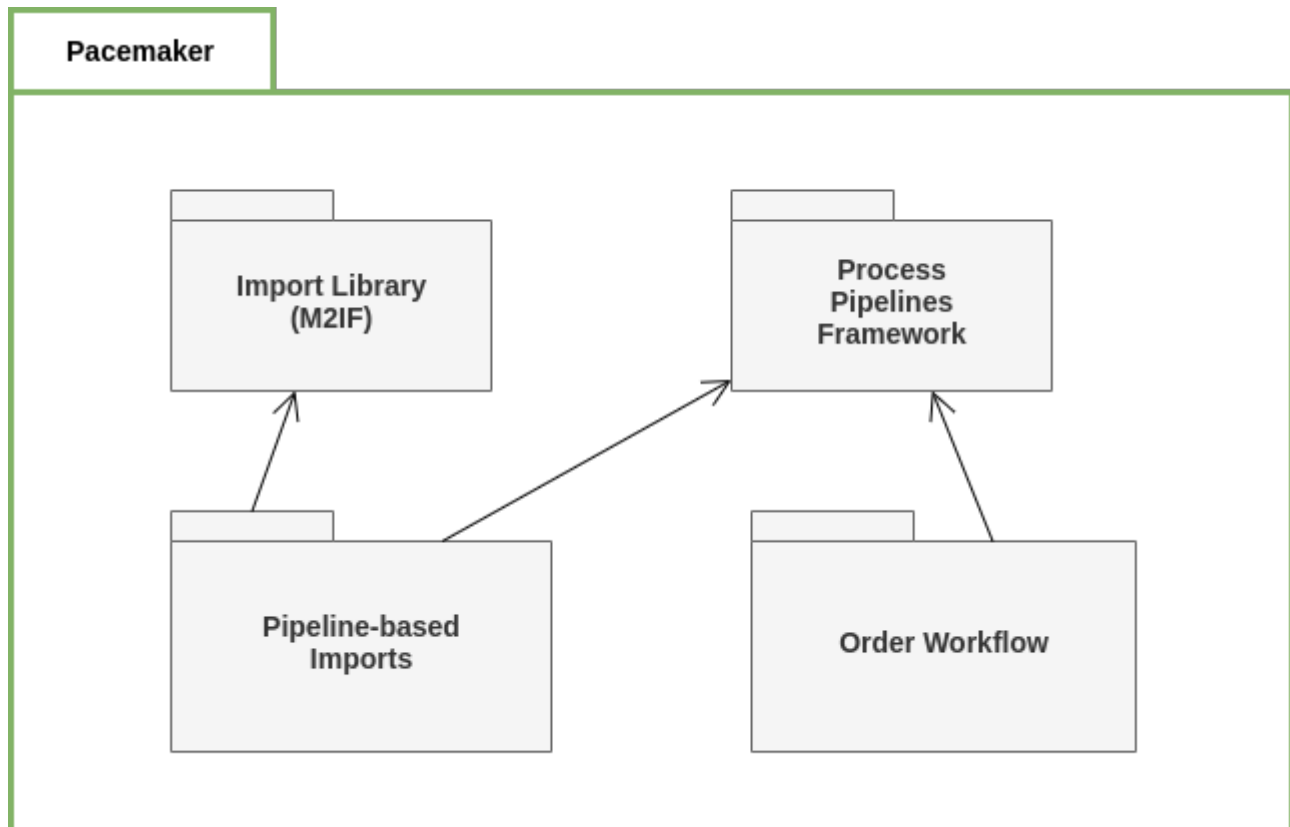
Figure 7. Pacemaker Components

Helpful Notes

Reference resources

Magento resources

- Extension dev guide
- User guide - product attributes
- User guide - complex data
- User guide - customer attributes
- v2.3 Install guide
- start **MQ** consumers via cron
- Supervisor

- Supervisor configuration documentation
- specific system requirements and essentals
- **ERP/OMS**

Pacemaker resources

- Import Framework
- Import cli simple
- **Import Framework** - Magento 2 Import Framework
- Import cli simple
- **Import Framework** - Sample Data
- **Import Framework** - Product Import example CSV
- **Import Framework** - Product Import
- Shortcuts
- Media storage solution

Other resources

- Mysql reference 8.0
- Mageplaza - how to upload product videos in **Magento 2**
- Informit - pipelines
- Wikipedia - **SPMD**
- Jenkins - pipelines
- Gitlab - pipelines
- innodb_flush_log_at_trx_commit

PHP resources

- str_pad() function description in PHP manual

Requirements

PHP Version

Compatible to PHP Version   >= 7.4

Magento

Magento version   >= 2.3

Installation

Before you start the installation, a running **Magento 2** instance is required.

Please refer to Magento's documentation: Install Magento using Composer.

Install Pacemaker via Composer

After the purchase of a **Pacemaker** license, you will receive the following required credentials from our support team:

- **username**
- **password**

You need to add the received credentials to the auth.json file of your Magento instance.

If you didn't use a auth.json file before, you'd find an auth.json.sample file, copy and rename it as auth.json.

Enter our repository (gitlab.met.tdintern.de), your username and password as followed into the http-basic section of your existing auth.json file.

```
{
    "http-basic": {
        "repo.magento.com": {
            "username": "...",
            "password": "..."
        },
        "gitlab.met.tdintern.de": {
            "username": "<YOUR-TOKEN-USER>",
            "password": "<YOUR-TOKEN-PASS>"
        }
    }
}
```

Register the repository

After adding the credentials, you need to register the repository as a possible source. Therefore you need to run the following command or add the repository manually into your composer.json file.

Commandline registration:

```
composer config repositories.repo.met.tdintern.de composer https://repo.met.tdintern.de/
```

Manual entry in composer.json (optional):

```
...
"repositories": {
"repo.met.tdintern.de": {
  "type": "composer",
  "url": "https://repo.met.tdintern.de/"
```

```
},
"repo.magento.com": {
  "type": "composer",
  "url": "https://repo.magento.com/"
}
...
```

Install the module

Run composer require techdivision/pacemaker-professional="~1.3.0" to install the module.

Configuration

Import GUI

The **import GUI** is a add-on for the **Pacemaker Professional Edition (PE)** and **Pacemaker Enterprise Edition (EE)**.

It enables shop owners or other people that work in the **Magento** backend to upload import files e.g., in **CSV** format, create the **.ok** file and start the import process.

It can be done by either the default **Magento CRON**, manually on the **CLI** with the bin/magento pacemaker:import <ID> command or by one of the Pipelines provided by the **Enterprise edition**.

Configure the GUI

In case, the GUI will be used with **Pacemaker Enterprise Edition (EE)**. The configuration can be found under the following path: menu:Stores[Configuration > TechDivision > Pacemaker > Upload] and should contain the following values

Figure 8. Configuration in Magento Backend

GUI settings

The configuration value for Commandline should be set to the **Pacemaker CLI** that has to be used to create the **.ok** file that is necessary to start the import.

By default this value should be set to vendor/bin/pacemaker.

As a pipeline should be used to import the files, the configuration value for Working Directory has to be the same as for the **Pacemaker** working directory , but **without** the leading var.

The configuration for Enable CRON **must** not be set to Yes because this will probably use the default Magento CRON to start the import and passing by the appropriate pipeline.

Upload files

In general, the **GUI** will provide the functionality to upload **CSV** files and adjust the settings for the import process itself, e.g. the mode that has to be used like add-update.

Depending on the **GUI** configuration and the used **Pacemaker edition**, the uploaded files will be imported directly or via one of the pipelines.

NOTE	Keep in mind, that in combination with **Pacemaker Enterprise Edition (EE)** and the configuration described above, the upload **GUI** can only be used to upload files. It is **not** possible to configure the import itself, as this will be done by the pipeline configuration that finally runs the import. Therefore, the configuration will have **no** impact.

Figure 9. Upload GUI for import files

Upload GUI

Via the drop-down Entities, the entity type that matches the uploaded files has to be selected.

If the entity type doesn't match the filename, an error will be triggered.

In case the Product entity has been chosen, the additional drop-down Import Mode will be rendered.

Import Modes

The Default import mode imports all the data that is available in the file, notwithstanding the data has changed since the last import or not.

In contrast, The import mode Change-Set Detection will compare the data (configurable change-sets) from the database with the one found in the import files and only updates the DB if the data has been changed.

They can improve performance massively in add-update mode, as invoking DB **CRUD** operations are very performance

intensive.

NOTE	The mode Change-Set Detection will be the default mode when importing data via default **Pacemaker** pipeline.
	It should be considered well also to activate the feature-configuration/import/caching-and-cache-warming.html#cache-warming[cache warming] functionality, because this can additionally have a significant impact on performance because all the data to be compared to, will be loaded during the bootstrap process in a very efficient manner.

Additional Parameters

The drop-down Operation provides the possibility to select the import mode.

Read more about this topic in the official Pacemaker Community Edition (CE) usage .

The checkboxes for Archive Artefacts, Debug Mode, Single Transaction can be used to activate the appropriate functionality.

To get more information about the feature behind these flags, you can also check out the official Pacemaker Community Edition (CE) usage .

The checkbox Cache Enabled enables the caching functionality.

In general, caching during the import is not very helpful, because the main impact on the performance will have the **CRUD** operations that can not be cached.

In case the import is processed over a network e.g., in the cloud, it can be helpful to lower the time necessary to load the data from the DB. Additionally, the cache warming functionality will only work if the cache will be enabled.

The drop-down Log Level defines the log level that has to be used.

By default this is set to warning, but can be changed to the given requirements.The log file named system.log will be found in the directory with the import artifacts or archived artifact, if the appropriate checkbox has been set. = Batch processing

Compared to the **Pacemaker Community Edition (CE)**, the **Pacemaker Professional Edition (PE)** has a built-in batch processing function that stacks data that needs to be added or updated and executes a multi-value SQL statement based on a set of configurable triggers.

By default, there should be no reason for customization or configuration, but in some exeptions or used as a framework, **Pacemaker** allows configuration when the batch processing is triggered.

- The first trigger is the stack size.
- The second trigger has dedicated attributes to start the execution for the multi-value **SQL** statement.

Configure the stack size

By default, the maximum stack size is 1000.

Before the next value is added to the stack, the SQL statement will be executed, and the stack cleared.

Additionally, a trigger takes care that the stack will also be processed when the import has been finished, but it still contains values.

It is not possible to configure the maximum stack size by the configuration of the workflow engine, as this is the case for

most other configuration options. Instead, the stack size has to be configured by the **DI** configuration.

The constructor of the TechDivision\Import\Batch\Actions\Processors\GenericBatchProcessor implementation, which provides the batch functionality, expects four arguments.

The fourth argument is the maximum stack size after that the stack will be cleaned-up.

In case the maximum stack size of the processor, that handles the creation of the product DateTime attribute values, has to be changed, the **DI** configuration can be overwritten, e.g. with

```
<service
    id="import_product.action.processor.product.datetime.create"
    class="TechDivision\Import\Batch\Actions\Processors\GenericBatchProcessor">
    <argument type="service" id="connection"/>
    <argument type="service" id="import_batch.repository.sql.statement"/>
    <argument type="collection">
        <argument type="constant">
            TechDivision\Import\Batch\Utils\SqlStatementKeys::CREATE_UPDATE_PRODUCT_DATETIME
        </argument>
    </argument>
    <argument type="integer">2000</argument>
</service>
```

Configure the dedicated attributes

The option to change the dedicated attributes that trigger the stack clean-up allows, besides the url_key attribute, to register additional attributes.

If the stack should also be cleaned-up, when a value for the attribute url_path has been added to the stack, the **DI** configuration can be overwritten by adding the appropriate attribute to the loader's collection argument.

```
<service
    id="import_batch.loader.product.varchar.processor.attribute.id"
    class="TechDivision\Import\Batch\Loaders\GenericAttributeIdLoader">
    <argument type="service" id="configuration"/>
    <argument type="service" id="import.processor.import"/>
    <argument type="collection">
        <argument
type="constant">TechDivision\Import\Product\Utils\MemberNames::URL_KEY</argument>
        <argument
type="constant">TechDivision\Import\Product\Utils\MemberNames::URL_PATH</argument>
    </argument>
</service>
```

It will be passed as the fifth argument to the processor that handles the creation of the product varchar attribute values and is based on the TechDivision\Import\Batch\Actions\Processors\GenericAttributeBatchProcessor implementation.

| **NOTE** | The url_key attribute triggers the stack clean-up of the processor, which handles the product's creation of the product varchar attributes, because URL key management makes it necessary always to have the actual **URL** keys in the database. |

Caching & cache warming

The **Pacemaker Professional Edition (PE)** adds extended caching functionality as well as the possibility to warm the caches.

In general, caching in an import a scenario may not improve speed significantly.

Still, it provides an additional option to improve speed or at least save money in a Cloud environment by reducing database queries to a minimum, which is indeed slower than a dedicated server.

The cache is separated into two types and reduces the amount of transferred data.

A static cache can **not** be deactivated because it is used to cache the global data and the artifacts that will be created during the import process.

For example, to create the variants after the simples have been created. By this, it prevents the database from unnecessary queries.

Console

The cache is **disabled** by default to avoid uncontrolled memory usage. But the configurable cache can be enabled on the console by adding the parameter --cache-enabled=true to one of the import commands like

```
vendor/bin/import-pro import:products --cache-enabled=true
```

Enable/Disable cache

In some cases, it will make sense to enable the cache functionality, especially in scenarios where **Pacemaker** has to work in a distributed environment e.g. on **AWS**.

To enabled/disabled the configurable cache or change the **TTL** without adding the console parameter every time it will be invoked, add a snippet <magento-install-dir>/app/etc/configuration/cache.json

```
{
  "caches": [
    {
      "type": "cache.static"
    },
    {
      "type": "cache.configurable",
      "enabled" : true,
      "time": 1440
    }
  ]
}
```

and set "enabled" : "true|false" depending the requirements.

The parameter time adjusts the TTL of the cache entries and should be changed only when there are good reasons for it.

| NOTE | It makes no sense to disable the cache when the cache warming functionality has been enabled, because |

loading depending on the size of the database, the data for the cache warming functionality can take some time, but then the data itself will not be cached and can **not** be used to reduce database access in the end.

Cache warming

The cache warming functionality is also part of the **Pacemaker Professional Edition (PE)**.

By activating the cache warming functionality, **Pacemaker** will load, by using optimized queries, as much data as possible into the cache. It will lower database access during the import process.

Cache warming, therefore, has the most significant impact on performance because all products, as well as there attributes and many relations, will be pre-loaded, and no database queries are necessary anymore.

TIP	Keep in mind that this great advantage comes with a massive memory footprint, as those data will be kept in memory until the import process has been finished.

The default cache-warmer configuration looks like

```
{
  "operations": {
    "general": {
      "catalog_product": {
        "cache-warmer": {
          "plugins": {
            "cache-warmer": {
              "id": "import_caching.plugin.cache.warmer",
              "params": {
                "cache-warmers": [
                  "import_caching.repository.cache.warmer.eav.attribute.option.value",
                  "import_caching.repository.cache.warmer.product",
                  "import_caching.repository.cache.warmer.product.varchar",
                  "import_caching.repository.cache.warmer.product.int",
                  "import_caching.repository.cache.warmer.product.text",
                  "import_caching.repository.cache.warmer.product.decimal",
                  "import_caching.repository.cache.warmer.product.datetime"
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

The **Pacemaker** provides cache-warmers for the product entity as well as the **EAV** attribute option values.

To enable the cache warming functionality, assuming you are using **Magento Community**, simply add the appropriate operation general/catalog_product/cache-warmer to a snippet like <magento-install-dir>/app/etc/configuration/shortcuts.json with the shortcuts which could then look like

```
{
  "shortcuts": {
    "ce": {
      "catalog_product": {
        "add-update": [
          "general/general/global-data",
          "general/general/move-files",
          "general/catalog_product/cache-warmer",
          "general/catalog_product/collect-data",
          "general/eav_attribute/convert",
          "general/eav_attribute/add-update.options",
          "general/eav_attribute/add-update.option-values",
          "general/eav_attribute/add-update.swatch-values",
          "general/catalog_category/convert",
          "ce/catalog_category/sort",
          "ce/catalog_category/add-update",
          "ce/catalog_category/add-update.path",
          "ce/catalog_category/add-update.url-rewrite",
          "ce/catalog_category/children-count",
          "ce/catalog_product/validate",
          "ce/catalog_product/add-update",
          "ce/catalog_product/add-update.variants",
          "ce/catalog_product/add-update.bundles",
          "ce/catalog_product/add-update.links",
          "ce/catalog_product/add-update.grouped",
          "ce/catalog_product/add-update.media",
          "general/catalog_product/add-update.msi",
          "general/catalog_product/add-update.url-rewrites"
        ]
      }
    }
  }
}
```

NOTE	In general, the cache warming functionality is only useful in the add-update operation, as the delete and replace operations doesn't make usage of pre-cached data, because of their implementation nature.

Finders

Last but not least, the library techdivision/caching, which is also part of **Pacemaker Professional Edition (PE)**, comes with replacements for the general finder implementations, which uses the configurable cache type.

Therefore, the default finders are overwritten by **DI** and use the configurable cache to cache the result of database queries. It makes sure the cache will be invalided when existing data has been replaced.

```
{
  "finder-mappings": {
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DATETIMES":
```

```
        "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DECIMALS":
        "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_INTS":
        "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_TEXTS":
        "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHARS":
        "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCTS":
        "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT":
        "import_caching.repository.finder.factory.unique.entity.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DATETIMES_BY_PK_AND_STORE_I
D":
        "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DECIMALS_BY_PK_AND_STORE_ID
":
        "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_INTS_BY_PK_AND_STORE_ID":
        "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_TEXTS_BY_PK_AND_STORE_ID":
        "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHARS_BY_PK_AND_STORE_ID
":
        "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHAR_BY_ATTRIBUTE_CODE_A
ND_ENTITY_TYPE_ID_AND_STORE_ID":
        "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHAR_BY_ATTRIBUTE_CODE_A
ND_ENTITY_TYPE_ID_AND_STORE_ID_AND_VALUE" :
    "import_caching.repository.finder.factory.unique.cached"
  }
}
```

TIP

Each of those finders replaces a version that doesn't use the cache.

So the finders can be used for a fine-grained cache configuration and allow the adjust the memory consumption of **Pacemaker** during the import process.

Fine-grained cache configuration

As described above, cache warming may have a significant impact on memory consumption.

In combination with the configuration of the finders that have to be used, it will be possible to adjust the memory consumption to your needs.

Therefore the snippet `<magento-install-dir>/app/etc/configuration/operations.json`overwrites the default cache warmers like

```
{
  "operations": {
    "general": {
      "catalog_product": {
        "cache-warmer": {
          "plugins": {
            "cache-warmer": {
              "id": "import_caching.plugin.cache.warmer",
              "params": {
                "cache-warmers": [
                  "import_caching.repository.cache.warmer.eav.attribute.option.value",
                  "import_caching.repository.cache.warmer.product.varchar",
                  "import_caching.repository.cache.warmer.product.int",
                  "import_caching.repository.cache.warmer.product.text",
                  "import_caching.repository.cache.warmer.product.decimal",
                  "import_caching.repository.cache.warmer.product.datetime"
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

and will remove the cache warmer "import_caching.repository.cache.warmer.product" for the products.

Additionally, a snippet like <magento-install-dir>/app/etc/configuration/finder-mappings.json, that overwrites the default finder configuration and replaces the cache finder implementation import_caching.repository.finder.factory.unique.entity.cached for the SQL statement TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT with a non-cached version import.repository.finder.factory.unique.

It should result in a snippet which would look like

```
{
  "finder-mappings": {
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DATETIMES":
      "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DECIMALS":
```

```
      "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_INTS":
      "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_TEXTS":
      "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHARS":
      "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCTS":
      "import.repository.finder.factory.yielded",
    "TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT":
      "import.repository.finder.factory.unique",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DATETIMES_BY_PK_AND_STORE_I
D":
      "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_DECIMALS_BY_PK_AND_STORE_ID
":
      "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_INTS_BY_PK_AND_STORE_ID":
      "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_TEXTS_BY_PK_AND_STORE_ID":
      "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHARS_BY_PK_AND_STORE_ID
":
      "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHAR_BY_ATTRIBUTE_CODE_A
ND_ENTITY_TYPE_ID_AND_STORE_ID":
      "import_caching.repository.finder.factory.yielded.cached",

"TechDivision\\Import\\Product\\Utils\\SqlStatementKeys::PRODUCT_VARCHAR_BY_ATTRIBUTE_CODE_A
ND_ENTITY_TYPE_ID_AND_STORE_ID_AND_VALUE" :
      "import_caching.repository.finder.factory.unique.cached"
  }
}
```

and prevents **Pacemaker** from caching product entities.

WARNING	• If only the cache warmer will be removed, the products will be loaded into the cache.
	• Instead all at once, what gives a significant performance boost, one after another.
	• It will result at the same level of memory consumption but with lower performance.

TechD•vision

Change-set detection

Comparing to the **Pacemaker Community Edition (CE)**, the **Pacemaker Professional Edition (PE)** comes with a change-set detection feature that detects, based on a configurable set of attributes, if the entity has changed since the last import.

Console

Change-set detection can be activated when invoking the appropriate import command.

To activate it, add the parameter --use-change-set=true when invoking the import, e.g.

```
vendor/bin/import-pro import:products --cache-enabled=true --use-change-set=true
```

NOTE | Do not forget to add the --cache-enabled=true parameter also, otherwise, performance will be reduced significantly.

Configuration

The change-set configuration is a simple array with the attribute names as values, that can be configured individually for each entity, e.g. for the entity catalog_product_entity.

the default configuration looks like

```
{
  "change-sets": {
    "catalog_product_entity": {
      "type_id": null,
      "updated_at": null,
      "has_options": null,
      "attribute_set_id": null,
      "required_options": null
    }
  }
}
```

If the value has to be converted to a specific type, e.g. an integer, before it should be compared against the one in the database, it is also possible to specify the type as value.

For example the change-set configuration for the entity catalog_category_product only tries to detect changes of the position attribute, whereas the position attribute **must** be of type integer

```
{
  "change-sets": {
    "catalog_category_product": {
      "position": "integer"
    }
  }
}
```

To override the default configuration, a snipped with the appropriate changes of the change-set configuration can be added, e.g. <magento-install-dir>/app/etc/configuration/change-sets.json.

> **TIP**
>
> All values in the database and the **CSV** files are in general of type string.
>
> It is **not** necessary to define the type string, instead not specify a value and setting it to null will prevent Pacemaker from doing the expensive typecasting.

Timestamp detection

Comparing to the **Pacemaker Community Edition (CE)**, the **Pacemaker Professional Edition (PE)**, comes with a timestamp detection feature.

It detects, based on the date found in the updated_at column if the entity has changed since the last import.

Console

The timestamp detection can be activated when invoking the appropriate import command.

To enable it, add the parameter --use-timestamp=true when invoking the import, e.g.

```
vendor/bin/import-pro import:products --cache-enabled=true --use-timestamp=true
```

> **NOTE**
>
> Do not forget to add the --cache-enabled=true parameter as well. Otherwise, the performance will be reduced significantly.

Configuration

The timestamp detection functionality has no additional configuration options.

Instead, it uses the change-set detection configuration to query whether the `updated_at' date has changed since the last import.

By the case of processing the primary row, and as the lines or store views, if any, is skipped.

Suppose that in many cases, where the connected **PIM** or **ERP** system is **not** able to provide a **CSV** file that only contains the deltas comparing to the last import, the timestamp detection may have a significant impact on performance, as only rows that contain changed data will be processed.

> **TIP**
>
> When the timestamp has been changed, further processing the import will automatically use the change-set functionality.
>
> To avoid this, consider using the additional paramater --use-change-set=false.

Usage

Run your first predefined import jobs

Pacemaker provides predefined import jobs, which can be used out of the box.

Therefore there are sample import files (**CSV**) included in the installed composer packages.

By importing these sample files, you will import **Magento's** sample data, like they would be created by running the sampledata:deploy command.

Requirements

The following tutorial requires an up and running **Pacemaker** like it is described on the following pages:

- How to install the module / extension
- How to configure the heartbeat (cron)
- How to configure the runner(s)

Copy sample files into observed import directory

You can copy all sample files into the import directory, and **Pacemaker** would automatically initialize import pipelines (What is a pipeline?) for each import bunch (a bunch of **CSV** files).

Execute the following command from the root directory of your **Magento** installation, depending on which kind of import you want to execute.

Catalog import (attributes, categories, products)

Sample data set 1

The first sample data set includes all kinds of imports (attribute sets, attributes, categories, and products), which are bundled in multiple bunches.

```
cp -R vendor/techdivision/pacemaker-import-catalog/sample-data/bunch1/* var/pacemaker/import
```

Sample data set 2

The following sample data set includes only category imports, which are bundled into two bunches.

```
cp -R vendor/techdivision/pacemaker-import-catalog/sample-data/bunch2/* var/pacemaker/import
```

Price import / inventory (stock) import

Sample data for price and inventory import includes only one import file each.

It is also possible to split up these imports into multiple files and optionally cluster them into multiple bunches like it is done for the catalog import.

Use the following command for price import:

```
cp -R vendor/techdivision/pacemaker-import-price/sample-data/bunch1/* var/pacemaker/import
```

Use the following command for inventory (stock) import:

```
cp -R vendor/techdivision/pacemaker-import-inventory/sample-data/bunch1/*
var/pacemaker/import
```

NOTE

After copying the import files into the target directory, you can observe the process by executing the pipeline:status command.

Alternatively, login into the backend (**Magento's admin UI**) and open the menu:System[Pacemaker > Pipelines] Grid.

Figure 10. Result output for bin/magento pipeline:status

Usage import GUI

The **import GUI** is a add-on for the **Pacemaker Professional Edition (PE)** and **Pacemaker Enterprise Edition (EE)**.

It enables shop owners or other people that work in the **Magento** backend to upload import files e.g., in **CSV** format, create the **.ok** file and start the import process.

It can be done by either the default **Magento CRON**, manually on the **CLI** with the bin/magento pacemaker:import <ID> command or by one of the Pipelines provided by the **Enterprise edition**.

Configure the GUI

In case, the GUI will be used with **Pacemaker Enterprise Edition (EE)**. The configuration can be found under the following path: menu:Stores[Configuration > TechDivision > Pacemaker > Upload] and should contain the following values

Figure 11. Configuration in Magento Backend

GUI settings

The configuration value for Commandline should be set to the **Pacemaker CLI** that has to be used to create the **.ok** file that is necessary to start the import.

By default this value should be set to vendor/bin/pacemaker.

As a pipeline should be used to import the files, the configuration value for Working Directory has to be the same as for the **Pacemaker** working directory , but **without** the leading var.

The configuration for Enable CRON **must** not be set to Yes because this will probably use the default Magento CRON to start the import and passing by the appropriate pipeline.

Upload files

In general, the **GUI** will provide the functionality to upload **CSV** files and adjust the settings for the import process itself, e.g. the mode that has to be used like add-update.

Depending on the **GUI** configuration and the used **Pacemaker edition**, the uploaded files will be imported directly or via one of the pipelines.

NOTE

Keep in mind, that in combination with **Pacemaker Enterprise Edition (EE)** and the configuration described above, the upload **GUI** can only be used to upload files.

It is **not** possible to configure the import itself, as this will be done by the pipeline configuration that finally runs the import.

Therefore, the configuration will have **no** impact.

TechD•vision

Figure 12. Upload GUI for import files

Upload GUI

Via the drop-down Entities, the entity type that matches the uploaded files has to be selected.

If the entity type doesn't match the filename, an error will be triggered.

In case the Product entity has been chosen, the additional drop-down Import Mode will be rendered.

Import Modes

The Default import mode imports all the data that is available in the file, notwithstanding the data has changed since the last import or not.

In contrast, The import mode Change-Set Detection will compare the data (configurable change-sets) from the database with the one found in the import files and only updates the DB if the data has been changed.

They can improve performance massively in add-update mode, as invoking DB **CRUD** operations are very performance

intensive.

	The mode Change-Set Detection will be the default mode when importing data via default **Pacemaker** pipeline.
NOTE	It should be considered well also to activate the feature-configuration/import/caching-and-cache-warming.html#cache-warming[cache warming] functionality, because this can additionally have a significant impact on performance because all the data to be compared to, will be loaded during the bootstrap process in a very efficient manner.

Additional Parameters

The drop-down Operation provides the possibility to select the import mode.

Read more about this topic in the official Pacemaker Community Edition (CE) usage .

The checkboxes for Archive Artefacts, Debug Mode, Single Transaction can be used to activate the appropriate functionality.

To get more information about the feature behind these flags, you can also check out the official Pacemaker Community Edition (CE) usage .

The checkbox Cache Enabled enables the caching functionality.

In general, caching during the import is not very helpful, because the main impact on the performance will have the **CRUD** operations that can not be cached.

In case the import is processed over a network e.g., in the cloud, it can be helpful to lower the time necessary to load the data from the DB. Additionally, the cache warming functionality will only work if the cache will be enabled.

The drop-down Log Level defines the log level that has to be used.

By default this is set to warning, but can be changed to the given requirements. The log file named system.log will be found in the directory with the import artifacts or archived artifact, if the appropriate checkbox has been set.

File Structure

Video import

As for every video, a preview image is needed as well, the video import, in general, is an extension of the image import.

Roles

For both images and videos, **Magento** provides so-called roles that can be used to define where the image or has to be rendered.

If necessary, the default roles can dynamically be extended. W henever a new role has been added, **Pacemaker** has support for it by adding new columns that match the role name.

By default, the following roles are available and can be used to define which role an imported image or video should have.

TechD·vision

Role	Description
base	• The base image is the primary image on the product detail page. • Image zoom is activated if you upload an image that is a larger image than the image container. • Depending on the zoom level that you want to achieve, the base image should be two or three times the container's size. • **Example sizes** *without Zoom:* 470 x 470 pixel *with Zoom:* 1100 x 1100 pixel
small	The small image is used for the product images in listings on category and search results pages and displays the product images needed for sections such as for Up-sells, Cross-sells, and the New Products List. • *Example size:* 470 x 470 pixel
thumbnail	Thumbnail images appear in the thumbnail gallery, shopping cart, and in some blocks such as Related Items. • *Example size:* 50 x 50 pixels
swatch	A swatch can be used to illustrate the color, pattern, or texture. Example size: 50 x 50 pixels • *Example size:* 50 x 50 pixels

File Structure

The file structure for images and videos can be dynamically and depends on the roles an image has to be related.

For example, when only one image with the role base should be imported, it's enough to have the columns base_image. The columns base_image_label and base_image_position are optional.

For each additional role, an image has to be related to, the additional columns are necessary, either it is the same image or not.

Column Name	Type	Example	Description
<role>_image	varchar	/m/b/mb01-blue-0.jpg	The relative path to the image, starting with a slash (/).
<role>_image _label	varchar	This is my image.	The image label used as **alt** text.
<role>_image _position	integer	1	The position the image should be rendered in the admin backend and frontend.
additional_im ages	varchar	/m/b/mb01-blue-0.jpg,/m/b/mb01-blue-1.jpg,/m/b/mb01-blue-2.jpg	This column **must** contain a comma (,) separated list of relative paths to images.
additional_im age_labels	varchar	This is my image 1,, This is my image 2	This column **must** contain a comma (,) separated list of **alt** texts for the images in the column additional_attributes whereas it **must** exactly have the same number of elements as additional images have been specified.

Column Name	Type	Example	Description
additional_image_positions	integer	1,3,2	This column **must** contain a comma (,) separated list positions for the images in the column additional_attributes whereas it **must** exactly have the same number of elements as additional images have been specified.
<role>_video	text	https://youtu.be/AzXrs1c92RY	The **URL** to the video should be loaded, by default, only videos from **YouTube** (needs an **API** key) or **Vimeo** are supported.
<role>_video_title	varchar(255)	Pacemaker Product Import	The title will be displayed on top of the video on the product detail page when the user hovers over the video.
<role>_video_description	text	This video shows a simple **Import Framework** product import in a **Magento 2** shop instance.	A description of the video which will be visible in the admin backend only.
<role>_video_provider	varchar(32)	YouTube	The name of the video provider, is either YouTube or Vimeo .
<role>_video_metadata	text	Pacemaker	Additional metadata for the video that will be visible in the Admin backend only.
additional_videos	varchar	https://youtu.be/AzXrs1c92RY,https://vimeo.com/GxXrs1c92BY	This column **must** contain a comma (,) separated list of URLs to additional videos that should be loaded.
additional_video_titles	varchar(255)	Pacemaker product import, A second video about product import	This column **must** contain a comma (,) separated list of titles for the videos in the column additional_videos whereas it **must** exactly have the same number of elements as additional videos have been specified. The title will be displayed above the video on the product detail page if a user hovers over the video.
additional_video_descriptions	text	Pacemaker product import,A second video about product import	This column **must** contain a comma (,) separated list of descriptions for the videos in the column additional_videos whereas it **must** exactly have the same number of elements as additional videos have been specified. The descriptions will be visible in the admin backend only.
additional_video_providers	varchar(32)	YouTube, Vimeo	This column **must** contain a comma (,) separated list of provides for the videos in the column additional_videos whereas it **MUST** exactly have the same number of elements as additional videos have been specified. One of YouTube or Vimeo can be used.
additional_video_metadata	text	YouTube, Vimeo	This column **must** contain a comma (,) separated list of metadata for the videos in the column additional_videos whereas it **must** exactly have the same number of elements as additional videos have been specified. The metadata will be visible in the admin backend only.
hide_from_product_page	varchar	/m/b/mb01-blue-0.jpg,/m/b/mb01-blue-2.jpg	Contains a comma (,) separated list of the relative image paths that should **not** be rendered on the product detail page, whereas it doesn't matter if it is an image with a role or one of the additional images

Column Name	Type	Example	Description
disabled_images	varchar	/m/b/mb01-blue-0.jpg,/m/b/mb01-blue-2.jpg	Contains a comma (,) separated list of the relative image paths that should **not** be rendered anywhere, whereas it doesn't matter if it is an image with a role or one of the additional images
WARNING		If you currently want to import videos to the additional_videos column (which is almost always the case if videos are **not** base, small or thumbnail), the corresponding preview images at the additional_images column **must** be placed first, because the index of the images is used to assign the videos in the additional_videos column.	

WARNING

If you currently want to import videos to the additional_videos column (which is almost always the case if videos are **not** base, small or thumbnail), the corresponding preview images at the additional_images column **must** be placed first, because the index of the images is used to assign the videos in the additional_videos column.

Here is a small example:

Column additional_images /a/z/AzXrs1c92RY.jpg,/m/b/mb03-black-0_alt1.jpg

Column additional_videos https://youtu.be/AzXrs1c92R

During the import process, when the additional_videos column is unpacked, the value is assigned based on its index, i.e., the video with the URL https://youtu.be/AzXrs1c92RY has index 0 and is thus assigned to the image with index 0 in the addtional_images column, i.e. the image /a/z/AzXrs1c92RY.jpg.

If the pictures in the column additional_images would now be reversed, i.e. /m/b/mb03-black-0_alt1.jpg,/a/z/AzXrs1c92RY.jpg then the video would be wrongly assigned to the image /m/b/mb03-black-0_alt1.jpg.

So the preview images of the videos must be in the additional_image column **first**.

Otherwise, the assignment does not work.

The order in the additional_videos column must correspond to the order of the preview images.

Otherwise, the transfer will not work.

The actual order for the display in Admin backend and frontend can then be imported accordingly using the *_position columns.

Dedicated media import

Besides the possibility of importing videos, like images, with the product import itself, the Pacemaker Professional Edition (PE) comes with a command that allows a dedicated import of media files, including pictures and videos.

To give you a better idea, we have a repository with sample data that contains a **CSV** file that only includes the necessary columns for a dedicated media import.

To run a dedicated media import and import a video for the product with the **SKU** 24-MB01, it is necessary to configure the YouTube API first, before the following c ommand can be invoked.

```
cd <magento-install-dir> \
  && rm var/pacemaker/import/*.csv \
  && cp <sample-data-pro-dir>/* var/pacemaker/import \
  && vendor/bin/pacemaker import:products:media add-update \
    --clear-artefacts=false \
    --archive-artefacts=false \
```

```
    --serial=import \
    --source-dir=var/pacemaker \
    --target-dir=var/pacemaker
```

After refreshing the indexes, the video should be visible on the product detail page on the frontend and in the admin backend.
** Components & Concept = How to extend

The **Pacemaker** import functionality is designed to work standalone.

Up from version 3.8.0, it is possible to use the **Magento** code directory <magento-install-dir>/app/code/ to extend the **Pacemaker** import functionality without the need to deploy it as a composer library.

Override existing classes

In some cases, it will be necessary to override a default class of the **Pacemaker** import library.

For example, if additional attributes have been added to a **non-EAV** entity or the import should keep going if a website that has been referenced in the **CSV** file is not available in the **Magento** instance.

For that purpose, a minimum **Magento** module has to be created.

1. In general, in every project that uses **Pacemaker** the module will be named Import, e.g. MyProject\Import.

At this point, the directory structure should look like

```
<magento-install-dir>/
 --app/
    --code/
       --MyProject/
          --Import/
             |--registration.php
              --etc/
                 --di.xml
                 --config.xml
                 --module.xml
                 --adminhtml/
                     --system.xml
```

As the **Pacemaker** import functionality is completely based on Symfony, at least a **DI** configuration, that enables us to override the default class must be available.

The configuration file **must** be located in the directory <magento-install-dir>/app/code/MyProject/Import/symfony/Resources/config/services.xml.

```
<magento-install-dir>/
 --app/
    --code/
       --MyProject/
          --Import/
             |--registration.php
             |--etc/
```

```
            |   --di.xml
            |   --config.xml
            |   --module.xml
            |   --adminhtml/
            |       --system.xml
            |--Observers/
            |   --ProductWebsiteObserver.php
             --symfony/
                --Resources/
                    --config/
                        --services.xml
```

The simple **DI** configuration that we used to override the original observer will have the following content:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">
    <services>
        <service
            id="import_product.observer.product.website"
            class="MyProject\Import\Product\Observers\ProductWebsiteObserver"/>
    </services>
</container>
```

The corresponding observer can extend the class that has to be overwritten, or may completely implement the custom business logic.

Given that the original class is extended, this could look as the following code:

```php
namespace MyProject\Import\Product\Observers;

/**
 * Observer that creates/updates the product's website relations.
 */
class ProductWebsiteObserver extends
\TechDivision\Import\Product\Observers\ProductWebsiteObserver
{

    /**
     * Process the observer's business logic.
     *
     * @return array The processed row
     */
    protected function process()
    {
        // custom code here
    }
```

```
}
```

Add custom functionality

Custom functionality can be added in the same way.

If the feature has to be extended, it will also be necessary to add the workflow engine's configuration.

The workflow engine configuration can either be accomplished be snippets located in the global configuration directory under <magento-install-dir>/app/etc/configuration or in the configuration directory of the module, which will be app/code/MyProject/Import/etc.

> **NOTE**   Everything else will **not** be different than writing an extension that will be deployed in the **Magento** vendor directory.

Register the module

Finally, to make the module available for usage in the **Pacemaker** import functionality, it must be registered in the workflow engine's configuration.

For this purpose, add a snippet <magento-install-dir>/app/code/configuration/additional-vendor-dirs.json which has the following content:

```
{
    "additional-vendor-dirs": [
        {
            "vendor-dir": "app/code",
            "libraries": [
                "MyProject/Import"
            ]
        }
    ]
}
```

Best practices examples

FAQ

- Composer runs into auth issues on my Mac OS X machine.
- The performance on the production/stating system is worse than on my local machine.
- Timestamp Detection does not work.
- General

Composer runs into auth issues on my Mac OS X machine.

> **Question**

As Solution Partner or Customer, you received a ext12345 username together with a token. It gives you access via **Composer** to the necessary libraries.

These credentials have to be added in your auth.json, which can, for example, be in the source directory of your project like src/auth.json.

Example:

```
{
    "http-basic": {
        "gitlab.met.tdintern.de": {
            "username": "ext00000",
            "password": "asaZIjkUIollKSADnrlm"
        }
    }
}
```

Whenever **Composer** is invoked, these credentials will be used for the **HTTP** download, **Composer** will do.

In some cases, you will receive a message from **Composer** that you will not have the necessary access rights to install **Pacemaker** or one of its packages.

Anwser

MacOS saves the credentials in the keychain on the host level (https://gitlab.met.tdintern.de).

When invoking **Composer** the first time, and it doesn't matter from which project, the first host entry from the keychain will be used and not the auth.json anymore.

It may lead to the problem that the Pacemaker libraries can not be loaded anymore because of missing access.

In the case of **Pacemaker**, it will be necessary to disable the caching of the system credentials for **HTTPS** calls.

Therefore execute the following commands to remove the credential helper from the **GIT** configuration

```
git config --local --unset-all credential.helper \
  && git config --global --unset-all credential.helper \
  && git config --system --unset-all credential.helper
```

After that, the credential helper has to re-initialized empty (because of any other tool like **xCode** for example may also use it) with the following command

```
git config --global --add credential.helper "" && composer clear-cache
```

Finally, search in the keychain tool for met and delete the entry gitlab.met.tdintern.de.

If the keychain has been deactivated, in the future **GIT** should **always** use the credentials from the auth.json from your project or the global one.

The performance on the production/stating system is worse than on my local machine.

Question

The performance between your local and any other system differs significantly. What can be the reason?

Anwser

Probably the **MySQL** transaction log has different settings.

The option innodb_flush_log_at_trx_commit by default has the value 1.

It means, that the transaction log will be written by MySQL after each commit.

This option controls the balance between strict **ACID** compliance for commit operations and higher performance that is possible when commit-related **I/O** operations are rearranged and done in batches.

Setting this value to 2, you can achieve better performance, but then you could lose transactions in a crash.

Possible values are

0	write and flush once per second
1	write and flush at each commit
2	write at commit, flush once per second

For example, switching this value from 1 to 2 the import performance improves from 02:06:10 to 00:03:29 h which is for sure significant

ID	Pipeline	Created	Finished	Duration
219	xxx_import_catalog	Oct 24, 2019 2:47:04 PM	Oct 24, 2019 4:53:14 PM	02:06:10 h
221	xxx_import_catalog	Oct 24, 2019 5:02:02 PM	Oct 24, 2019 5:05:31 PM	00:03:29 h

Timestamp Detection does not work.

Question

I'm using the Pacemaker Professional Edition (PE) > 3.8.0, when I try to activate the timestamp detection with --use-timestamp=true, everything seems to work fine but the performance is at the same level as when i activate the change-set detection with --use-change-set=true.

The log file does not contain any error messages, as well as the console.

Anwser

It may result out of the problem that the date format in your **CSV** file is different from the default format and the necessary date format has not been configured in the configuration.

Pretending you are using **Magento Commerce**, to change the date format to Y-m-d H:i:s, create a snippet named <magent-install-dir>/app/etc/configuration/operations.json that contains the following content

```json
{
  "operations": {
    "ee": {
      "catalog_product": {
        "validate": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "listeners": [
                {
                  "plugin.process.start": [
                    "import.listener.reset.loader.eav.attribute.option.value"
                  ],
                  "plugin.process.success": [
                    "import.listener.stop.validation"
                  ]
                }
              ],
              "subjects": [
                {
                  "id": "import.subject.validator",
                  "create-imported-file": false,
                  "date-converter": {
                    "source-date-format": "Y-m-d H:i:s"
                  },
                  "file-resolver": {
                    "prefix": "product-import"
                  },
                  "listeners": [
                    {
                      "subject.artefact.header.row.process.start": [
                        "import.listener.validate.header.row"
                      ]
                    }
                  ],
                  "params": {
                    "custom-validations": {
                      "sku": [
                        "/.+/"
                      ],
                      "product_type": [
                        "simple",
                        "virtual",
                        "configurable",
                        "bundle",
                        "grouped",
                        "giftcard",
                        "designyourown"
                      ],
```

```json
            "visibility": [
              "Not Visible Individually",
              "Catalog",
              "Search",
              "Catalog, Search"
            ]
          }
        },
        "observers": [
          {
            "import": [
              "import_product.observer.composite.base.validate"
            ]
          }
        ],
        "callbacks": [
          {
            "sku": [
              "import.callback.custom.regex.validator"
            ],
            "store_view_code": [
              "import.callback.store.view.code.validator"
            ],
            "attribute_set_code": [
              "import.callback.attribute.set.name.validator"
            ],
            "product_type": [
              "import.callback.custom.array.validator"
            ],
            "tax_class_id": [
              "import_product.callback.validator.tax.class"
            ],
            "product_websites": [
              "import.callback.store.website.validator"
            ],
            "visibility": [
              "import.callback.visibility.validator"
            ],
            "related_skus": [
              "import_product.callback.validator.link"
            ],
            "upsell_skus": [
              "import_product.callback.validator.link"
            ],
            "crosssell_skus": [
              "import_product.callback.validator.link"
            ],
            "created_at": [
              "import.callback.validator.datetime"
```

```json
                ],
                "updated_at": [
                  "import.callback.validator.datetime"
                ],
                "special_price_to_date": [
                  "import.callback.validator.datetime"
                ],
                "special_price_from_date": [
                  "import.callback.validator.datetime"
                ],
                "custom_design_to": [
                  "import.callback.validator.datetime"
                ],
                "custom_design_from": [
                  "import.callback.validator.datetime"
                ],
                "new_to_date": [
                  "import.callback.validator.datetime"
                ],
                "new_from_date": [
                  "import.callback.validator.datetime"
                ],
                "price": [
                  "import.callback.validator.number"
                ],
                "special_price": [
                  "import.callback.validator.number"
                ],
                "map_price": [
                  "import.callback.validator.number"
                ],
                "msrp_price": [
                  "import.callback.validator.number"
                ],
                "qty": [
                  "import.callback.validator.number"
                ],
                "is_returnable": [
                  "import_product_ee.callback.rma.validator"
                ]
              }
            ]
          }
        ]
      }
    }
  },
  "replace": {
    "plugins": {
```

```
            "subject": {
              "id": "import.plugin.subject",
              "subjects": [
                {
                  "id": "import_product_ee.subject.bunch",
                  "date-converter": {
                    "source-date-format": "Y-m-d H:i:s"
                  },
                  "file-resolver": {
                    "prefix": "product-import"
                  },
                  "params": {
                    "copy-images": false
                  },
                  "observers": [
                    {
                      "import": [
                        "import_product_ee.observer.composite.base.replace"
                      ]
                    }
                  ]
                }
              ]
            }
          },
          "add-update": {
            "plugins": {
              "subject": {
                "id": "import.plugin.subject",
                "subjects": [
                  {
                    "id": "import_product_ee.subject.bunch",
                    "date-converter": {
                      "source-date-format": "Y-m-d H:i:s"
                    },
                    "file-resolver": {
                      "prefix": "product-import"
                    },
                    "params": {
                      "copy-images": false,
                      "clean-up-media-gallery": true,
                      "clean-up-empty-image-columns": true,
                      "clean-up-website-product-relations": true,
                      "clean-up-category-product-relations": true,
                      "clean-up-empty-columns": [
                        "special_price",
                        "special_price_from_date",
                        "special_price_to_date"
```

```
                              ]
                            },
                            "observers": [
                                {
                                    "import": [
                                        "import_product_ee.observer.composite.base.add_update"
                                    ]
                                }
                            ]
                        }
                    ]
                }
              }
            }
          }
        }
      }
    }
}
```

General

Question

General

Anwser

Field contents should be enclosed in double-quotes (").

Enterprise

Getting started

Pacemaker is an extension **bundle (1)** for **Magento 2**.

- It is a new way to organize your background processes inside your shop system.
- **Pacemaker** uses the pipeline pattern, which is familiar to most developers from tools like Gitlab, Jenkins, or Travis.
- Developers are using these tools to manage complex processes like building and deployment of software.
- With **Pacemaker** it is possible to use this powerful pattern also in your eCommerce infrastructure.
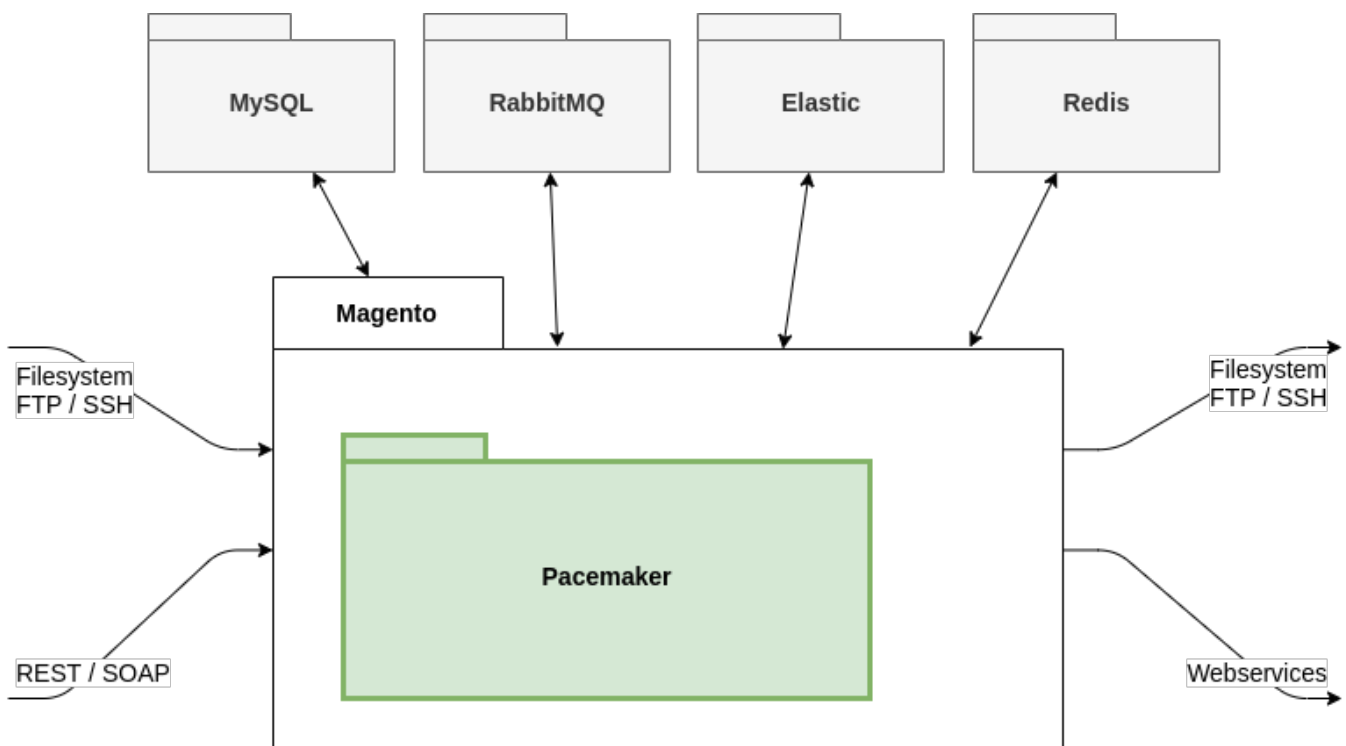
Pacemaker Context

Figure 13. Pacemaker Context View

Pacemaker has access to all existing interfaces of **Magento** as well as the possibility to introduce new connections to foreign or local systems.

Pacemaker Components

Pacemaker consists of many modules. In the current version, these modules can be grouped into four components.

For a better understanding of **Pacemaker**, it is necessary to know that each of these components is a solution for different types of problems.

Each component has its own dependencies, and whenever you introduce new functionality to **Pacemaker**, you need to identify the right component to integrate your code.
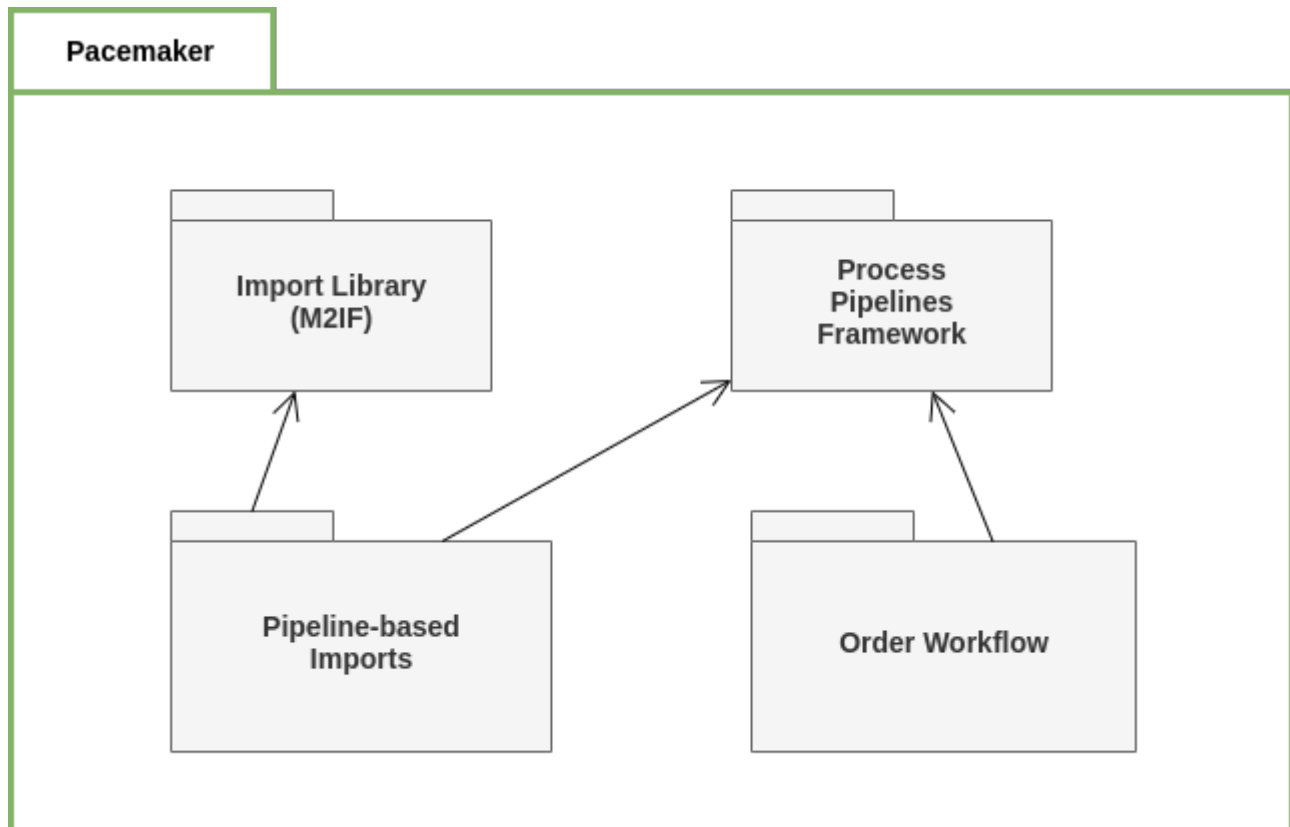
Figure 14. Pacemaker Components

Helpful Notes

Setup RabbitMQ on MacOS X

Pretending you have installed **RabbitMQ**, you may have to execute the following steps to connect **Magento** and **RabbitMQ**.

Create Credentials & Configuration

To fix these issues or to create the connection if not already done, start by creating the **RabbitMQ** user and virtual host as well as setting the permissions with

```
rabbitmqctl add_user admin admin
rabbitmqctl add_vhost pacemaker
rabbitmqctl set_permissions -p pacemaker admin "." "." ".*"
```

In the commands above, we create a user with the username admin and the password admin.

Also, a virtual host for RabbitMQ with the name pacemaker has been created.

Then add the queue node to the Magento configuration under app/etc/env.php. With the **username/password** and virtual host values from above, this has to look like

```
return [
    'queue' => [
        'amqp' => [
            'host' => 'localhost',
            'port' => '5672',
            'user' => 'admin',
            'password' => 'admin',
            'virtualhost' => 'pacemaker',
            'ssl' => 'false'
        ]
    ],
];
```
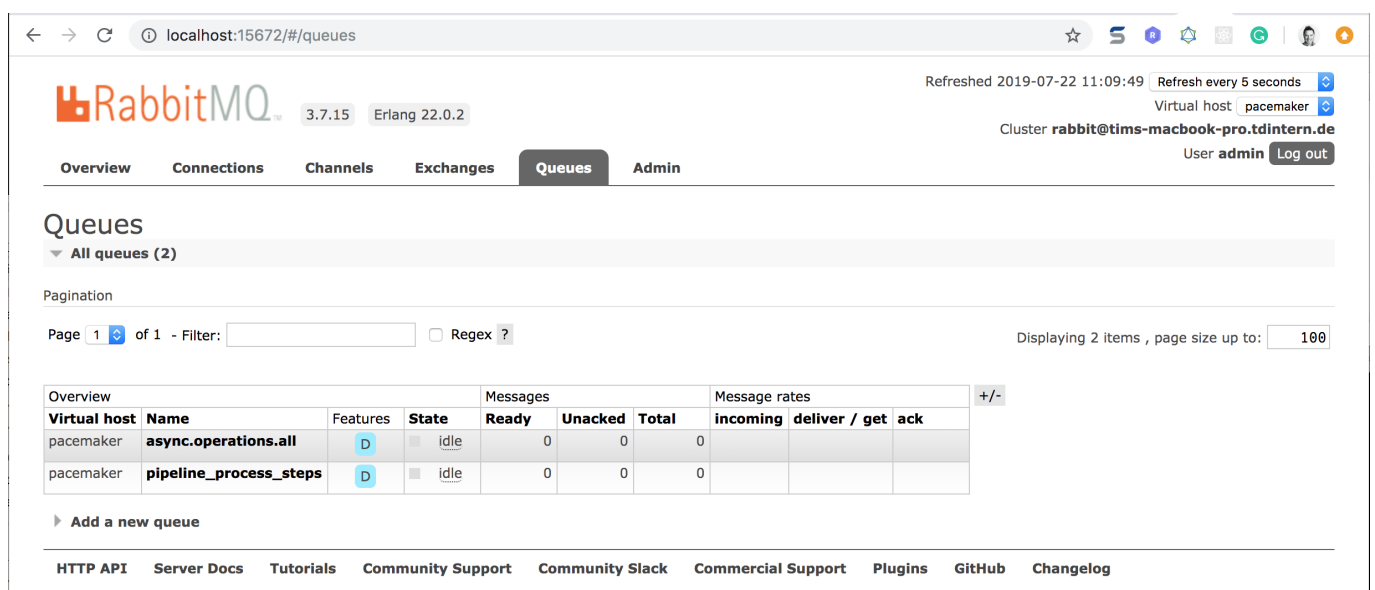
After that, finalize the Magento setup with the following command:

```
bin/magento setup:upgrade
```

If this command is executed successfully, the queues within **RabbitMQ** are created, and the connection will be established.

To start the runner, enter the following command:

```
bin/magento queue:consumers:start pipelineRunner
```


*Figure 15. Additionally in the **RabbitMQ GUI** the queues should now be visible like*

CLI status Update

To get a brief overview of running processes, which can be useful for local development and debugging, it is possible to execute a console command that displays the status of running pipelines on the console

```
bin/magento pipeline:status -w 2
```


Figure 16. Now it should render the following output

Issues with missing AMQP configuration

If you did not setup the **RabbitMQ** connection when installing **Magento**, you will receive one of the two errors below when you try to start the Pacemaker runner.

The message Unknown connection name amqp signals that the **Magento** configuration under app/etc/env.php is missing the **AMQP** connection

```
$ bin/magento queue:consumers:start pipelineRunner

In ConnectionTypeResolver.php line 43:

  Unknown connection name amqp
```

```
queue:consumers:start [--max-messages MAX-MESSAGES] [--batch-size BATCH-SIZE] [--area-code
AREA-CODE]
[--pid-file-path PID-FILE-PATH] [--] <consumer>
```

Whereas the message ACCESS_REFUSED - Login was refused using the authentication mechanism AMQPLAIN. For details see the broker logfile. addresses an issue with the configured **username/password** in the configuration.

```
$ bin/magento queue:consumers:start pipelineRunner

In AbstractConnection.php line 689:

  ACCESS_REFUSED - Login was refused using the authentication mechanism AMQPLAIN.

For further details, please check the broker logfile.

queue:consumers:start [--max-messages MAX-MESSAGES] [--batch-size BATCH-SIZE] [--area-code
AREA-CODE]
[--pid-file-path PID-FILE-PATH] [--] <consumer>
```

Run your first predefined import jobs

Pacemaker provides predefined import jobs, which can be used out of the box.

Therefore there are sample import files (**CSV**) included in the installed composer packages.

By importing these sample files, you will import **Magento's** sample data, like they would be created by running the sampledata:deploy command.

Requirements

The following tutorial requires an up and running **Pacemaker** like it is described on the following pages:

- How to install the module / extension
- How to configure the heartbeat (cron)
- How to configure the runner(s)

Copy sample files into observed import directory

You can copy all sample files into the import directory, and **Pacemaker** would automatically initialize import pipelines (What is a pipeline?) for each import bunch (a bunch of **CSV** files).

Execute the following command from the root directory of your **Magento** installation, depending on which kind of import you want to execute.

Catalog import (attributes, categories, products)

Sample data set 1

The first sample data set includes all kinds of imports (attribute sets, attributes, categories, and products), which are bundled

in multiple bunches.

```
cp -R vendor/techdivision/pacemaker-import-catalog/sample-data/bunch1/* var/pacemaker/import
```

Sample data set 2

The following sample data set includes only category imports, which are bundled into two bunches.

```
cp -R vendor/techdivision/pacemaker-import-catalog/sample-data/bunch2/* var/pacemaker/import
```

Price import / inventory (stock) import

Sample data for price and inventory import includes only one import file each.

It is also possible to split up these imports into multiple files and optionally cluster them into multiple bunches like it is done for the catalog import.

Use the following command for price import:

```
cp -R vendor/techdivision/pacemaker-import-price/sample-data/bunch1/* var/pacemaker/import
```

Use the following command for inventory (stock) import:

```
cp -R vendor/techdivision/pacemaker-import-inventory/sample-data/bunch1/*
var/pacemaker/import
```

> After copying the import files into the target directory, you can observe the process by executing the pipeline:status command.
>
> Alternatively, login into the backend (**Magento's admin UI**) and open the menu:System[Pacemaker > Pipelines] Grid.

NOTE
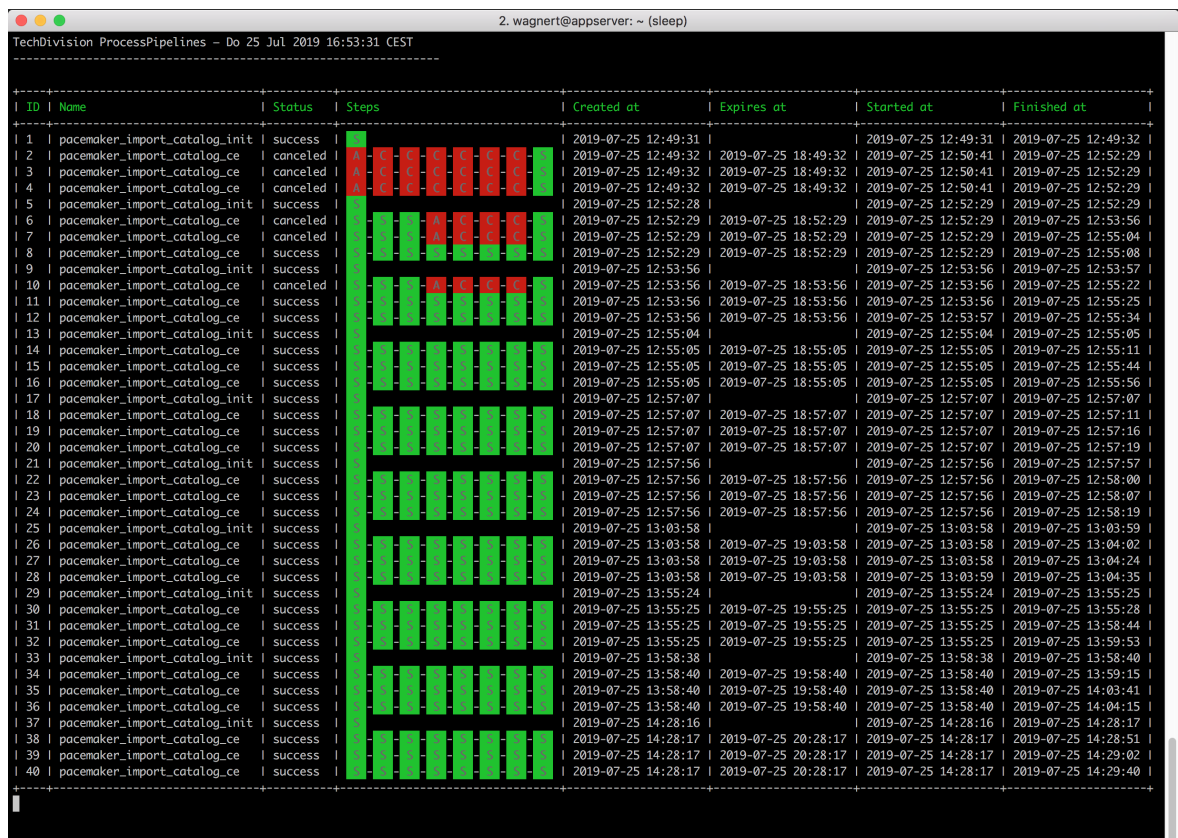
Figure 17. Result output for bin/magento pipeline:status

Reference resources

Magento resources

- Extension dev guide

- User guide - product attributes

- User guide - complex data

- User guide - customer attributes

- v2.3 Install guide

- start **MQ** consumers via cron

- Supervisor

- Supervisor configuration documentation

- specific system requirements and essentals

- **ERP/OMS**

Pacemaker resources

- Import Framework

- Import cli simple

- **Import Framework** - Magento 2 Import Framework

- Import cli simple
- **Import Framework** - Sample Data
- **Import Framework** - Product Import example CSV
- **Import Framework** - Product Import
- Shortcuts
- Media storage solution

RabbitMQ resources

- Install **RabbitMQ**
- Install **RabbitMQ** - homebrew

Other resources

- Mysql reference 8.0
- Mageplaza - how to upload product videos in **Magento 2**
- Informit - pipelines
- Wikipedia - **SPMD**
- Jenkins - pipelines
- Gitlab - pipelines
- innodb_flush_log_at_trx_commit

PHP resources

- str_pad() function description in PHP manual

Requirements

PHP Version

Compatible to PHP Version   >= 7.4

Magento

Magento version   >= 2.3

RabbitMQ

- Java Queue messaging **RabbitMQ**.
- **RabbitMQ** is an open source message broker software.

It accepts messages from producers, and delivers them to consumers and acts like a middleman which can be used to reduce loads and delivery times taken by web application servers.

RabbitMQ resources

- Install **RabbitMQ**
- Install **RabbitMQ** - homebrew = RabbitMQ

SupervisorD

Heartbeat

Runner (MQ Consumer)

Installation

Before you start the installation, a running **Magento 2** instance is required.

Please refer to Magento's documentation: Install Magento using Composer.

Install Pacemaker via Composer

After the purchase of a **Pacemaker** license, you will receive the following required credentials from our support team:

- **username**
- **password**

You need to add the received credentials to the auth.json file of your Magento instance.

If you didn't use a auth.json file before, you'd find an auth.json.sample file, copy and rename it as auth.json.

Enter our repository (gitlab.met.tdintern.de), your username and password as followed into the http-basic section of your existing auth.json file.

```
{
    "http-basic": {
        "repo.magento.com": {
            "username": "...",
            "password": "..."
        },
        "gitlab.met.tdintern.de": {
            "username": "<YOUR-TOKEN-USER>",
            "password": "<YOUR-TOKEN-PASS>"
        }
    }
}
```

Register the repository

After adding the credentials, you need to register the repository as a possible source. Therefore you need to run the following command or add the repository manually into your composer.json file.

Commandline registration:

```
composer config repositories.repo.met.tdintern.de composer https://repo.met.tdintern.de/
```

Manual entry in composer.json (optional):

```
...
"repositories": {
"repo.met.tdintern.de": {
  "type": "composer",
  "url": "https://repo.met.tdintern.de/"
},
"repo.magento.com": {
  "type": "composer",
  "url": "https://repo.magento.com/"
}
...
```

Install the module

Run composer require techdivision/pacemaker:"~1.3.0" to install the module.

Configuration

General settings

Pipeline settings

Pipelines are the core components for **Pacemaker**.

Please refer to Components & Concepts > Process Pipelines for more details.

Some settings define the overall behavior of pipelines. This setting can be found under menu:Stores[Configuration > Pacemaker > Pipeline Settings]

Working Directory

Location [global]	var/pacemaker/pipelines

Default: "var/pipelines"

Name Pattern [store view]	%pipeline_id

Default: "%pipeline_id" | Available placeholder: %pipeline_id, %year, %month, %day, %hour, %minute, %second

Mini Heartbeat

Enable mini heartbeat [global]	Yes

If enabled every finished pipeline step will cause a heartbeat for own pipeline.

*Figure 18. Configuration in **Magento** Backend*

Working directory

In this section, you may configure the location of the working directory. The path could be relative to **Magento's** root directory or absolute.

Further, you can specify how the working directory should be named.

It is possible to use following listed placeholder:

- %pipeline_id
- %year
- %month
- %day
- %hour
- %minute
- %second

Mini heartbeat

- This feature triggers a heartbeat every time a step is executed.
- This heartbeat is restricted to the one pipeline, which the concerned step belongs.

Archive pipelines

This pipeline contains two steps:

- The first step is to compress all working directories which are older than the configured period to a .tar.gz.
- The second step will delete all active directories, which are older than the configured period.

	Components & Concept
NOTE	Please refer to Process Pipelines for technical insights.

Configuration

The configurations for the periods can be done in **Magento's** admin **UI** menu:Stores[Configuration > Pacemaker > Pipeline Settings]

Figure 19. Configuration in **Magento** *Backend*

Archive pipeline

The configuration menu:Pipeline Execute Time[Clean Up] contains a cron expression value, which defines when the clean up (archive pipeline) for the working directories should run.

| BY DEFAULT, THIS VALUE IS SET TO | 3:00 am daily |

The configuration menu:Archive Cleanup[Compress Pipeline Data Older Than] defines the file compression period.

| BY DEFAULT, THIS VALUE IS SET TO | 3 months |

The configuration menu:Archive Cleanup[Delete Pipeline Data Older Than] defines the file compression period.

| BY DEFAULT, THIS VALUE IS SET TO | 6 months |

Pipeline cancel

Depending on the implementation of custom pipelines to avoid endless running pipelines, there is a cancel routine, eliminating all pipelines older than the configured time in menu:Archive Cleanup[Cancel Pipelines Older Than].

| BY DEFAULT, THIS VALUE IS SET TO | 3 weeks |

Process pipelines

Import library

Import GUI

The import **GUI** is an add-on for the **Pacemaker Professional Edition (PE)** and **Pacemaker Enterprise Edition (EE)**.

It enables shop owners or other people that work in the **Magento** backend to upload import files, e.g. in **CSV** format, creates the **.ok** file and start the import process.

It can exclusively be done by one of the Pipelines provided by the **Pacemaker Enterprise Edition (EE)**.

Configure the GUI

In case the **GUI** is used with the **Pacemaker Enterprise Edition (EE)**, the configuration can be found under the following path: menu:Stores[Configuration > TechDivision > Pacemaker > Upload] and should contain the following values

Figure 20. Configuration in Magento Backend

GUI Settings

The configuration value for Commandline should be set to the **Pacemaker CLI** that has to be used to create the **.ok** file that is necessary to start the import.

By default, this value should be set to vendor/bin/pacemaker.

As a pipeline should it be used to import the files, the configuration value for Working Directory has to be the same as for the **Pacemaker** working directory , but **without** the leading var.

The configuration for Enable CRON **must** be set to No because this will probably use the default **Magento CRON** to start the import and passing by the appropriate pipeline.

Upload files

In general, the **GUI** will provide the functionality to upload **CSV** files and adjust the settings for the import process itself, e.g. the mode that has to be used like add-update.

Depending on the **GUI** configuration and the used **Pacemaker** edition, the uploaded files will be imported directly or via one of the pipelines.

NOTE	Keep in mind, that in combination with **Pacemaker Enterprise Edition (EE)** and the configuration described above, the upload **GUI** can only be used to upload files. It is **not** possible to configure the import itself, as this will be done by the pipeline configuration that finally runs the import. Therefore, the configuration will have **no** impact.

*Figure 21. Upload **GUI** for import files*

Upload GUI

Via the drop-down Entities, the pipeline that matches the uploaded files must be selected.

If the pipeline does not match the filename, an error will be triggered.

In case the Product pipeline has been chosen, the additional drop-down Import Mode will be rendered.

Import modes

The Default import mode imports all the data that is available in the file, notwithstanding the data has changed since the last

import or not.

In contrast to this, The import mode Change-Set Detection will compare the data (using so-called and configurable changesets) from the database with the one found in the import files and only updates the **DB** if the data has been changed.

They can improve performance massively in add-update mode, as invoking **DB CRUD** steps are highly expensive in terms of performance.

NOTE	The mode Change-Set Detection will be the default mode when importing data via the default **Pacemaker** pipeline.
	It should be considered well also to activate the cache warming functionality, because this can additionally have a significant impact on performance because all the data to be compared to, will be loaded during the bootstrap process in a very efficient manner.

Additional parameters

The drop-down step provides the possibility to select the import mode. Read more about this topic in the official **Pacemaker Community Edition (CE)** Getting Started .

The checkboxes for Archive Artefacts, Debug Mode, and Single Transaction can activate the appropriate functionality.

To get more information about these flags' functionality, you can also look at the the official **Pacemaker Community Edition (CE)** Getting Started .

The checkbox Cache Enabled enables the caching functionality.

In general, caching during the import is not very helpful because the main impact on the performance will have the **CRUD** steps that can not be cached.

In case the import is processed over a network, e.g. in the cloud, it can be helpful to lower the time necessary to load the data from the **DB**.

Additionally, the cache warming functionality will only work if the cache will be enabled.

The drop-down Log Level defines the log level that has to be used.

By default, this is set to warning, but can be changed to the given requirements.

The log file named system.log will be found the directory with the import artifacts or in the archived artifact, if the appropriate checkbox has been set.

Pipeline based imports

Catalog import

The catalog import is one of the predefined pipelines (What is a pipeline?), which you can use after installing Pacemaker.

There are sample files, which can be used as a template for your data files or testing purposes (see Run your first predefined import jobs).

You will read how it works, how to configure, and how to customize this pipeline on this page.

NOTE	*Components & Concept*

Please refer to Import Processes for technical insights.

Pipeline definition

The catalog import pipeline is designed to import the whole catalog data at once.

Therefore attributes, attribute-sets, categories, and products need to be handled in the same pipeline.

However, all of these import steps are optional, and it depends on the given files, whether this data will be imported or not.

Import pipeline steps

Stage	Step	Description
Prepare	move_files	Move import files to the working directory of the current pipeline
Transformation	product_transformation	This step has a dummy executor in default and is designed to customize for mapping and transformation purpose
Pre-Import	index_suspender_start	Activates delta index suspending to avoid cron based re-indexing during the import
Attribute Set Import	attribute_set_import	Create/Update attribute sets
Attribute Import	attribute_import	Create/Update attributes
Category Import	category_import	Create/Update categories
Product Import	product_import	Create/Update products
Post-Import	index_suspender_stop	Disable suspending of delta indexers

Configuration

In Magento's backend (admin-ui) you'll find settings for the catalog import under following path: Stores > Configuration > Pacemaker > Import > Catalog Import

Figure 22. Configuration in Magento Backend

Configuration	Description	Default Value
General Settings > Source Directory	Defines the source directory for import files. The pacemaker will observe this directory to initialize an import pipeline. **This setting is for all pacemaker imports.**	var/pacemaker/import
Catalog Import > Enable Catalog Import Pipeline	Active toggle for the source directory observer for catalog import.	Yes
Catalog Import > File Name Pattern	Regular expression, which defines the source file name for source directory observer.	/(attribute-set|attribute|category|product)-import_(?P<identifier>[0-9a-z\-]*)([_0-9]*?).(csv|ok)/i

Price import

Inventory import

Headline

Line item extension

This component provides extension attributes for order items.

- It can be enabled independent of the order export component.
- It generates line item numbers, which are unique within the dedicated order.
- It is something **Magento** does not have by default but is required sometimes by **ERP/OMS**.

Configuration

The configurations for the line item extension can be executed in **Magento's** admin **UI** Stores > Configuration > Pacemaker > Order Workflow

General Settings

Figure 23. Line item extension settings

Configuration	Default Value	Description
Enable Line Items	No	Feature toggle for the whole module. If enabled, the extension attributes will be loaded for each order item.
Enable Number Generator	No	If enabled a plugin will generate unique line item numbers per order every time an order is placed
Number Offset	1	This option appears if Enable Number Generator is set to Yes. The value defines the count sequence for line item numbers. For example, the value 3 would mean the numbers will be 3, 6, 9, 12, etc.

String Padding

Configuration	Default Value	Description
Enable String Padding	No	If enabled the function str_pad will be applied to the generated number. Please refer to the str_pad() function description in PHP manual.
String Padding Length	6	Defines the number of chars to which the number will fill up
String Padding Character	0	The character, which will be used to fill up the padding
String Padding Direction	Left	In which direction the padding should be applied

Example

For **SAP** systems, we usually use the settings:

Configuration	Value
Enable Line Items	Yes
Enable Number Generator	Yes
Number Offset	10
Enable String Padding	Yes
Enable String Length	6
Enable String Character	0
Enable String Direction	LEFT

This leads in line item numbers like 000010, 000020, 000030, …

Components & Concepts

Process pipelines

One of the central components of the **Pacemaker** is the ProcessPipelines module.

The concept behind this module is the Pipeline Design Pattern.

The Pattern

To describe this pattern in short:

You need to split up each process into multiple stages. Each stage has **at least one step**.

While the stages have a precise sequence, the steps inside these stages can run in parallel or random order, since they do not depend on each other, but on the stage.

It requires a decoupling of the execution and the organization of these steps.

A runner-driven architecture is used to fulfill this requirement (SPMD).

You will find such integrations in build- and deployment tools like Jenkins or GitLab.

Realization

In the installation section, you already touched the both major parts of ProcessPipelines, while configuring the heartbeat cron job (which is the organizational unit) and configuring the pipeline runners (which are the execution unit).

Heartbeat and Runner

In the following image, you can see the responsibilities of both units.
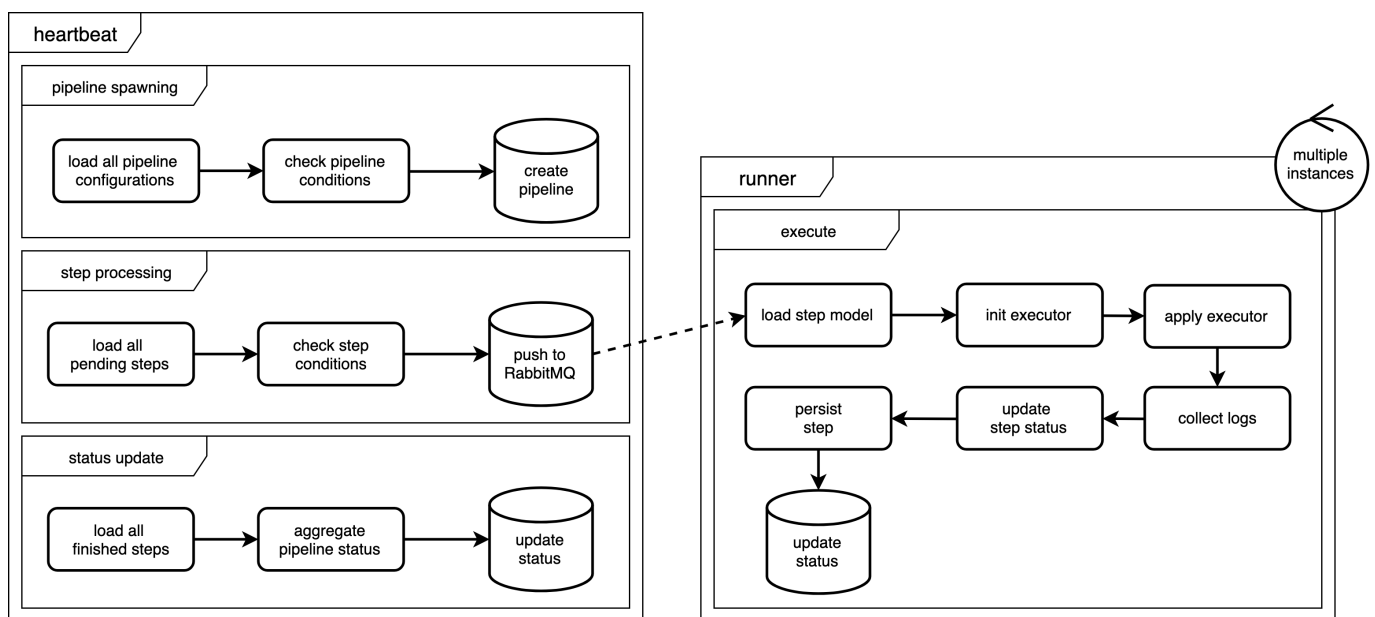
Figure 24. Heartbeat and Runner

The heartbeat is responsible for the initialization of new pipelines and publishing messages for the runner if a step can be executed. And since the status of a pipeline is an aggregation of all status of its steps, the heartbeat is also responsible for this aggregation.

The runner performs every step that is in the queue but does not care about its status or anything else. It gives us high scalability since these runners can run on different or multiple machines.

Configuration and Process Execution

The pipelines are configured in **XML** files, which are merged between the modules. It happens in the same way you already know from **Magento's** configuration XMLs (e.g. config.xml, di.xml).

BY DEFAULT, THIS VALUE IS SET TO	*Pipeline Initializer* Pipelines can also be defined by code (without **XML** files) if you need to create pipelines dynamically. Read the Pipeline initializer documentation for details.

- Please refer to the How to configure a processPipeline page.

A pipeline configuration has **at least one** condition, which will be checked periodically by the heartbeat.

If a pipeline is ready to be spawned, the heartbeat creates a new pipeline in the database.

- Please refer to the How to create a pipeline condition page.

Every step within a pipeline has **at least one** condition, which will be checked periodically by the heartbeat.

If a step is ready to be executed, the heartbeat pushes an execution message to the **RabbitMQ**.

- Please refer to the How to create a step condition page.

The next free runner will execute every message in the **RabbitMQ**.

CAUTION	**The amount of runners is at the same time the limiting factor for parallel execution.** The runner instantiates and executes executor class.

- Please refer to the How to create an executor page. = Pipeline initializer

This component is designed to instantiate new pipelines, which are not part of the own code or not defined in an **XML** file (e.g., you need a flexible count of steps).

How it works?

This module integrates into the heartbeat execution.

Every time the heartbeat is executed, a data fetcher chain (TechDivision\PacemakerPipelineInitializer\Model\InitializationDataFetcherChain), which is an instance of TechDivision\PacemakerPipelineInitializer\Api\InitializationDataFetcherInterface collects the data from all registered data fetcher, and initiates pipelines depending on the given data.

Interfaces

InitializationDataFetcherInterface

```
interface InitializationDataFetcherInterface
{
    /**
     * @return InitializationDataInterface[]
     */
    public function execute(): array;
}
```

InitializationDataFetcherInterface

```
interface InitializationDataInterface
{
    /**
     * @return array
     */
```

```php
    public function getPipelineConfiguration(): array;

    /**
     * @param array $pipelineConfiguration
     * @return void
     */
    public function setPipelineConfiguration(array $pipelineConfiguration): void;

    /**
     * @return array
     */
    public function getArguments(): array;

    /**
     * @param array $arguments
     * @return void
     */
    public function setArguments(array $arguments): void;
}
```

When to use this component?

You should use this component if you need to create pipelines dynamically.

If the steps of your pipeline are fix, you should use the XML configuration .

With this module, you can create a pipeline by defining a PHP array or by loading an existing pipeline configuration from **XML** as an array and passing it to the data fetcher chain.

You will find an example for an implementation of this approach in techdivision/pacemaker-import-base package ⬚ TechDivision\PacemakerImportBase\Model\ImportFilesDataFetcher. = How do I make an import tolerant to non-existing SKUs?

Currently, it is only possible to make the import tolerant to non-existing **SKUs** by setting debug-mode=true .

How to:

enable also importing with non-existing **SKUs**, the following entries in pipeline.xml and di.xml are necessary

Step 1: add the Debug Import Executer to the Pipeline

Activate the DebugImportExecutor with following Step in pipeline.xml:

```xml
 <step name="price_import"
executorType="<Projectname>\PriceImport\Virtual\Executor\DebugImportExecutor"
sortOrder="40"/>
```

pipeline.xml

```xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="pacemaker_import_price">
        <step name="price_import"
executorType="<Projectname>\PriceImport\Virtual\Executor\DebugImportExecutor"
sortOrder="40"/>
    </pipeline>
</config>
```

Step 2: activate the Debug mode

- Add the item debug-mode to di.xml:

```
<item name="debug-mode" xsi:type="string">true</item>
```

di.xml

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <virtualType name="<Projectname>\PriceImport\Virtual\Executor\DebugImportExecutor"
type="TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor">
        <arguments>
            <argument name="options" xsi:type="array">
                <item name="debug-mode" xsi:type="string">true</item>
            </argument>
        </arguments>
    </virtualType>
</config>
```

Usage

How to extend

Process Pipelines

How to configure a pipeline

BY DEFAULT, THIS VALUE IS SET TO

Following pages could also be interesting for you:

How to add steps to an existing pipeline How to change the executor for an existing pipeline step

To define a new pipeline, you need to add a pipeline.xml file into the etc directory of your module.

Here is some sample content for a pipeline.xml file:

```
<?xml version="1.0"?>
```

```xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="my_custom_pipeline" description="Some description" use-working-
directory="true">
        <conditions>
            <pipeline_condition
type="MyCompany\MyModule\Helper\Condition\Pipeline\CheckSomeThing" description="Some
description"/>
        </conditions>
        <step name="my_first_step"
executorType="MyCompany\MyModule\Model\Executor\DoSomeThing" sortOrder="10" description="" >
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit5"
description="Try step up to 5 times"/>
            </conditions>
        </step>
        <step name="my_second_step"
executorType="MyCompany\MyModule\Model\Executor\DoSomeMoreStuff" sortOrder="20"
description="" >
            <arguments>
                <argument key="some_key" value="some_value" />
            </arguments>
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Run after the first step"/>
                <step_condition
type="MyCompany\MyModule\Helper\Condition\Step\CheckSomeThing" description="Check
something..."/>
            </conditions>
        </step>
        <step name="my_third_step"
executorType="MyCompany\MyModule\Model\Executor\DoClearingStuff" sortOrder="30"
description="" runAlwaysStep="true">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try step up to 1 times"/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsFinished"
description="Run always when one step started"/>
            </conditions>
        </step>
    </pipeline>
</config>
```

Description for XML nodes and attributes

Node	Description
pipeline	Definition of a new pipeline configuration. You can overwrite or extend existing pipelines by using the same name.
-- name	**Required**, string; Unique identifier, which is used for instantiating a pipeline by CLI and can be used for condition rules, etc.
-- description	**Optional**, string; Description text for the pipeline. Will be displayed in UI and CLI as 'name' of the pipeline.
-- use-working-directory	**Optional**, boolean; If defined, a working directory will be autogenerated while pipeline instantiation. The path for this directory is configurable and can be retrieved within an executor by $step->getWorkingDir().
pipeline/conditions	**Optional**; One or multiple conditions which should be true to cause a new pipeline execution
pipeline/conditions/pipeline_condition	Configuration of a pipeline condition
-- type	**Required**, string; Defines the class, which will be executed to run the condition check.
-- description	**Optional**, string; Description text for given condition. It will be displayed on UI, logs, CLI, etc.
pipeline/step	Each pipeline has at least one step. The step describes which executor should be run once the step conditions are true
-- name	**Required**, string; Unique identifier, which is used for running/executing the step by CLI and can be used for condition rules, etc.
-- executorType	**Required**, string; Defines the class, which will be executed to run the step.
-- sortOrder	**Required**, string; Defines the sorting of steps. Useful for resorting to the steps by other modules/extensions.
-- description	**Optional**, string; Description text for given step. Will be displayed on UI, logs, CLI, etc.
-- runAlwaysStep	**Optional**, boolean; If this flag is set, the step will be executed in any case. Even if the pipeline is canceled or abandoned. Kind of 'finally' step.
pipeline/step/arguments	**Optional**; Holds one or many executor arguments
pipeline/step/arguments/argument	Step executor argument
-- key/value	**Required**, string; Since arguments are array at all, both arguments represent the key and value of each array node.
pipeline/step/conditions	One or multiple conditions that should be true to cause the step execution.
pipeline/step/conditions/step_condition	Configuration of a step condition
-- type	**Required**, string; Defines the class, which will be executed to run the condition check.

Node	Description
-- description	**Optional**, string; Description text for given condition. It will be displayed on UI, logs, CLI, etc.

NOTE	There are already several default conditions and executors available therefore.

How to create a pipeline condition

Pipeline Conditions

Each pipeline has at least one condition, which implements the TechDivision\ProcessPipelines\Api\PipelineConditionInterface interface.

Existing Conditions

There are some ready-to-use conditions.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn
Disables auto spawning of given pipeline. Pipelines, which use this condition can not be spawned by heartbeat.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoSiblingInProgress
Disable spawning of the given pipeline, while this pipeline is already in progress.

Usage

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline">
        <conditions>
            <pipeline_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn"
description="Disable auto spawning"/>
        </conditions>
        ...
    </pipeline>
</config>
```

Provide arguments from condition to steps

Since version 1.1.0 there is an additional interface (TechDivision\ProcessPipelines\Api\ArgumentProviderInterface) for conditions, which enables you to provide arguments from the pipeline condition all pipeline steps.

Reasons

Imagine you have a condition, which checks the file system for a specific file pattern. If the file is present, you need to run a new pipeline for exactly this file.

Therefore you need to provide the information to your steps, which should handle the file.

Usage Examples

```php
<?php

use TechDivision\ProcessPipelines\Api\PipelineConditionInterface;
use TechDivision\ProcessPipelines\Api\ArgumentProviderInterface;

class FileExistsCondition implements PipelineConditionInterface, ArgumentProviderInterface
{
    const SOME_MAGIC_FILE_PATTERN = '...';

    public function isReady(array $pipelineConfiguration)
    {
        $this->searchForFiles(self::SOME_MAGIC_FILE_PATTERN);
        return $this->hasFiles();
    }

    public function getArguments()
    {
        return $this->getFiles();
    }
}
```

Virtual Conditions

By using the virtualType feature of Magento's DI it is easy to create new conditions with some new params. Therefore there are some "template" conditions.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\CronExpression

    Defines execution time for a pipeline using regular cron expression.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\ConfigurableCronExpression

    Defines execution time for a pipeline using regular cron expression, while this expression can be configured in Magento admin.

Usage

Create virtual spawn conditions using di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
```

```xml
    ...
    <virtualType name="MyCompany\MyModule\Virtual\Condition\EveryNight"
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\CronExpression">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="cron_expression" xsi:type="string">0 2 * * *</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\FullImport"
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\ConfigurableCronExpression">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="config_path"
xsi:type="string">my_module/crontab/full_import</item>
            </argument>
        </arguments>
    </virtualType>
    ...
</config>
```

Use the virtual condition in pipeline.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Some example pipeline">
        <conditions>
            <pipeline_condition type="MyCompany\MyModule\Virtual\Condition\EveryNight"
description="Run once in the night"/>
        </conditions>
        ...
    </pipeline>
    <pipeline name="another_example_pipeline" description="Some example pipeline">
        <conditions>
            <pipeline_condition type="MyCompany\MyModule\Virtual\Condition\FullImport"
description="Customer defined cron expression"/>
        </conditions>
        ...
    </pipeline>
</config>
```

How to create a step condition

Each step can have conditions, which implement the TechDivision\ProcessPipelines\Api\StepConditionInterface interface.

Existing Conditions

There are some ready-to-use conditions.

TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted
> Run the given step only if the previous steps are successfully finished.

TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsFinished
> Run the given step only if the previous steps are not in a pending, running or enqueued state.

Usage

NOTE	See How to configure a pipeline for more details about the XML structure.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline">
        ...
        <step name="my_third_step"
executorType="MyCompany\MyModule\Model\Executor\DoSomeThing" sortOrder="1522" description=""
runAlwaysStep="">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Run only after previous"/>
            </conditions>
        </step>
        ...
    </pipeline>
</config>
```

Virtual Conditions

Template Conditions

By using the virtualType feature of **Magento's** DI it is easy to create new conditions with some new params.

Therefore there are some "template" conditions.

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit
> Limitation for repeats for given steps.

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts
> Timeout between retries.

TechDivision\ProcessPipelines\Helper\Condition\Step\WaitForStepIsNotRunning
> Disables execution of a step while the defined steps (even in other pipelines) are running. For example only one import

process at the same time possible…

TechDivision\PacemakerImportBase\Model\Condition\Step\NoConflictingStepsInProcess

The list of conflicting step names can be set via **DI**. Verifies that these steps are not running or enqueued. This check works overall existing pipelines.

TechDivision\PacemakerImportBase\Model\Condition\Step\IsStageReady

This condition was designed to verify whether a stage is ready to be executed.

Ready to use virtual conditions

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1

Attempts limit 1

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit2

Attempts limit 2

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit5

Attempts limit 5

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit10

Attempts limit 10

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes5

5 minutes delay between attempts

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes15

15 minutes delay between attempts

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes30

30 minutes delay between attempts

Usage

BY DEFAULT, THIS VALUE IS SET TO

In following example module virtual conditions are used to allow parallel execution.

Create virtual spawn conditions using di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType name="MyCompany\MyModule\Virtual\Condition\AttemptsLimit42"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="limit" xsi:type="string">42</item>
            </argument>
        </arguments>
    </virtualType>
```

```
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\TimeBetweenAttempts\Minutes42"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="minutes" xsi:type="string">42</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\AllDownloadsAreReady"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="step_filter"
xsi:type="string">download_media_step,download_csv_step</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\NoImportIsRunning"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\WaitForStepIsNotRunning">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="step_names"
xsi:type="string">import_stock_step,import_products_step</item>
            </argument>
        </arguments>
    </virtualType>
    ...
</config>
```

Use the virtual condition in pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline">
        ...
        <step name="some_step" executorType="MyCompany\MyModule\Model\Executor\DoSomeThing"
sortOrder="10" description="" >
            <conditions>
                <step_condition type="MyCompany\MyModule\Virtual\Condition\AttemptsLimit42"
description="Retry up to 42 times"/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes5"
description="Wait 5 minutes between attempts"/>
                <step_condition
type="MyCompany\MyModule\Virtual\Condition\AllDownloadsAreReady" description="Start only if
downloads are ready"/>
```

```
            <step_condition
type="MyCompany\MyModule\Virtual\Condition\NoImportIsRunning" description="Ensure no other
import process is active"/>
        </conditions>
      </step>
      ...
    </pipeline>
</config>
```

How to create an executor

Pipeline Step Executor

Each pipeline step has one executor, which implements the interface. \TechDivision\ProcessPipelines\Api\ExecutorInterface.

Abstract Executors

There are some abstract executors, which should be extended by your executor.

TechDivision\ProcessPipelines\Model\Executor\AbstractExecutor
    Base executor, which already implements methods for logging and warnings

TechDivision\ProcessPipelines\Model\Executor\AbstractShellExecutor
    Shell executor for run CLI and bash scripts.

Concrete Executors

There are some *ready to use* executors.

TechDivision\ProcessPipelines\Model\Executor\DropCache
    Cache invalidation executor

TechDivision\ProcessPipelines\Model\Executor\Reindex
    Reindex executor

Usage

To use these executors, you need to define them as following in your pipeline.xml.

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline for reindex and drop
cache">
        <conditions>
            <pipeline_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn"
```

```
description="Disable auto spawning"/>
        </conditions>
        <step name="example_reindex"
executorType="TechDivision\ProcessPipelines\Model\Executor\Reindex" sortOrder="10"
description="" >
            <arguments>
                <argument key="indexes"
value="catalog_category_product,catalog_product_category,catalogsearch_fulltext" />
            </arguments>
        </step>
        <step name="example_cache_drop"
executorType="TechDivision\ProcessPipelines\Model\Executor\DropCache" sortOrder="20"
description="" >
            <arguments>
                <argument key="caches" value="collections,full_page,block_html" />
            </arguments>
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Run after the first step"/>
            </conditions>
        </step>
    </pipeline>
</config>
```

Testing executors on a local environment (DEV)

While developing a new executor, it is necessary to execute them without waiting for heartbeat and without having some runner up and running (see [Cron and Consumer/Runner](./system-cron-consumer.md)).

Therefore we introduce the **CLI** command pipeline:dev:run:executor.

Usage

Run a customer executor in the same way as the runner would do.

```
bin/magento pipeline:dev:run:executor --arguments='{\"arg1\": \"value-1\", \"arg2\": 2}'
--pipeline-id=102 --step-id=1562 '\MyCompany\MyModule\Model\Executor\DoSomeThing'
```

The options --arguments, --pipeline-id and --step-id are optional. Sometimes there are dependencies inside your executor for these values. = How to deactivate a pipeline

In some cases, e.g. if a custom process is necessary, it will be useful to deactivate the default pipelines that will be provided by the modules

- TechDivision_PacemakerImportCatalog

- TechDivision_PacemakerImportInventory

- TechDivision_PacemakerImportPrice

They are part of the **Pacemaker** distribution.

Usage

To deactivate the default Pipelines, disable the appropriate modules, assuming that you are in the **Magento** root directory, from the command-line with

```xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <process_pipelines>
            <mini_heartbeat>
                <enabled>1</enabled>
            </mini_heartbeat>
        </process_pipelines>
        <techdivision_pacemaker_import>
            <general>
                <source_directory>../import</source_directory>
                <media_source_directory>../import/media</media_source_directory>
            </general>
            <catalog>
                <!-- Disable default pacemaker pipeline -->
                <enabled>0</enabled>
                <!-- Enable project specific import pipeline -->
                <my_project_pipeline_enabled>1</my_project_pipeline_enabled>
                <ready_file_name>import.ok</ready_file_name>
            </catalog>
            <price>
                <enabled>0</enabled>
            </price>
            <inventory>
                <enabled>0</enabled>
            </inventory>
        </techdivision_pacemaker_import>
    </default>
</config>
```

and invoke bin/magento setup:upgrade again.

Import

Extend and override default functionality

The **Pacemaker** import functionality is designed to work standalone. But, up from version 3.8.0, it is possible to use the **Magento** code directory <magento-install-dir>/app/code/ as well to extend the **Pacemaker** import functionality without the need to deploy it as a composer library.

Override existing classes

In some cases, it will be necessary to override a default class of the **Pacemaker** import library.

For example, if additional attributes have been added to a **non-EAV** entity or the import should keep going if a website that has been referenced in the **CSV** file is not available in the **Magento** instance.

A minimum **Magento** module has to be created.

In every project that uses **Pacemaker** the module will be named Import, e.g. MyProject\Import.

At this point, the directory structure should look like

```
<magento-install-dir>/
 --app/
   --code/
     --MyProject/
       --Import/
         |--registration.php
         --etc/
           --di.xml
           --config.xml
           --module.xml
           --adminhtml/
               --system.xml
```

As the **Pacemaker** import functionality is completely based on **Symfony**, at least a **DI** configuration enables us to override the default class.

The configuration file **must** be located in the directory <magento-install-dir>/app/code/MyProject/Import/symfony/Resources/config/services.xml.

```
<magento-install-dir>/
 --app/
   --code/
     --MyProject/
       --Import/
         |--registration.php
         |--etc/
         |   --di.xml
         |   --config.xml
         |   --module.xml
         |   --adminhtml/
         |       --system.xml
         |--Observers/
         |   --ProductWebsiteObserver.php
          --symfony/
            --Resources/
              --config/
                  --services.xml
```

The simple DI configuration that we used to override the original observer will have the following content.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">
    <services>
        <service
            id="import_product.observer.product.website"
            class="MyProject\Import\Product\Observers\ProductWebsiteObserver"/>
    </services>
</container>
```

The corresponding observer can extend the class that has to be overwritten or completely implement the custom business logic. Assuming that the original class will be extended, this can look like

```php
namespace MyProject\Import\Product\Observers;

/**
 * Observer that creates/updates the product's website relations.
 */
class ProductWebsiteObserver extends
\TechDivision\Import\Product\Observers\ProductWebsiteObserver
{

    /**
     * Process the observer's business logic.
     *
     * @return array The processed row
     */
    protected function process()
    {
        // custom code here
    }
}
```

Add custom functionality

Custom functionality can be added the same way. If functionality has to be extended, it'll also be necessary to add the workflow engine's configuration. The workflow engine configuration can either be done be snippets located in the global configuration directory under <magento-install-dir>/app/etc/configuration or in the configuration directory of the module, which will be app/code/MyProject/Import/etc then.

NOTE Everything else will **NOT** be different than writing an extension that will be deployed in the Magento vendor directory.

Register the module

Finally, to make the module available for usage in the **Pacemaker** import functionality, it must be registered in the workflow engine's configuration. To do so, add a snippet <magento-install-dir>/app/code/configuration/additional-vendor-dirs.json which has the following content

```
{
    "additional-vendor-dirs": [
        {
            "vendor-dir": "app/code",
            "libraries": [
                "MyProject/Import"
            ]
        }
    ]
}
```

Import processes

Transform foreign import sources

Often the import files are not ready to be imported when we receive them from the foreign systems.

Commonly, there is data missing, which needs to be filled with **Magento** insides like category paths or website codes. There is a transformation step inside the given import pipelines.

See the example, where we will replace the default executor with custom logic.

Manipulate the pipeline

By creating a own pipeline.xml in the etc folder of our module, we can create own pipelines and overwrite existing.

In this example, we want to change the executor of the step product_transformation step within the pacemaker_import_catalog pipeline.

Content of app/code/MyModule/CustomCatalogImport/etc/pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="pacemaker_import_catalog">
        <step
                name="product_transformation"

executorType="MyModule\CustomCatalogImport\Model\Executor\TransformationExecutor"
                sortOrder="40"
        />
    </pipeline>
```

```
</config>
```

We creating a pipeline node with the name "pacemaker_import_catalog" and add here a single step node with the name "product_transformation".

Both nodes are merged between the pipelines. By defining an own executorType we will change the previous defined executor since the sequence in the module.xml of our module gives our module higher priority.

Transformation logic

Once we created our module and changed the executor for the transformation step, we need to make the executor class. In the previous part, we already chose the class name MyModule\CustomCatalogImport\Model\Executor\TransformationExecutor, which means we need to create the file app/code/MyModule/CustomCatalog/Import/Model/Executor/TransformationExecutor.php according to the **PSR-4** autoloader.

BY DEFAULT, THIS VALUE IS SET TO	We recommend the directory structure Model/Executor, Model/Condition/Step and Model/Condition/Pipeline for the corresponding classes.

Place the following code; we already have a valid executor.

```php
<?php
declare(strict_types=1);

namespace MyModule\CustomCatalogImport\Model\Executor;

use TechDivision\ProcessPipelines\Api\StepInterface;
use TechDivision\ProcessPipelines\Model\Executor\AbstractExecutor;

class TransformationExecutor extends AbstractExecutor
{
    /**
     * @inheritdoc
     */
    public function process(StepInterface $step): void
    {

    }
}
```

NOTE	In order to test your current implementation you can add following code into the body of your process method and execute the pipelines .
	``` $this->getLogger()->info("It works ;)"); ```
	Use the **CLI** to observe the progress of your pipeline (bin/magento pipeline:status -w 2).
	See the details for your pipeline with bin/magento pipeline:detail <PIPELINE_ID> and retrieve the log for your transformation step with bin/magento pipeline:step:show <STEP_ID>. The result should be something like

```
--- ATTEMPT 1 ---
[2020-02-11 11:10:40] executor.INFO: It works ;)
```

Now you're able to add your custom logic to this executor to transform the given files.

The file paths for this import pipeline are present in the $step as arguments.

Execute $files = (array)$step->getArgumentValueByKey('files'); to fetch a list of file paths.

Examples

Checkout following resource for an example module.

This example shows how to implement a mapping layer, which transforms data from **CSV** to **CSV** before running the import.

Add Steps to an existing Pipeline

Sometimes it is required to add additional steps to an existing import pipeline. Use cases could be

- Add re-indexations and cache invalidation steps
- Add cache warming processes
- Add image cropping executions
- and others

See how to create a own module

Manipulate the pipeline

By creating a own pipeline.xml in the etc folder of our module, we are able to create own pipelines and extend existing.

In this example, we want to add additional steps to the existing pacemaker_import_catalog pipeline.

Content of app/code/TechDivision/CustomCatalogImport/etc/pipeline.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="pacemaker_import_catalog">
        <step name="reindex_attributes"
executorType="TechDivision\ProcessPipelines\Model\Executor\Reindex" sortOrder="100"
description="Run full reindex">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
                <step_condition
type="TechDivision\AddNewStepsToExistingPipeline\Virtual\Condition\Step\IsReindexingStage"
description="Ensure it is time for reindexing."/>
```

```
                </conditions>
        </step>
        <step name="drop_cache"
executorType="TechDivision\ProcessPipelines\Model\Executor\DropCache" sortOrder="110"
description="Drop relevant caches">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Previous step needs to be finished."/>
            </conditions>
        </step>
    </pipeline>
</config>
```

We create a pipeline node with the name "pacemaker_import_catalog" and add here two step nodes.

With the sortOrder attribute we define the position within our pipeline.

In this example, both steps will run as last.

Examples

Checkout following resource for an example module(**Pacemaker**) Custom Catalog Import Add New Steps To Existing Pipeline). = Create an additional import process

Sometimes we need additional imports besides the given.

In this example, we create our import pipeline, which observes the import directory for files and executes the import without writing any PHP code, but re-using existing executors.

BY DEFAULT, THIS VALUE IS SET TO You will find all this code also in this example module.

Create an own module

As the first step, we need to introduce a custom module. Please refer to Create a New Module in **Magento's** developer documentation.

In this example, we name the module MyModule_ProductStatusUpdate.

Our module's primary purpose will be to observe the import directory for a file with a specific name pattern. If this file is given we will introduce an import with the **Import Library**.

```
mkdir -p app/code/MyModule/ProductStatusUpdate/etc
touch app/code/MyModule/ProductStatusUpdate/etc/module.xml
touch app/code/MyModule/ProductStatusUpdate/registration.php
```

Content of app/code/MyModule/ProductStatusUpdate/registration.php

```php
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_ProductStatusUpdate',
    __DIR__
);
```

Content of app/code/MyModule/ProductStatusUpdate/etc/module.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="MyModule_ProductStatusUpdate" setup_version="1.0.0">
        <sequence>
            <module name="TechDivision_PacemakerImportBase"/>
        </sequence>
    </module>
</config>
```

Since we will use components from TechDivision_PacemakerImportBase module, we need to add this module to the loading sequence of our new module.

Define the import pipeline

We create a pipeline.xml file within the etc folder of our module and add following content:

app/code/MyModule/ProductStatusUpdate/etc/pipeline.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="product_status_update_import" description="Example Pipeline for a custom
product update import" use-working-directory="true">
        <conditions>
            <pipeline_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn" description="No
automatic start for this pipeline"/>
        </conditions>
        <step name="move_files"
executorType="TechDivision\PacemakerImportBase\Model\Executor\MoveFilesToWorkingDirectory"
sortOrder="10" description="Move files to working directory.">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
            </conditions>
        </step>
        <step name="product_status_update"
```

```
executorType="TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor" sortOrder="20"
description="Import product data">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Previous step needs to be finished."/>
                <step_condition
type="MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess"
description="Avoid conflicts between import steps."/>
            </conditions>
            <arguments>
                <argument key="command" value="import:products" />
                <argument key="operation" value="add-update" />
                <argument key="configuration" value="MyModule_ProductStatusUpdate::etc/m2if-
config.json" />
            </arguments>
        </step>
    </pipeline>
</config>
```

In this sample, we create a pipeline with two steps.

- The first step moves the relevant files to the working directory.
- The second step executes the import.

But let's go thru the code step-by-step.

Also check out the XML config documentation for deeper insides.

Pipeline definition

The first node of our **XML** file defines a new pipeline.

```
<pipeline
    name="product_status_update_import"
    description="Example Pipeline for a custom product update import"
    use-working-directory="true" />
```

With the attribute name, we give our pipeline a unique name. If there is already a pipeline with the same name that exists within our magento instance they will be merged.

Please refer to Transform foreign import source and Add steps to an existing pipeline to learn how to use this behaviour to extend existing pipelines.

The attribute description contains a human-readable description of the purpose of our pipeline. This content will be used in the admin **UI**.

Since our new pipeline works with files, we need a working directory for every instance of our pipeline.

Therefore we add the attribute use-working-directory and set it to true.

Pipeline conditions

With the `pipeline_condition`s, we define when our pipeline should be initialized.

NOTE | Plese refer to How to create a pipeline condition for details.

```
<pipeline>
    <conditions>
        <pipeline_condition
            type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn"
            description="No automatic start for this pipeline"/>
    </conditions>
</pipeline>
```

We do not want to have periodical instantiation for our product updates since we want to run the import once the required import file is present in the file system.

Therefore we will use the pipeline initializer .

To avoid an instantiation via the heartbeat we need to add the condition TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn.

Move files step

Our first step will move the relevant import files to the working directory of our pipeline.

```
<step
    name="move_files"

executorType="TechDivision\PacemakerImportBase\Model\Executor\MoveFilesToWorkingDirectory"
    sortOrder="10"
    description="Move files to working directory.">

    <conditions>
        <step_condition
            type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
            description="Try once."/>
    </conditions>
</step>
```

We name our step (e.g. move_files).

Step names are relevant if we need to specify parallel and sequential execution. As executor we use the already existing class`TechDivision\PacemakerImportBase\Model\Executor\MoveFilesToWorkingDirectory`.

Please refer to How to create an executor

The sortOrder of our step is essential since we want to execute the steps in a sequence (not parallel).

Since this step is the first in our pipeline, it does not require any specific sequence relevant conditions.

By adding the condition TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1 we ensure that our pipeline will be canceled if errors appear while moving the files to the working directory.

Import data step

The second step will execute the import.

We need to ensure this step is running after the first. And we need to check whether other imports are probably already running and wait until they are finished to avoid deadlocks in the database.

```
<step
    name="product_status_update"
    executorType="TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor"
    sortOrder="20"
    description="Import product data">

    <conditions>
        <step_condition
            type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
            description="Try once."/>
        <step_condition

type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
            description="Previous step needs to be finished."/>
        <step_condition

type="MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess"
            description="Avoid conflicts between import steps."/>
    </conditions>

    <arguments>
        <argument key="command" value="import:products" />
        <argument key="operation" value="add-update" />
        <argument key="configuration" value="MyModule_ProductStatusUpdate::etc/m2if-
config.json" />
    </arguments>
</step>
```

We name the step and define an executor. To run **Import Framework** based import, we can re-use the class TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor.

As sortOrder we need to define a higher number than for the move_files step since we want to use the condition TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted. This condition verifies whether steps with lower sortOrder are successfully executed.

To avoid endless retries of this step in case of errors, we need to add the TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1 as we did it for the first step.

As the third condition for this step, we add

MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess. This condition does not exist yet, but we will create it in the next part of this documentation.

The import executor expects a set of arguments, which defines the command and operation to execute as well as the path to the configuration, we want to pass to the **Import Library**.

BY DEFAULT, THIS VALUE IS SET TO	Take a look into the **Import Library** documentation for a better understanding of command, operation and configuration.

Avoid execution of conflicting steps

In the previous part of this documentation, we've already defined the step condition MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess.

Now we need to create this class.

The namespace of this class contains Virtual. This class will be auto-generated by **Magento**.

Therefore we create a di.xml in our module and add the following content into it.

app/code/MyModule/ProductStatusUpdate/etc/di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <virtualType
name="MyModule\ProductStatusUpdateImport\Virtual\Condition\NoConflictingStepInProcess"
type="TechDivision\PacemakerImportBase\Model\Condition\Step\NoConflictingStepsInProcess">
        <arguments>
            <argument name="stepNames" xsi:type="array">
                <item name="product_status_update"
xsi:type="string">product_status_update</item>
                <item name="product_import" xsi:type="string">product_import</item>
            </argument>
        </arguments>
    </virtualType>
</config>
```

BY DEFAULT, THIS VALUE IS SET TO	Refer to Magento DevDocs for details about virtual types.

This approach allows us to extend the list of conflicting processes.

Our condition checks now for active import steps from our pipeline as well as for the import step from the pacemaker_import_catalog pipeline.

We also should extend the list of conflicting steps for the pacemaker_import_catalog pipeline, by adding the following code to the di.xml of our module.

app/code/MyModule/ProductStatusUpdate/etc/di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```xml
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType
name="TechDivision\PacemakerImportCatalog\Virtual\Condition\NoConflictingStepInProcess">
        <arguments>
            <argument name="stepNames" xsi:type="array">
                <item name="product_status_update"
xsi:type="string">product_status_update</item>
            </argument>
        </arguments>
    </virtualType>
</config>
```

Import Framework configuration file

Now we need to add the **Import Framework** configuration file to the path we pass to our import executor (MyModule_ProductStatusUpdate::etc/m2if-config.json).

The content of this configuration file looks as following:

app/code/MyModule/ProductStatusUpdate/etc/m2if-config.json

```json
{
    "magento-edition": "CE",
    "magento-version": "2.3.4",
    "operation-name" : "add-update",
    "archive-artefacts" : false,
    "debug-mode" : false,
    "entity-type-code" : "catalog_product",
    "listeners" : [
        {
            "app.set.up" : [
                "import.listener.render.ansi.art",
                "import.listener.initialize.registry"
            ]
        },
        {
            "app.tear.down" : [
                "import.listener.clear.registry"
            ]
        }
    ],
    "databases" : [],
    "loggers": [
        {
            "name": "system",
            "channel-name": "logger/system",
            "type": "Monolog\\Logger",
            "handlers": [
                {
```

```
                    "type": "Monolog\\Handler\\ErrorLogHandler",
                    "formatter": {
                        "type": "Monolog\\Formatter\\LineFormatter",
                        "params" : [
                            {
                                "format": "[%datetime%] %channel%.%level_name%: %message%
%context% %extra%",
                                "date-format": "Y-m-d H:i:s",
                                "allow-inline-line-breaks": true,
                                "ignore-empty-context-and-extra": true
                            }
                        ]
                    }
                }
            ],
            "processors": [
                {
                    "type": "Monolog\\Processor\\MemoryPeakUsageProcessor"
                }
            ]
        }
    ],
    "operations" : [
        {
            "name" : "add-update",
            "plugins" : [
                {
                    "id": "import.plugin.cache.warmer"
                },
                {
                    "id": "import.plugin.global.data"
                },
                {
                    "id": "import.plugin.subject",
                    "subjects": [
                        {
                            "id": "import.subject.move.files",
                            "identifier": "move-files",
                            "file-resolver": {
                                "prefix": "product-status-update"
                            },
                            "ok-file-needed": true
                        },
                        {
                            "id": "import_product.subject.bunch",
                            "identifier": "files",
                            "file-resolver": {
                                "prefix": "product-status-update"
                            },
```

```
                    "params" : [
                        {
                            "copy-images" : false,
                            "clean-up-empty-columns" : [
                                "base_image",
                                "small_image",
                                "swatch_image",
                                "thumbnail_image",
                                "special_price",
                                "special_price_from_date",
                                "special_price_to_date"
                            ]
                        }
                    ],
                    "observers": [
                        {
                            "import": [
                                "import.observer.attribute.set",
                                "import.observer.additional.attribute",
                                "import_product.observer.product",
                                "import_product.observer.product.attribute.update"
                            ]
                        }
                    ]
                }
            ]
        },
        {
            "id": "import.plugin.archive"
        }
    ]
  }
 ]
}
```

With this configuration, we define an import, which updates attributes for already existing products.

Please refer to **Import Framework** documentation for details.

Pipeline instantiation

In the last part of this documentation, we teach the pipeline-initializer to observe the import directory for our import files and instantiates our pipeline once the file is present.

All this can be entirely done in a declarative way.

System configuration

Let's add some configuration options to **Magento's** admin **UI**.

We create a system.xml in the etc/adminhtml directory of our module an adds the following content into this file.

app/code/MyModule/ProductStatusUpdate/etc/adminhtml/system.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
    <system>
        <section id="techdivision_pacemaker_import">
            <group id="product_status_update" showInDefault="1" showInStore="0"
showInWebsite="0" sortOrder="1000" translate="label">
                <label>Product Status Update</label>
                <field id="enabled" translate="label" type="select" sortOrder="25"
showInDefault="1" showInWebsite="0" showInStore="0" canRestore="0">
                    <label>Enable Pipeline</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
                <field id="file_name_pattern" translate="label comment" type="text"
sortOrder="30" showInDefault="1" showInWebsite="0" showInStore="0" canRestore="0">
                    <label>File Name Pattern</label>
                    <comment>Pattern for import file bunches</comment>
                </field>
            </group>
        </section>
    </system>
</config>
```

It will add a new configuration group to menu:Stores[Configuration > Pacemaker > Import] with two fields.

One to enable/disable our pipeline and a second one, which defines the file name pattern for our import files.

Create now a default configuration by creating following file:

app/code/MyModule/ProductStatusUpdate/etc/config.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <techdivision_pacemaker_import>
            <product_status_update>
                <enabled>1</enabled>
                <file_name_pattern><![CDATA[/product-status-update_(?P<identifier>[0-9a-z\-
]*)([_0-9]*?).(csv|ok)/i]]></file_name_pattern>
            </product_status_update>
        </techdivision_pacemaker_import>
    </default>
</config>
```

It will enable our module by default and pre-defines a file name pattern.

Extend Pipeline Initializer

Add the following content to the di.xml of our module.

app/code/MyModule/ProductStatusUpdate/etc/di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType
name="MyModule\ProductStatusUpdateImport\Virtual\ConfigProvider\GetFileNamePattern"
type="TechDivision\PacemakerImportBase\Model\ConfigProvider\GetFileNamePattern">
        <arguments>
            <argument name="scopeConfigPath"
xsi:type="string">techdivision_pacemaker_import/product_status_update/file_name_pattern</arg
ument>
        </arguments>
    </virtualType>
    <virtualType name="MyModule\ProductStatusUpdateImport\Virtual\ImportBunchResolver"
type="TechDivision\PacemakerImportBase\Model\ImportBunchResolver">
        <arguments>
            <argument name="getFileNamePattern"
xsi:type="object">MyModule\ProductStatusUpdateImport\Virtual\ConfigProvider\GetFileNamePatte
rn</argument>
        </arguments>
    </virtualType>
    <type name="TechDivision\PacemakerImportBase\Model\ImportFilesDataFetcher">
        <arguments>
            <argument name="resolverConfig" xsi:type="array">
                <item name="my_module.product_status_update_import" xsi:type="array">
                    <item name="resolver"
xsi:type="object">MyModule\ProductStatusUpdateImport\Virtual\ImportBunchResolver</item>
                    <item name="validator"
xsi:type="object">TechDivision\PacemakerImportBase\Api\ImportBunchValidatorInterface</item>
                    <item name="pipeline_name"
xsi:type="string">product_status_update_import</item>
                    <item name="enable_config_path"
xsi:type="string">techdivision_pacemaker_import/product_status_update/enabled</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

With this configuration, we register an additional virtual type (auto-generated class)
MyModule\ProductStatusUpdateImport\Virtual\ConfigProvider\GetFileNamePattern. This class retrieves the file pattern to
our "ImportBunchResolver". If you do not want a configurable file pattern, you could also implement

your implementation for TechDivision\PacemakerImportBase\Api\GetFileNamePatternInterface and use this instead of the
virtual type.

The "ImportBunchResolver" is also a virtual type and requires our previously defined FileNamePattern provider.

The third node of the XML code above extends the resolverConfig array of the class TechDivision\PacemakerImportBase\Model\ImportFilesDataFetcher.

The new item we add to this array requires at least three items:

- resolver: This is an instance of TechDivision\PacemakerImportBase\Api\ImportBunchResolverInterface. We add here our virtual type, which we defined before.
- validator: This is an instance of TechDivision\PacemakerImportBase\Api\ImportBunchValidatorInterface. Since we do not add any custom validation logic, we can add the interface directly, and Magento's DI will instantiate the default preference for it.
- pipeline_name: Our pipeline's name should be instantiated once the resolver found import files, and the validator approved them.

The fourth item in this array is optional. With enable_config_path we can add the configuration path to our feature toggle.

It allows us to disable the instantiation of this pipeline.

Examples

You will find an example module, which also provides sample **CSV** files to test this import.

Create a own module

As the first step, we need to introduce a custom module.

Please refer to Create a New Module in **Magento's** developer documentation.

In this example, we name the module MyModule_CustomCatalogImport.

```
mkdir -p app/code/MyModule/CustomCatalogImport/etc
touch app/code/MyModule/CustomCatalogImport/etc/module.xml
touch app/code/MyModule/CustomCatalogImport/registration.php
```

Content of registration.php

```php
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_CustomCatalogImport',
    __DIR__
);
```

Content of module.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="MyModule_CustomCatalogImport" setup_version="1.0.0">
        <sequence>
```

```
            <module name="TechDivision_PacemakerImportCatalog"/>
        </sequence>
    </module>
</config>
```

We need to specify TechDivision_PacemakerImportCatalog module in the sequence of our new module.xml to be able to overwrite the existing pipeline configuration.

Order workflow

Add custom export format

Create an own module

First of all, we need to introduce a custom module.

Please refer to Create a New Module in **Magento's** developer documentation.

In this example, we name the module MyModule_CustomOrderExportFormat.

```
mkdir -p app/code/MyModule/CustomOrderExportFormat/etc
touch app/code/MyModule/CustomOrderExportFormat/etc/module.xml
touch app/code/MyModule/CustomOrderExportFormat/registration.php
```

app/code/MyModule/CustomOrderExportFormat/registration.php

```php
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_CustomOrderExportFormat',
    __DIR__
);
```

app/code/MyModule/CustomOrderExportFormat/etc/module.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="MyModule_CustomOrderExportFormat" setup_version="1.0.0">
        <sequence>
            <module name="TechDivision_PacemakerOrderExport"/>
        </sequence>
    </module>
</config>
```

We need to specify TechDivision_PacemakerOrderExport module in the sequence of our new module.xml to be able to overwrite the existing pipeline configuration.

Create your format executor

Now we need to create our own executor, which will load the order and generate the target format for our export.

Therefore we create a class as follows.

app/code/MyModule/CustomOrderExportFormat/Model/MyExecutor.php

```php
<?php
declare(strict_types=1);

namespace MyModule\CustomOrderExportFormat\Model;

use Magento\Framework\Exception\LocalizedException;
use Magento\Sales\Api\OrderRepositoryInterface;
use TechDivision\ProcessPipelines\Api\ExecutorLoggerInterface;
use TechDivision\ProcessPipelines\Api\StepInterface;
use TechDivision\ProcessPipelines\Model\Executor\AbstractExecutor;

class MyExecutor extends AbstractExecutor
{
    /**
     * @var OrderRepositoryInterface
     */
    private $orderRepository;

    /**
     * @param ExecutorLoggerInterface $logger
     * @param OrderRepositoryInterface $orderRepository
     */
    public function __construct(
        ExecutorLoggerInterface $logger,
        OrderRepositoryInterface $orderRepository
    ) {
        parent::__construct($logger);
        $this->orderRepository = $orderRepository;
    }

    /**
     * @param StepInterface $step
     * @return void
     * @throws LocalizedException
     */
    public function process(StepInterface $step)
    {
        // Load the order with the order_id given in step arguments
        $order = $this->orderRepository->get((int)$step->getArgumentValueByKey('order_id'));

        // Formatting logic here...
    }
```

```
}
```

BY DEFAULT, THIS VALUE IS SET TO	If you want to use the default transport adapter to move the result to a location within the system, you need to persist the body within the current working directory in the file order-export.body. Please refer to example module for more details.

Register the executor and dropdown select

Finally, it would be helpful if you would include your executor in the format list. That will automatically add your formatter to he dropdown in menu:Store[Configuration > Pacemaker > Order Workflow].

app/code/MyModule/CustomOrderExportFormat/etc/di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="TechDivision\PacemakerOrderExport\Model\ExportFormatProvider">
        <arguments>
            <argument name="formatList" xsi:type="array">
                <item name="my_custom_format" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_format</item>
                    <item name="label" xsi:type="string">My Custom Format</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportFormat\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module. = Add custom transport adapter

You can add an executor, which transports your transformed order to a target destination, in the same way like described for custom export formatter .

To register your executor you need to add following entry to the di.xml of your module:

app/code/MyModule/CustomOrderExportTransportAdapter/etc/di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="TechDivision\PacemakerOrderExport\Model\TransportAdapterProvider">
        <arguments>
            <argument name="transportAdapterList" xsi:type="array">
                <item name="my_custom_transport_adapter" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_transport_adapter</item>
```

```
                        <item name="label" xsi:type="string">My Custom Transport Adapter</item>
                        <item name="type"
xsi:type="string">MyModule\CustomOrderExportTransportAdapter\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module. = Add custom order response handler

You can add an executor, which handles the order export response, in the same way like described for custom export formatter .

To register your executor you need to add following entry to the di.xml of your module:

app/code/MyModule/CustomOrderExportResponseHandler/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="TechDivision\PacemakerOrderExport\Model\ResponseHandlerProvider">
        <arguments>
            <argument name="handlerList" xsi:type="array">
                <item name="my_custom_response_handler" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_response_handler</item>
                    <item name="label" xsi:type="string">My Custom Response Handler</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportResponseHandler\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module. = Add custom notification handler

You can add an executor, which sends a (or multiple) notification, in the same way like described for custom export formatter .

To register your executor you need to add following entry to the di.xml of your module:

app/code/MyModule/CustomOrderExportNotification/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
```

```xml
    <type name="TechDivision\PacemakerOrderExport\Model\NotificationHandlerProvider">
        <arguments>
            <argument name="notificationHandlerList" xsi:type="array">
                <item name="my_custom_notification" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_notification</item>
                    <item name="label" xsi:type="string">My Custom Notification</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportNotification\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module. = Order Update Feature

Pacemaker supports the order update in several ways.

You can install all order update modules with all their dependencies using the Pacemaker Order Update component, which calls the entire order update.

Pacemaker Order Update is part of Pacemaker Enterprise Edition.

```json
{
  "name": "pacemaker/component-order-update",
  "type": "metapackage",
  "require": {
    "php": "~7.2.0||~7.3.0||~7.4.0",
    "pacemaker/component-process-pipelines": "^1.5.4",
    "techdivision/order-update-notification-dispatcher": "^1.0.0",
    "techdivision/order-update-notification-dispatcher-api": "^1.0.5",
    "techdivision/order-update-file-system-observer": "^1.0.3",
    "techdivision/order-update-invoice": "^1.1.1",
    "techdivision/order-update-file-system-observer-invoice": "^1.0.1",
    "techdivision/order-update-shipment": "^1.1.1",
    "techdivision/order-update-file-system-observer-shipment": "^1.0.2",
    "techdivision/order-update-creditmemo": "^1.1.2",
    "techdivision/order-update-file-system-observer-creditmemo": "^1.0.2",
    "techdivision/order-update-cancelation": "^1.1.1",
    "techdivision/order-update-file-system-observer-cancellation": "^1.0.3"
  }
}
```

What can the Pacemaker Order Update component do?

Order Update File System Observer

The module takes care of all the logic of a job update file system, watching and reading the file located in the var/pacemaker/import directory.

```php
class FileReaderProvider
{
    /**
     * @var array
     */
    private $readerGroupedByPatternKey = [];
/**
     * configuration => [
     *       <key> => [
     *           [disabled] => bool,
     *           'pattern_key' => string
     *           'reader_instance' => instance of FileReaderInterface
     *       ]
     * ]
     *
     * @param array $configuration
     * @throws LocalizedException
     */
    public function __construct(
        array $configuration = []
    ) {
        foreach ($configuration as $key => $config) {
            if (isset($config['disabled']) && $config['disabled'] === true) {
                continue;
            }

            if (!isset($config['pattern_key'])) {
                throw new LocalizedException(__('"pattern_key" needs to be defined'));
            }

            if (!($config['reader_instance'] instanceof FileReaderInterface)) {
                throw new LocalizedException(
                    __(
                        '%class needs to implement %interface',
                        [
                            'class' => get_class($config['reader_instance']),
                            'interface' => FileReaderInterface::class
                        ]
                    )
                );
            }

            if (!isset($this->readerGroupedByPatternKey[$config['pattern_key']])) {
                $this->readerGroupedByPatternKey[$config['pattern_key']] = [];
            }
```

```
            $this->readerGroupedByPatternKey[$config['pattern_key']][$key] =
$config['reader_instance'];
        }
    }

    /**
     * @param string $patternKey
     * @return FileReaderInterface[]
     */
    public function execute(string $patternKey): array
    {
        if (!isset($this->readerGroupedByPatternKey[$patternKey])) {
            return [];
        }

        return $this->readerGroupedByPatternKey[$patternKey];
    }
```

and creates pipelines for them

```
class PipelineInitializer implements InitializationDataFetcherInterface
{
....
    /**
     * @var ApplyFileReaders
     */
    private $applyFileReaders;
 /**
     * configuration => [
     *      <key> => [
     *          [disabled] => bool,
     *          'pattern_key' => string
     *          'reader_instance' => instance of FileReaderInterface
     *      ]
     * ]
     *
     * @param array $configuration
     * @throws LocalizedException
     */
    public function __construct(
        array $configuration = []
    ) {
        foreach ($configuration as $key => $config) {
            if (isset($config['disabled']) && $config['disabled'] === true) {
                continue;
            }

            if (!isset($config['pattern_key'])) {
```

```
                throw new LocalizedException(__('"pattern_key" needs to be defined'));
            }

            if (!($config['reader_instance'] instanceof FileReaderInterface)) {
                throw new LocalizedException(
                    __(
                        '%class needs to implement %interface',
                        [
                            'class' => get_class($config['reader_instance']),
                            'interface' => FileReaderInterface::class
                        ]
                    )
                );
            }

            if (!isset($this->readerGroupedByPatternKey[$config['pattern_key']])) {
                $this->readerGroupedByPatternKey[$config['pattern_key']] = [];
            }

            $this->readerGroupedByPatternKey[$config['pattern_key']][$key] =
$config['reader_instance'];
        }
    }

    /**
     * @param string $patternKey
     * @return FileReaderInterface[]
     */
    public function execute(string $patternKey): array
    {
        if (!isset($this->readerGroupedByPatternKey[$patternKey])) {
            return [];
        }

        return $this->readerGroupedByPatternKey[$patternKey];
    }
....
```

Order Update File System Observer CreditMemo

- The TechDivision_OrderUpdateFileSystemObserverCreditmemo module does the job of observe a creditmemo.
- The module creates the pipelines for the **creditmemo** file.
- The filename of the resulting **JSON** file must start with creditmemo, and then any character creditmemo*.json under the directory var/pacemaker/import using order-update-file-system-observer-creditmemo

Order Update File System Observer Invoice

- The TechDivision_OrderUpdateFileSystemObserverInvoice module does the job of observe a creditmemo.

- The module creates the pipelines for the **invoice** file.
- The filename of the resulting **JSON** file must start with invoice, and then any character invoice*.json under the directory var/pacemaker/import using order-update-file-system-observer-invoice

Order Update File System Observer Shipment

- The TechDivision_OrderUpdateFileSystemObserverShipment module does the job of observe a creditmemo.
- The module creates the pipelines for the **shipment** file.
- The filename of the resulting **JSON** file must start with shipment, and then any character shipment*.json under the directory var/pacemaker/import using order-update-file-system-observer-shipment

Order Update File System Observer Cancelation

- The TechDivision_OrderUpdateFileSystemObserverCancelation module does the job of observe a creditmemo.
- The module creates the pipelines for the **cancelation** file.
- The filename of the resulting **JSON** file must start with cancelation, and then any character cancelation*.json under the directory var/pacemaker/import using order-update-file-system-observer-cancelation

Order Update CreditMemo

Handles notification and Reads the steps and executes them to create a creditmemo order-update-creditmemo

Order Update Shipment

It processes the notifications, reads the steps, and executes them to create a shipment. order-update-shipment

Order Update cancelation

It processes the notifications, reads the steps, and executes them to create a cancelation order-update-cancelation

Order Update Invoice

It processes the notifications, reads the steps, and executes them to create a Invoice order-update-invoice

In this example, we can see the **PROCESS** *method that executes the step*

```
class InvoiceCreator extends AbstractExecutor
{
......
    /**
     * @param StepInterface $step
     * @return int|void
     * @throws ExecutorException
     */
    public function process(StepInterface $step)
    {
        $notificationId = (string)$step->getArgumentValueByKey('notification_identifier');
        $notification = $this->loadNotification->execute($notificationId);
        $creationDocument = $notification->getDocument();
```

```
    if (!($creationDocument instanceof InvoiceCreationInterface)) {
        throw new ExecutorException(
            __('Notification document is not instanceof %1',
InvoiceCreationInterface::class)
        );
    }

    $externalId = (string)$step->getArgumentValueByKey('external_order_id');
    $order = $this->resolveExternalId->execute($externalId);
    if (empty($order)) {
        throw new ExecutorException(
            __('Can not resolve order for external id: %1', $externalId)
        );
    }

    $invoice = $this->invoiceCreator->execute($order, $creationDocument);
    $invoice->setIncrementId($creationDocument->getTargetIncrementId());
    $this->invoiceRepository->save($invoice);

    $this->logger->info(
        sprintf(
            'Invoice Increment Id was updated to %s',
            $invoice->getIncrementId()
        )
    );
    }
}
.....
```

Best Practices & Examples

Integration Techdivision Imagecache

Custom integration pipeline GUI

FAQ

- Composer runs into auth issues on my Mac OS X machine.
- The performance on the production/stating system is worse than on my local machine.
- Timestamp Detection does not work.
- General
- Supervisor does not restart the runners any more.
- Attribute code does not exist, verify the attribute and try again.
- What are the general rules?

- Which type of product do I have to specify for articles with characteristics (e.g., color)?
- How do I connect profiling articles?
- How must the file names be structured?
- How must the category assignment take place?
- Which value must be specified in the StoreView?
- How can products be assigned to individual websites?
- Values sometimes do not work, why?
- Why are images not displayed even though the path in the file is correct to the actual path?
- Product.csv ‭ visibility is set wrong
- URLs are not generated correctly.
- Special characters in categories Category.csv and Product.csv.
- I get the following SQL error during import after adding a field to one of the Magento standard tables.

Composer runs into auth issues on my Mac OS X machine.

Question

As Solution Partner or Customer, you received a ext12345 username together with a token. It gives you access via **Composer** to the necessary libraries.

These credentials have to be added in your auth.json, which can, for example, be in the source directory of your project like src/auth.json.

Example:

```
{
    "http-basic": {
        "gitlab.met.tdintern.de": {
            "username": "ext00000",
            "password": "asaZIjkUIollKSADnrlm"
        }
    }
}
```

Whenever **Composer** is invoked, these credentials will be used for the **HTTP** download, **Composer** will do.

In some cases, you will receive a message from **Composer** that you will not have the necessary access rights to install **Pacemaker** or one of its packages.

Anwser

MacOS saves the credentials in the keychain on the host level (https://gitlab.met.tdintern.de).

When invoking **Composer** the first time, and it doesn't matter from which project, the first host entry from the keychain will be used and not the auth.json anymore.

It may lead to the problem that the Pacemaker libraries can not be loaded anymore because of missing access.

In the case of **Pacemaker**, it will be necessary to disable the caching of the system credentials for **HTTPS** calls.

Therefore execute the following commands to remove the credential helper from the **GIT** configuration

```
git config --local --unset-all credential.helper \
   && git config --global --unset-all credential.helper \
   && git config --system --unset-all credential.helper
```

After that, the credential helper has to re-initialized empty (because of any other tool like **xCode** for example may also use it) with the following command

```
git config --global --add credential.helper "" && composer clear-cache
```

Finally, search in the keychain tool for met and delete the entry gitlab.met.tdintern.de.

If the keychain has been deactivated, in the future **GIT** should **always** use the credentials from the auth.json from your project or the global one.

The performance on the production/stating system is worse than on my local machine.

Question

The performance between your local and any other system differs significantly. What can be the reason?

Anwser

Probably the **MySQL** transaction log has different settings.

The option innodb_flush_log_at_trx_commit by default has the value 1.

It means, that the transaction log will be written by MySQL after each commit.

This option controls the balance between strict **ACID** compliance for commit operations and higher performance that is possible when commit-related **I/O** operations are rearranged and done in batches.

Setting this value to 2, you can achieve better performance, but then you could lose transactions in a crash.

Possible values are

0	write and flush once per second
1	write and flush at each commit
2	write at commit, flush once per second

For example, switching this value from 1 to 2 the import performance improves from 02:06:10 to 00:03:29 h which is for sure significant

ID	Pipeline	Created	Finished	Duration
219	xxx_import_catalog	Oct 24, 2019 2:47:04 PM	Oct 24, 2019 4:53:14 PM	02:06:10 h
221	xxx_import_catalog	Oct 24, 2019 5:02:02 PM	Oct 24, 2019 5:05:31 PM	00:03:29 h

Timestamp Detection does not work.

Question

I'm using the Pacemaker Professional Edition (PE) > 3.8.0, when I try to activate the timestamp detection with --use-timestamp=true, everything seems to work fine but the performance is at the same level as when i activate the change-set detection with --use-change-set=true.

The log file does not contain any error messages, as well as the console.

Anwser

It may result out of the problem that the date format in your **CSV** file is different from the default format and the necessary date format has not been configured in the configuration.

Pretending you are using **Magento Commerce**, to change the date format to Y-m-d H:i:s, create a snippet named <magent-install-dir>/app/etc/configuration/operations.json that contains the following content

```json
{
  "operations": {
    "ee": {
      "catalog_product": {
        "validate": {
          "plugins": {
            "subject": {
              "id": "import.plugin.subject",
              "listeners": [
                {
                  "plugin.process.start": [
                    "import.listener.reset.loader.eav.attribute.option.value"
                  ],
                  "plugin.process.success": [
                    "import.listener.stop.validation"
                  ]
                }
              ],
              "subjects": [
                {
                  "id": "import.subject.validator",
                  "create-imported-file": false,
                  "date-converter": {
                    "source-date-format": "Y-m-d H:i:s"
                  },
                  "file-resolver": {
                    "prefix": "product-import"
                  },
                  "listeners": [
                    {
                      "subject.artefact.header.row.process.start": [
                        "import.listener.validate.header.row"
                      ]
```

```
                            }
                        ],
                        "params": {
                          "custom-validations": {
                            "sku": [
                              "/.+/"
                            ],
                            "product_type": [
                              "simple",
                              "virtual",
                              "configurable",
                              "bundle",
                              "grouped",
                              "giftcard",
                              "designyourown"
                            ],
                            "visibility": [
                              "Not Visible Individually",
                              "Catalog",
                              "Search",
                              "Catalog, Search"
                            ]
                          }
                        },
                        "observers": [
                          {
                            "import": [
                              "import_product.observer.composite.base.validate"
                            ]
                          }
                        ],
                        "callbacks": [
                          {
                            "sku": [
                              "import.callback.custom.regex.validator"
                            ],
                            "store_view_code": [
                              "import.callback.store.view.code.validator"
                            ],
                            "attribute_set_code": [
                              "import.callback.attribute.set.name.validator"
                            ],
                            "product_type": [
                              "import.callback.custom.array.validator"
                            ],
                            "tax_class_id": [
                              "import_product.callback.validator.tax.class"
                            ],
                            "product_websites": [
```

```
          "import.callback.store.website.validator"
        ],
        "visibility": [
          "import.callback.visibility.validator"
        ],
        "related_skus": [
          "import_product.callback.validator.link"
        ],
        "upsell_skus": [
          "import_product.callback.validator.link"
        ],
        "crosssell_skus": [
          "import_product.callback.validator.link"
        ],
        "created_at": [
          "import.callback.validator.datetime"
        ],
        "updated_at": [
          "import.callback.validator.datetime"
        ],
        "special_price_to_date": [
          "import.callback.validator.datetime"
        ],
        "special_price_from_date": [
          "import.callback.validator.datetime"
        ],
        "custom_design_to": [
          "import.callback.validator.datetime"
        ],
        "custom_design_from": [
          "import.callback.validator.datetime"
        ],
        "new_to_date": [
          "import.callback.validator.datetime"
        ],
        "new_from_date": [
          "import.callback.validator.datetime"
        ],
        "price": [
          "import.callback.validator.number"
        ],
        "special_price": [
          "import.callback.validator.number"
        ],
        "map_price": [
          "import.callback.validator.number"
        ],
        "msrp_price": [
          "import.callback.validator.number"
```

```json
          ],
          "qty": [
            "import.callback.validator.number"
          ],
          "is_returnable": [
            "import_product_ee.callback.rma.validator"
          ]
        }
      ]
    }
  ]
}
    },
    "replace": {
      "plugins": {
        "subject": {
          "id": "import.plugin.subject",
          "subjects": [
            {
              "id": "import_product_ee.subject.bunch",
              "date-converter": {
                "source-date-format": "Y-m-d H:i:s"
              },
              "file-resolver": {
                "prefix": "product-import"
              },
              "params": {
                "copy-images": false
              },
              "observers": [
                {
                  "import": [
                    "import_product_ee.observer.composite.base.replace"
                  ]
                }
              ]
            }
          ]
        }
      }
    },
    "add-update": {
      "plugins": {
        "subject": {
          "id": "import.plugin.subject",
          "subjects": [
            {
              "id": "import_product_ee.subject.bunch",
```

```
              "date-converter": {
                "source-date-format": "Y-m-d H:i:s"
              },
              "file-resolver": {
                "prefix": "product-import"
              },
              "params": {
                "copy-images": false,
                "clean-up-media-gallery": true,
                "clean-up-empty-image-columns": true,
                "clean-up-website-product-relations": true,
                "clean-up-category-product-relations": true,
                "clean-up-empty-columns": [
                  "special_price",
                  "special_price_from_date",
                  "special_price_to_date"
                ]
              },
              "observers": [
                {
                  "import": [
                    "import_product_ee.observer.composite.base.add_update"
                  ]
                }
              ]
            }
          ]
        }
      }
    }
   }
  }
 }
}
```

General

Question

General

Anwser

Field contents should be enclosed in double-quotes (").

Supervisor does not restart the runners any more.

Question

After working for a while without issues, the Supervisor stops restarting the runner consumers. The Supervisor log displays an error like FATAL Exited too quickly (process log may have details).

Anwser

Even if a process exits with an expected exit code, Supervisor will consider the start as a failure if the process goes quicker than startsecs. Which is default 1 second.

You must set the option startsecs=0 to the Supervisor configuration to avoid this behavior.

Please refer to How to configure the runners .

What are the general rules?

Question

What are the general rules?

Anwser

Configurable and Simple

- Use columns configurable_variations and configurable_variation_labels.
- The assignment takes place on the level of the 'Configurable'. The **SKU's** of simples and the reference to the attribute's appropriate attribute code are specified in connection with the attribute's value.
- Is configurable via the backend.

General

- File names can be split into several blocks as discussed.
- If only one file exists, however, it must also end with 01.csv.
- Please use a comma instead of a semicolon as separator.

Which type of product do I have to specify for articles with characteristics (e.g., color)?

Question

Which type of product do I have to specify for articles with characteristics (e.g., color)?

Anwser

Configurable and Simple

How do I connect profiling articles?

Question

How do I connect profiling articles?

Anwser

- via the columns configurable_variations and configurable_variation_labels.
- the assignment takes place on the configurable level.

Here the Simple SKU's and the corresponding attribute code, including the attribute values, are specified

How must the file names be structured?

Question

How must the file names be structured?

Anwser

Filenames are configurable at the backend.

General

- Filenames can be split into several bunches.
- If there is only one file, it must also end with '_01.csv'.
- As a separator please use a comma instead of a semicolon.

How must the category assignment take place?

Question

How must the category assignment take place?

Anwser

- the categories must start with Default Category; this is the root category.
- the fields is_anchor and include_in_menu must be 1 to be clickable and to be displayed in the menu structure.
- the field store_view_code must be filled with the values de,fr,uk,nl,it,at,ch,es,se,dk (already corresponds to the conversion as implemented in the Base Setup).

Which value must be specified in the StoreView?

Question

Which value must be specified in the StoreView?

Anwser

The value must be supplied with the StoreView code from **Magento**.

How can products be assigned to individual websites?

Question

How can products be assigned to individual websites?

Anwser

In the file Product.csv in the field product_websites, comma-separated with the website codes from **Magento**.

Values sometimes do not work, why?

Question

Values sometimes do not work, why?

Anwser

- The values must be specified within the field, separated by commas.
- There must not be a space e.g. abc, def wrong | abc,def correct.

Why are images not displayed even though the path in the file is correct to the actual path?

Question

Why are images not displayed even though the path in the file is correct to the actual path?

Anwser

Magento can only process two folder levels at the same time.

Product.csv visibility is set wrong

Question

Product.csv visibility is set wrong

Anwser

- IS: Catalog,Search
- SHOULD BE: Catalog, Search

URLs are not generated correctly.

TURNING ONLINE PROJECTS INTO SUCCESS
ANALYSE, KONZEPTION, DESIGN, WEBENTWICKLUNG, ONLINE-MARKETING

Question

URLs are not generated correctly.

Anwser

The **Url's** for the products can't be generated correctly at the moment and cause an error.

The reason for this is that the value inside the url_key field may be empty. It causes **Magento** to try to generate the **URL** automatically, but only the name of the product is used.

Since this value is identical in the file for the Configurable and Simples, this constellation creates **Duplicate Key Errors** when generating the **URL Rewrites**.

Special characters in categories Category.csv and Product.csv.

Question

Special characters in categories Category.csv and Product.csv.

Anwser

e.g. Default Category/Loungemöbel/Barhocker / Tresenhocker ⬚ must be passed as Default Category/Loungemöbel/"Barhocker / Tresenhocker".

I get the following SQL error during import after adding a field to one of the Magento standard tables.

Question

I get the following SQL error during import after adding a field to one of the Magento standard tables.

```
SQLSTATE[HY093]: Invalid parameter number: number of bound variables does not match the
number of tokens when executing SQL "INSERT INTO catalog_product_entity_media_gallery
(attribute_id,value,media_type,disabled,modified_at)
VALUES (90,/w/p/wp12-blue_main.jpg,image,0,:modified_at)"
in file `media_20161021-161909_04.csv` on line 407
```

Anwser

The **Pacemaker** knows how to handle **Magento 2** default functionality.

If you are using modules that add additional columns to tables, or add them yourself, which will be utilized by **Pacemaker**, these columns should have a default value set.

Importing data into tables with custom columns without default value will fail as **Pacemaker** does not know what kind of data is being expected.

If a module must add columns without a default value, you need to manually add values for these columns to the import **CSV** file.