EasternGraphics
visualize your business

# OFML Data Structure and Registration (DSR)

Specification version 3.6

Editors: Bernd Heinemann, Stefan Bleuel, Thomas Gerth

August 1, 2023

## Legal disclaimer

# Contents

# References

| | |
|---|---|
| [asv] | Article Specific View setup (ASV) Specification. EasternGraphics GmbH |
| [gln] | GLN `http://en.wikipedia.org/wiki/Global_Location_Number` |
| [glos] | Libraries, Series & Co. – fundamental OFML terms. EasternGraphics GmbH |
| [gtin] | GTIN `http://en.wikipedia.org/wiki/Global_Trade_Item_Number` |
| [iso639-1] | ISO 639-1 `http://en.wikipedia.org/wiki/ISO_639` |
| [iso3166] | ISO 3166 `http://en.wikipedia.org/wiki/ISO_3166` |
| [iso8601] | ISO 3166 `http://en.wikipedia.org/wiki/ISO_8601` |
| [oap] | OFML Aided Planning (OAP) Spezifikation. EasternGraphics GmbH |
| [ofml] | OFML – Standardized data description format of the office furniture industry. Industrieverband Büro und Arbeitswelt e. V. (IBA) |
| [ppr] | Preisprofile (PPR) in Herstellerdaten zur Verarbeitung mit pCon-Produkten. EasternGraphics GmbH |
| [utc] | UTC `http://en.wikipedia.org/wiki/Coordinated_Universal_Time` |
| [xcf] | Extensible Catalog Format (XCF) Specification. EasternGraphics GmbH |

# 1 Preliminary notes

Regarding file names pay attention to case sensitivity corresponding to their formation rule. The file name has to be written exactly as the stored keys in the registration file. This convention is urgent to guarantee the operativeness of the OFML data (see [ofml]) platform independent. In principle the use of small letters is recommended.

For all path definitions the slash character ('/') will be used throughout this specification. Alternatively the backslash '\' may be used.

Key and key value defined in this specification have to be used exactly in the spelling defined here.

The following styles will be used for the identification of different elements:

`Example`
`File respectively path name`
***Key defination***
*`Key reference`*

The identifier behind the key name specifies:

| | |
|---|---|
| **[M]** | mandatory key |
| **[M/O]** | conditional key |
| **[O]** | optional key |

In many places of this specification the values of specific keys are used to define file or path names. In that case the specified key is marked with prefix '$' and the reference is put in parentheses. The data root directory which is used in path definitions has the symbolic name `<data>`.

# 2 Directory structure

## 2.1 Structure of OFML product and catalog data

The illustration below shows the general idea of the commonly used directory structure in pCon applications. In the following sections each directory and its content will be explained in detail.

```
<data>/
    catalogs/
        images/ ................................................................... [3.1.3]
        profiles/ ................................................................. [3.1.2]
    registry/ ...................................................................... [5.2]
    ($manufacturer)/
        ($program)/
            ($version)/.............................................................[6.1]
                ($region)/
                ($version)/
                    cat/............................................................[6.3]
                    db/.............................................................[6.2]
                    etc/ .........................................................[6.4.4]
                    html/ ........................................................[6.4.3]
                        de/
                        en/
                        fr/
                        ...
                    image/........................................................[6.4.1]
                    mat/ .........................................................[6.4.2]
                        l/
                        m/
                        s/
                    meta/
                    oam/
                    oap/
        priceprofiles/..............................................................[6.6]
            ($ppr_version)/
```

Some elements in the structure can be stored as a **ZIP archive**:

```
<data>/
    ($manufacturer)/
        ($program)/
            ($region)/
                ($version)/
                    cat/xcf.zip
                    image/image.zip
                    mat/mat.zip
```

Notes for the use of ZIP archives:
The ZIP archive contains the files exactly as they would be in the directory (i.e., no additional sub folders). An exception is the `mat.zip`, see 6.4.2. If a file exists inside the archive and in the directory too, the file from the directory will be used.

## 2.2 Example structure

```
<data>/
    catalogs/
        images/
            man_de-2011.0.jpg
            man_de-2012.0.jpg
            man_fr-2011.0.jpg
            man_fr-2012.1.jpg
            ...
        profiles/
            man.cfg
            man_de-2011.0.cpr
            man_de-2012.0.cpr
            man_fr-2011.0.cpr
            man_fr-2012.1.cpr
            ...
    man/
        series/
            1/odb.ebase
            2/odb.ebase
            3/odb.ebase
            DE/
                1/
                    cat/xcf.zip
                    db/pdata.ebase
                    etc/artsetup.csv
                    image/image.zip
                    mat/mat.zip
                    oam/oam.ebase
                2/
                    cat/xcf.zip
                    db/pdata.ebase
                    etc/artsetup.csv
                    image/image.zip
                    mat/mat.zip
                    oam/oam.ebase
            FR/
                1/
                    cat/xcf.zip
                    db/pdata.ebase
                    etc/artsetup.csv
                    image/image.zip
                    mat/mat.zip
                    oam/oam.ebase
                3/
                    cat/xcf.zip
                    db/pdata.ebase
                    etc/artsetup.csv
                    image/image.zip
                    mat/mat.zip
                    oam/oam.ebase
        ...
    registry/
        man/
            man.jpg
        MAN.cfg
        man_series_DE_1.cfg
        man_series_DE_2.cfg
        man_series_FR_1.cfg
        man_series_FR_3.cfg
        ...
```

# 3 Data and Catalog profiles

This chapter describes the registration of OFML data in a pCon application.

## 3.1 Directory structure of data registration

All applications have to support a directory structure that looks like this:

| Directory (relative) | Short description |
|---|---|
| `/bin` | executable program files, system libraries |
| `/lib` | module directory (optional) |
| `/data` | OFML product and catalog files |
| `/data/catalogs` | catalog profiles and associated resources |
| `/etc/data` | data profile directory |
| `/etc/startup` | configuration files |

Usually these directories are located underneath the program directory.
Particular directories can be outside of the program directory, e.g. in order to realize server solutions.

OFML data is registered via data profiles or catalog profiles:

- A data profile describes a single OFML dataset of a manufacturer without an identification for distribution region and version.
  Date profiles are used if in the application can or should be used only a single OFML dataset of the manufacturer.
- A catalog profile describes an OFML dataset (catalog) of a manufacturer resp. supplier with identification for distribution region and version.
  Catalog profiles allow for simultaneous usage of various OFML datasets of a manufacturer/supplier, differing in distribution region and/or version[1]. However, not (yet) all application support this. Therefore, for other applications a data profile[2] should be provided anyway.

The data profiles and catalog profiles to be processed by the application are specified in the start file `default.cfg` located in the directory `/etc/startup`.

Amongst others, this file contains the key *`app.gf.data.profile`* and/or the key *`app.gf.data.catalogs`*. They contain the names of the profiles which have to be loaded.
Alternatively these keys can be read from a separate file. This file can be specified by the key *`app.gf.data.profile.registration`*[3].

### 3.1.1 Registration of data profiles via `app.gf.data.profile`

This key is used by applications *not* yet supporting the processing of catalog profiles.

Data profiles will be stored as files with extension `.cfg` in the profile directory.
The profile directory can be specified by the key *`app.gf.data.profile.path`*. This key is optional. If it is not defined the locale profile directory (`/etc/data`) will be used. Data profiles are privileged loaded from the directory *`app.gf.data.profile.path`*. If no profile directory exists there the application attempts to load it from the locale directory.

[1] This concept sometimes is called *Multiple price lists*.
[2] the so-called compatibility data profile
[3] Key *`app.gf.data.profile.registration`* takes precedence over the other two keys.

In the key *app.gf.data.profile* the names of the profile files are given without the extension `.cfg`. Several profiles are separated by semicolon.

**Example 1**

```
default.cfg:

app.gf.data.profile.path = \\SERVER\data\profiles
app.gf.data.profile = standard_DE_1;man1
```

**Example 2**

```
default.cfg:

app.gf.data.profile.path = \\SERVER\data\profiles
app.gf.data.profile = standard_DE_1;man1;man2
```

**Example 3**

```
default.cfg:

app.gf.data.profile.path = \\SERVER\data\profiles
app.gf.data.profile.registration = \\SERVER\data\profiles\app.profiles

\\SERVER\data\profiles\app.profiles:

app.gf.data.profile = standard_DE_1;man1;man2;man3
```

### 3.1.2 Registration of catalog profiles via `app.gf.data.catalogs`

This key is used by applications supporting the processing of catalog profiles.

> Note:
> If the key *app.gf.data.catalogs* does not exist in the profile registration file, an application reads the profiles from the old key *app.gf.data.profile*. This ensures backward compatibility of an application that supports catalog profiles with respect to traditional data installations that have not been migrated yet. However, the old key *app.gf.data.profile* is not evaluated if key *app.gf.data.catalogs* is present in the profile registration file.

The name of a catalog profile file has the following structure:

```
($brand)_($catalog_id).cpr
```

The key *app.gf.data.catalogs* contains the file names of the profiles to be loaded *including* file extension but without a path. Multiple profiles are each separated by a semicolon.

I addition to catalog profiles in the key conventional data profiles can be specified, too. This way manufacturers are supported, which do not provide catalog profiles yet.

Catalog profiles are stored by default in the profile directory `catalogs/profiles` under the OFML data directory. An alternate profile directory can be specified via optional key `app.gf.data.profile.path` in the boot file `etc/startup/default.cfg` of an OFML application. If a catalog profile does not exist in the profile directory, the application looks for the profile in its local path `etc/data`.

**Example**

```
app.gf.data.catalogs=man1_de-2012.1.cpr;man2.cfg;man3_any-2012.4.cpr
```

### 3.1.3 Storage of catalog profile resources

An optional logo to represent a catalog in a catalog selection of the application can be stored in 2 different sizes as:

Catalog logo small:

```
catalogs/images/($brand)_($catalog_id)_.jpg
[catalogs/images/($brand)_($catalog_id)_.png]
```

Image size: [1 - max. 100] x 20 (W x H in pixels)

Catalog logo large:

```
catalogs/images/($brand)_($catalog_id).jpg
[catalogs/images/($brand)_($catalog_id).png]
```

Image size: [1 - max. 200] x 40 (W x H in pixels)

The format of the image file is JPEG[4]. Optionally also PNG can be used, however this is not supported by all applications. If the logo is given in JPEG format as well as in PNG format, PNG takes precedence.

If no logo is stored for the catalog, the application uses the logo of the brand or of the manufacturer to represent the catalog (see section 4.2.3).

## 3.2 Data profiles

Data profiles are text files based on a fixed scheme. The following table gives a review about the individual sections:

| Section | Short description |
|---|---|
| `[config]` | Settings and paths |
| `[<lang>]` | language specific descriptions |
| `[lib:`*$group*`]` | packages of a group *$group* |

Annotation:
The group identifier *$group* should use the manufacturer identifier specified in the registration. It is allowed to enlarge the name by a successive number or use free identifier names, to pool specific packages in separate sections.

### 3.2.1 Settings and paths

[1] ***path* [M/O]**
– defines the package search path

The entry `path` specifies the path to the packages registered with the profile. This entry has to be specified if the packages are not located in the locale data directory (`<program>/data`) of the software system. Only one path is allowed within a profile.
The entry has to provide a complete path information in the form `<drive>/<path>` or `//<server>/<path>` without a closing slash ('/').

[4]For image format conventions see section 6.5.

[2] ***use_version* [O]**
– permits the use of not released data

Normally the value is *true*. With *false* the versioning will be deactivated. For that compatible libraries without versioning are assumed.

### 3.2.2 Descriptions

The name of the section*<lang>* has to be specified according to ISO language code (ISO–639-1).

[3] ***name* [O]**
– defines a short term for the profile

[4] ***desc* [O]**
– defines a detailed description for the profile

### 3.2.3 Packages of a group

In the section *[lib:$group]* all manufacturer specific packages are specified which have to be used by the software system. Each entry consists of the name of the package registration file without file extension `.cfg` (see section 5.2) followed by the equality sign and the loading mode. This mode is defined either as *true* or as *false*, where *true* instructs the software system to use the package.

For a given OFML library only one package[5] may be specified.

### 3.2.4 Example for a data profile: standard_DE_1.cfg

```
[config]
path =

[en]
name = Standard
desc = profile for EasternGraphics basic libraries

[lib:egr]
egr_office_np_DE_1=true
egr_accessories_DE_1=true
```

## 3.3 Catalog profiles

A catalog is a collection of products of a given manufacturer resp. supplier and comprises – from an OFML point of view – all OFML packages needed for the selection, graphical presentation, configuration and order processing of the products of the catalog[6]. A catalog has always both a spatial dimension (sales region) and a temporal dimension (version).

A catalog profile defines a release of a catalog of a manufacturer/supplier.

[5]for a defined distribution region and a defined version
[6]Sometimes as a synonym the term *price list* is used.

### 3.3.1 Structure

The catalog profile is stored as a configuration file. The profile settings are defined as key–value pairs

```
<key> = <value>
```

each in a single line and divided into several sections.

Lines starting with character # are interpreted as a comment.

Below the sections and keys of a catalog profile file are described in detail.

| section | explanation |
|---|---|
| `[catalog]` | settings and paths |
| `[<lang>]` | language specific descriptions |
| `[lib]` | OFML packages used by the catalog |

A catalog profile uses UTF–8 as the code page. The first 3 bytes may contain the optional byte order mark for UTF–8 (`EF BB BF`). Alternatively, it is permitted to save a catalog profile in UTF–16 Little Endian. The usage of UTF–16 is declared by the byte order mark `FF FE`.

### 3.3.2 Settings

[5] ***catalog_id*** **[M]**
– specifies the unique identification key of the catalog.

The catalog ID uniquely identifies a catalog and the products inserted from this catalog into an OFML project. Therefore, for each new release of a catalog a new catalog ID must be assigned. It is not permitted for two different versions of a catalog to use the same catalog ID.

The catalog ID has the following structure: `<identifier>.<revision>`

The identifier identifies the catalog and must match the regular expression `[a-z][a-z0-9_-]*`. An identifier of the form `<sales region>-<year of release>` is recommended.
Catalog revisions are used to represent changes (corrections) within a catalog. The revision number is a natural number greater than or equal to 0. For the initial issue of a catalog revision number 0 has to be used. At each delivered change of the catalog it must be increased accordingly.

Catalogs which are supposed to be processed simultaneously in an application have to possess different identifiers in their catalog ID. A revision of a catalog replaces a previous revision of the same catalog with a lower revision number.

```
Example: catalog_id = de-2011.1
```

[6] ***release_date*** **[M]**
– declares the release date of the catalog.

The value of this key has to be specified as an ISO date in the form `YYYY-MM-DD` [iso8601].

```
Example: release_date = 2011-03-17
```

[7] ***brand*** **[M]**
– defines the OFML identifier of the brand under which the catalog is listed.

The value of the key can not be freely assigned. It has to be registered in the central registry database of manufacturers (s. 4.1).
If the catalog is not listed under a specific brand, here the short name of the manufacturer/supplier has to be specified (key *manufacturer*, see 5.3).

[8] ***brand_id* [M]**
– defines the commercial ID of the brand under which the catalog is listed.

The value of the key can not be freely assigned. It has to be registered in the central registry database of manufacturers (s. 4.1).
Using this key, additional information is retrieved from the central registry database of manufacturers (e.g. the display name or the manufacturer's address).
If the catalog is not listed under a specific brand, here the *manufacturer_id* of the manufacturer/supplier has to be specified.

[9] ***distribution_region* [M]**
– defines the basic sales region for which the catalog is valid.

Using the sales region, an application can categorize different catalogs in order to submit replacement proposals to the user.

[10] ***valid_from* [O]**
– defines the date from which on the catalog may be used.

The value has to be formatted as an ISO date `YYYY-MM-DD` [iso8601].

The validity period specified via *valid_from* and *valid_to* has only informative character. Applications do not limit the use of the catalog to that period. However, this information can be used to display warnings or recommendations to uninstall the catalog if the period has exceeded.

If the lower limit of the validity period is not specified, the date `1970-01-01`, sufficiently far in the past, will be assumed.

[11] ***valid_to* [O]**
– defines the date up to which the catalog may be used.

The value has to be formatted as an ISO date `YYYY-MM-DD` [iso8601].

If the upper limit of the validity period is not specified, the date `9999-12-31`, sufficiently far in the future, will be assumed.

[12] ***path* [O]**
– defines the path to the used OFML data.

If the OFML data used by the catalog is not located in the local data directory of the application, the base path to the data has to be specified in this key.
The path has to be stated in the form `<drive>/<path>` or `//<server>/<path>` without final slash ('/').

[13] ***ppr_region_id* [O]**
– assigns the catalog to a price profile region.

If a price profile region is activated, together with the catalog a price profile package has to be distributed, in which this region is defined (see also 6.6).

[14] ***ppr_version*** **[M/O]**
– defines the version of the used price profile.

The version of the used price profile has to be specifed if a price profile region is activated via key *ppr_region_id* (see also 6.6).

```
Example: ppr_version = 1
```

### 3.3.3 Descriptions

The name of a catalog as well as optional descriptions for a default language are stated in section `[catalog]`. Furthermore, these texts can be stored for additional languages in additional optional sections `[<lang>]`. The key `[<lang>]` defines the language and must be specified as a two-letter ISO language code (ISO–639-1) [iso639-1].

[15] ***catalog_name*** **[M]**
– specifies the name of the catalog.

The name should allow for an easy identification of the catalog by the user, e.g.:
`catalog <sales region> <year>/<issue>`.

> Note:
> Catalogs are usually hierarchically listed below the manufacturer (supplier). Applications that use a manufacturer–independent list of available catalogs, can concatenate the manufacturer name with the catalog name. Therefore, the catalog name itself should not include the name of the manufacturer.

[16] ***description*** **[O]**
– specifies a detailed description of the catalog.

Fixed line breaks can be defined in the description using `\n`.

### 3.3.4 Data packages

The packages used by a catalog are specified in the section `[lib]`.

For each package a key of form

```
<package key> = [true|false]
```

has to be specified, where *<package key>* corresponds to the name of the package registration file without file extension (see section 5.2) and value `true` specifies, that the package is used in the context of the catalog.

For a given OFML library only one package[7] may be specified.

If a package registration file cannot be loaded, the entire catalog can not be loaded/used.

A catalog profile may reference packages of several manufacturers, but the packages with catalog data have to be from the same manufacturer resp. from manufacturers of same concern (*brand*) or from manufacturers `::egr::` and `::ofml::`.

[7] for a defined distribution region and a defined version

# 4 Registration of manufacturers and concerns

## 4.1 Central registration database

Manufacturers[8] and concerns are registered centrally by EasternGraphics.
(Please contact cto@EasternGraphics.com.)
This avoids conflicts assigning unique keys more than once. The registration data is maintained in a global manufacturer database (`Manufacturers.ebase`).

This database contains the assignment of code designations (IDs) of manufacturers and concerns, as well as their names.

A list with allocated IDs can be requested from EasternGraphics at any time.

The following keys might partially not be used if an application derives the information directly from the global manufacturer database.

## 4.2 Manufacturer registration

For each manufacturer an own registration file has to be created. In this file manufacturer specific, package independent content can be specified.

The file name is formed from the commercial identifier of the manufacturer (according to the global manufacturer database, see above). The file extension is `.cfg`.

The manufacturer registration files are located in the directory `<data>/registry`.

Example:

```
<data>/registry/MAN.cfg
```

### 4.2.1 Manufacturer registration format

The format of the manufacturer registration files is equal to the format of the package registration files (see 5.1). The file is separated in two parts: a language independent `[general]` section and several language-specific `[<ISO-identifier>]` sections. In principle each key can be used both in section `[general]` as well as in language-specific sections. The application always is looking for a key first in the relevant language-specific section.

In principle each key of the package registration can be used in the manufacturer registration too[9]. This key will be used if it is not specified in the package registration.

[8]Here, a *manufacturer* is generally defined as any entity that provides and distributes OFML data. In addition to actual manufacturers of products, this can also be an association or similar.

[9]However, in practice this makes sense not for all keys.

Example:

```
[general]
address.name = EasternGraphics GmbH
address.street = Einsteinstrasse 1
address.city = Ilmenau
address.zip = 98693
address.country = Deutschland
address.tel = 03677 6782-0
address.fax = 03677 678250
address.email = info@EasternGraphics.com
address.www = www.EasternGraphics.com
ppr_region_id = GER

[de]
manufacturer_name = Standard

[en]
manufacturer_name = Default
address.country = Germany
```

### 4.2.2 Manufacturer registration keys

[1] ***concern_id*** **[O]**
– defines the unique identifier of the concern the manufacturer belongs to

The identifier may contain alphanumeric characters including the underscore, where the first character must be a letter:

`[a-zA-Z][a-zA-Z0-9_]*`

**Attention**: This key needs to be registered (see 4.1).

Note:
The identifier can contain both uppercase and lowercase letters, but the same correct spelling must be maintained at each use of the identifier. There must not be two identifiers, which differ only in the case of the characters.

[2] ***manufacturer_name*** **[M]**
– defines the (language dependent) name of the manufacturer

[3] ***ppr_region_id*** **[O]**
– determines the region key to be used in the price profile of the manufacturer.

This key will be referenced in the price profile in column $country$ of table $profile$ (see [ppr]). (See also 6.6)

`Example: ppr_region_id = BENELUX`

[4] ***address.name* [O]**
[5] ***address.street* [O]**
[6] ***address.postbox* [O]**
[7] ***address.city* [O]**
[8] ***address.zip* [O]**
[9] ***address.state* [O]**
[10] ***address.country* [O]**
[11] ***address.tel* [O]**
[12] ***address.fax* [O]**
[13] ***address.email* [O]**
[14] ***address.www* [O]**

The meaning of these keys is self explaining (see also the example above), so the document forgoes a detailed description of the address keys.

A special feature of these keys is that the value can be a multiline text. For this purpose, an index must be appended to the key as a postfix. The order of the indices and not the sequence of the lines determines the composition of the text.

[15] ***sort_names* [O]**
– forces the alphanumeric sorting of the series according to the localized series names within the representation in the software system.

By allocation with *false* (standard) no sorting will happen, the series will be shown in order like defined in the data profile (see 3.2 on page 8).

```
Example: sort_names = true
```

[16] ***release_note* [O]**
– defines an arbitrary text for the manufacturer

This text can contain package independent information. A software system could display the text in the catalog view, for example.

```
Example: release_text = Version 2.3 with price alignments from April the 1st
```

[17] ***gln_id* [O]**
– specifies the GLN (Global Location Number) of the manufacturer/concern (see [gln])

[18] ***series_name.<program_id>* [O]**
– Name of commercial series *program_id*

The name can be used by the application, e.g., in selection dialogs.

If this key is not specified for a given commercial series *program_id*, the application must try to determine the appropriate OFML library by means of commercial identifiers of manufacturer and series ID, in order to determine the name (key *program_name*) from the registration file. However, this procedure is not reliable, because there is not always a clear relationship between commercial series and OFML library.

```
Example: series_name.EX = Example
```

[19] ***external_catalog.url* [O]**
[20] ***external_catalog.name* [O]**

The *external_catalog.url* key must appear only once in section `[general]`. If the value of the key is a valid HTTPS URL, any catalog data stored in the manufacturer's OFML packages will be ignored and the application will offer a connection to the **external catalog** defined by the URL instead.

The key *external_catalog.name* can be used to specify language-specific names for the catalog. If no key exists in the corresponding language-specific section for the catalog language set in the application, the key from section `[general]` is used. If the key does not exist there either, the name of the manufacturer will be used (see key *manufacturer_name* above).

External catalogs are *not* supported in all pCon applications. In order to insert articles from the external catalog into a project, the external catalogs must serve a special, application-specfic API[10].

Example:

```
[general]
external_catalog.url=https://www.example.com
external_catalog.name=Example Catalog

[de]
external_catalog.name=Beispielkatalog
```

### 4.2.3 Storage of manufacturer resources

Manufacturer resources are stored in the directory `<data>/registry/($manufacturer)`.

The following resources are defined:

Manufacturer logo:

```
registry/($manufacturer)/manufacturer.jpg
[registry/($manufacturer)/manufacturer.png]
```

- Image size: [1 - max. 200] x 40 (width x height in pixels)

Large manufacturer logo (optional):

```
registry/($manufacturer)/($manufacturer)_l.jpg
[registry/($manufacturer)/($manufacturer)_l.png]
```

- The image may have a maximum edge length of 2000 pixels.
- As a recommendation, the longest edge should be at least 1000 pixels long.
- There is no specification about the aspect ratio.
- The applications are responsible for correctly processing images with different sizes.

The format of the image file has to be JPEG or PNG[11]. If the logo is given in JPEG format as well as in PNG format, PNG takes precedence.

[10] Details can be obtained from the product managers or the support.
[11] For general image format conventions see section 6.5.

## 4.3 Concern registration

The registration of a concern (corporate group) works similar to the manufacturer registration. It is preceding to the manufacturer registration like selfsame precedes the package registration.

For each concern a registration file can be created. Concern-specific content can be stored in this file.

The file name is formed from the identifier of the concern (according to the global manufacturer database, see 4.1). The file extension is `.cfg`.

The concern registration files are located in the directory `<data>/registry`.

### 4.3.1 Concern registration format

The format of the concern registration files is as far as possible identical to the manufacturer registration.

Each key of the manufacturer or package registration can be defined in the concern registration too. This key will be used if it is not defined in the manufacturer or package registration.

Example:

```
[general]
concern_id=egr

[de]
concern_name = EasternGraphics GmbH
```

### 4.3.2 Concern registration keys

The following keys are defined additionally for the concern registration according to the manufacturer registration:

[1] ***concern_name* [M]**
– defines the (language dependent) name of the concern

### 4.3.3 Storage of Concern resources

Concern resources are stored in the directory `<data>/registry/($concern)`.

The following resources are defined:

Concern logo:

```
registry/($concern)/concern.jpg
[registry/($concern)/concern.png]
```

- Image size: [1 - max. 200] x 40 (width x height in pixels)

Large concern logo (optional):

```
registry/($concern)/($concern)_l.jpg
[registry/($concern)/($concern)_l.png]
```

- The image may have a maximum edge length of 2000 pixels.
- As a recommendation, the longest edge should be at least 1000 pixels long.
- There is no specification about the aspect ratio.
- The applications are responsible for correctly processing images with different sizes.

The format of the image file has to be JPEG or PNG[12]. If the logo is given in JPEG format as well as in PNG format, PNG takes precedence.

[12]For general image format conventions see section 6.5.

# 5 Registration of OFML packages

For each package which is to be used in a pCon application a registration file has to be provided. This way the content, the structure and different constraints for the package will be specified.

Regarding repository of these files see section 2.1 and regarding integration within data resp. catalog profiles see sections 3.2 resp. 3.3.

## 5.1 Package registration format

The registration file format consists of (language specific) sections, keys and corresponding values. Blank lines and comments (initiated with a number sign '#') are allowed.
Language dependent values are specified in its own section. The name of the section results from the double-digit ISO language code (ISO–639-1) (see 5.4). The denotation of keys and values will be explained in the next section.

## 5.2 Registration

In the data root directory the directory `<data>/registry` is located. In that directory the registration files are stored. The name of a package registration file has to be created out of the keys which are recorded in the file. The file name format follows this convention[13]:

```
($manufacturer)_($program)_($region)_($version).cfg
```

The package registration files are referenced by their names from data profiles (see section 3.2) resp. catalog profiles (see section 3.3).

## 5.3 Package registration keys

[1] ***encoding* [O]**
– defines the encoding of the registry file and the package itself

This key should be specified only if all data in the package possesses the given character encoding. The applications then try to decode the data accordingly, regardless of the encoding, specified by the system locale.

Possible values are:

| Encoding | Note |
|---|---|
| *ISO-8859-1* | ISO–Latin-1 (default) |
| *UTF-8* | according to standard ISO/IEC 10646-1 (UCS Transformation Format 8 Bit) |

```
Example: encoding = ISO-8859-1
```

[13] Conversely, the values for `program` and `region` can not be uniquely determined by string separation from the name of a package registration file (since program identifiers themselves can contain an underscore).

[2] ***manufacturer*** **[M]**
– defines the unique OFML identifier of the manufacturer

The identifier may contain alphanumeric characters, where the first character must be a letter and letters must be lowercase:

```
[a-z][a-z0-9]*
```

**Attention**: This key needs to be registered (see 4.1).

The identifier of the manufacturer forms the path name on the first level of the directory structure defined in section 2.1.

```
Example: manufacturer = ofml
```

[3] ***program*** **[M]**
– defines the unique OFML identifier of the OFML library (series) within the manufacturer

The identifier may contain alphanumeric characters including the underscore, where the first character must be a letter:

```
[a-zA-Z][a-zA-Z0-9_]*
```

The identifier of the OFML package forms the path name underneath the manufacturer level (see section 2.1).

```
Example: program = goiex
```

[4] ***manufacturer_id*** **[M/O]**
– defines the unique commercial identifier of the manufacturer

The identifier may contain alphanumeric characters, where letters must be uppercase:

```
[A-Z0-9]*
```

**Attention**: This key needs to be registered (see 4.1).

Mandatory key if a manufacturer registration exists and for packages of type *product*. Each identification code must be used uniformly in all packages of the manufacturer. An explicit one-to-one mapping *manufacturer_id* ↔ *manufacturer* has to be guaranteed.
Also in referencing packages only one identification code is allowed, i.e. referencing several manufacturers at the same time is not possible.

```
Example: manufacturer_id = EG
```

[5] ***program_id*** **[M/O]**
– defines (within a manufacturer) the unique identifiers of the commercial series[14] mapped in this OFML package

Mandatory key for packages of type *product*. For packages of type *catalog* the identifiers of all commercial series of the referenced packages have to be declared here if they are not defined in the registration file of these packages.

A series ID must not contain more than 16 characters. Only the upper case letters and digits from the ASCII character set as well as the underscore '_' should be used.
The usage of the following characters is *not* recommended: `\/?:*"><|,;=` as well as the space character[15].

[14] see [glos]
[15] If these characters are used, correct data processing on all platforms and in all applications of EasternGraphics GmbH is *not* guaranteed.

Each identification code should be used only once in all OFML packages of one manufacturer. This ensures an one-to-one reverse mapping *program_id* $\rightarrow$ *program*. Several identification codes are separable by semicolon. The identification code has to be exactly the same as stored in the commercial data.

If commercial series (*program_id*) are shared within different OFML packages the following requirements have to be fulfilled:

1. the article number must be unique within the manufacturer (no repetition in other OFML packages)

   AND

2. the article numbers must be present in the catalog of the respective OFML library

```
Example: program_id = GX
         program_id = L1;L2;Z1
```

[6] ***distribution_region*** **[M/O]**
– defines the identifier of the distribution region of the package

Distribution regions represent a logical separation for the repository of catalogs and commercial data. Possible identifiers for distribution regions are the state codes according to ISO–3166-1 Alpha-2 resp. Alpha-3 (see [iso3166]) or any other term containing the characters [a-z, A-Z, 0-9] and starting with an allowed letter. The identifier for the distribution region forms the directory name underneath the series (see 2.1). Mandatory key for packages of type *product* and *catalog*.

```
Example: distribution_region = DE
```

[7] ***release_version*** **[M]**
– defines the version number for the registered package

There are different regulations for the structure of the version number depending whether the package is referenced in the data profile or in the catalog profiles of the manufacturer:

- If the package is referenced in the data profile, the version number has to be specified in the form `Major.Minor.Build`.
  The parts are whole numbers, major numbers starting with 1, others with 0.
  For incrementing the version number after changes in product data, the following regulations are recommended:

  | | |
  |---|---|
  | Build | Incrementing the Build number means a *minimal* change, error fixes in graphical data, pricing, etc. |
  | Minor | Incrementing the Minor number means beyond going changes, e.g. removal or adding of articles, properties, property values, etc. Even a normal price update requires a increment of the Minor number. |
  | Major | Incrementing the Major number means serious changes like incompatible changes in graphical data. |

  If the package also is referenced in catalog profiles[16], minor and build numbers are irrelevant, i.e. always have to be stated as 0.

- If the package only is referenced in catalog profiles, the version number is single whole number greater than 0 (i.e. consists only of a major number).

[16] In this case the data profile acts as the so-called compatibility data profile.

In both cases, before releasing a changed OFML package the release version has to be incremented. i.e. the version number is strictly monotonic increasing.
An installation routine for OFML packages with this version number then can decide whether an update of the library is necessary.

The version number is independent of the distribution region. This way several packages within one manufacturer can have identical version numbers. Then, these numbers have to have different distribution regions.

```
Example: release_version = 1.3.1
```

[8] ***release_date*** **[M]**
– defines the date of the package release

The date format has to be specified according to ISO date format (ISO–8601): YYYY-MM-DD (see [iso8601])

```
Example: release_date = 2004-01-17
```

[9] ***release_timestamp*** **[O]**
– specifies the final creation time of the package.

The time is normalized to UTC (s. [utc]) and given in the following format: `yyyyMMddhhmmss`

```
Example: release_timestamp = 20100204145736
```

It is recommended to specify the key *release_timestamp*, because it allows the applications – together with the keys *release_version* and *release_date* – to reliably detect whether a new package version is installed[17].

[10] ***release_state*** **[O]**
– defines the release status of the package

Possible values are: *alpha, beta, rc, test, final*. For all none empty values deviating from *final* the software system displays a message (see *release_text*).

```
Example: release_state = final
```

[11] ***release_text*** **[O]**
– defines a state text which can be displayed by the software system

(see *release_state*).

```
Example: release_text = Non official test version!
```

[12] ***version*** **[M]**
– defines the major number of the registered package

This version must be identical with the major number of the key *release_version*.
The version number builds the name-space inside the OFML package and a directory underneath the distribution region. Both must be maintained synchronously within one library (see 2.1.)

```
Example: version = 1
```

[17] compared to the time when a given project was last saved or updated

[13] ***languages*** **[M]**
– defines the languages supported by the package

All language identifier are double-digit lower case values according to ISO language codes (ISO–639-1). Several languages are separable by semicolon.
If the software system demands a language not mentioned here, the standard language of the package will be used. The standard language is the first termed language in the list.

```
Example: languages = de;en;fr;nl;es
```

[14] ***type*** **[M]**
– defines the type of the package

The following types are allowed:

| Type | Description |
|---|---|
| *found* | basic OFML library without product and catalog data |
| *product* | OFML library with product data and optional catalog data |
| *catalog* | Catalog data without product data |
| *extension* | loadable extension library |

```
Example: type = found
```

[15] ***productdb*** **[O]**
– contains the class name for the product database

The full qualified OFML class name (see [ofml]) is required.

```
Example: productdb = ::ofml::xoi::xOiOCDProductDB21
```

[16] ***productdb_path*** **[O]**
– defines the path to the files of the product database

The path is relatively to the data root directory.

```
Example: productdb_path = ofml/goiex/DE/1/db
```

[17] ***oam_path*** **[O]**
– defines the path to the OAM database

This key is maintained like *productdb_path*.
By default: `<productdb_path>/../oam`

```
Example: oam_path = ofml/goiex/DE/1/oam
```

[18] ***mddb_path*** **[O]**
– defines the path to the MetaDialog database

This is located by default in:

```
<data>/($manufacturer)/($program)/($version)
```

This key is specified like *productdb_path*.

```
Example.: mddb_path = ofml/goiex/1/md
```

[19] ***proginfo* [O]**
– defines the name of the program information class

The full qualified OFML class name (see [ofml]) is required.

```
Example: ::ofml::goiex::gOiExProgInfo
```

[20] ***proginfodb_path* [O]**
– defines the path to the program information database

This is located by default in:

```
<data>/($manufacturer)/($program)/($version)
```

The path is relatively to the data root directory. It is effective only if in the default path is no info DB (That means, an Info-DB in the default path of the series has precedence over an possibly existing DB in proginfodb_path).

```
Example: proginfodb_path = ofml/oi/1
```

[21] ***category* [M/O]**
– determinates a category for the library

A category defines in which planning hierarchy objects from the library are inserted. Mandatory key for packages of type *product*.

| Category | Description |
|---|---|
| *furniture* | Furniture and articles placed like furniture |
| *building* | Room or building elements |
| *undefined* | Basic or not assignable library |

```
Example: category = furniture
```

[22] ***depend* [O]**
– defines the packages which are required before loading this package (dependencies).

Several packages are separated by semicolon.
The list should be irreducible, i.e. contain only packages directly required for this package. Keep in mind that the given packages may have dependencies, too (risk of cyclic dependencies).

Key syntax: `::($manufacturer)::($program)::($release_version)/($distribution_region);...`

The declaration of minor and build numbers in *`release_version`* is optional.

The release version will be verified by the application if the present package was loaded from a data profile, or if required package is not listed in the catalog profile, from which the present package was loaded[18].

For catalog profiles compliance with the dependencies between the packages listed therein is presupposed. The applications processing catalog profiles are free whether they verify the dependencies anyhow. Alternatively, verification also may take place whithin the installation routine.

If the release version is verified by the application, the following conditions must be fulfilled in order that the currently installed version of the required package can be used:

- If the required package is not listed in the data resp. catalog profile, the major number of the installed version has to exactly match the major number given in this key. Otherwise, the major number of the installed version must be greater than or equal to major number given in this key.
  If both major numbers identical, and if the minor number is given in this key, the minor number will be examined, too:
- The minor number of the installed version must be greater than or equal to the minor number given in this key.
  If both minor numbers identical, and if the build number is given in this key, the build number will be examined, too:
- The build number of the installed version must be greater than or equal to the build number given in this key.

```
Example: depend = ::ofml::goi::1.2.0/ANY;::ofml::xoi::1.11.10/ANY
```

[23] ***catalogs* [M/O]**
– specifies the libraries referencing articles from this library within its catalog data

If articles of the present package are referenced in the catalog data from other libraries, these libraries *have to* be specified in this key, otherwise, in certain application scenarios the catalog entry required for the insertion of an article can not be determined or searching the catalog for an article can fail.

Several libraries have to be separated by semicolon.

Key syntax: `::($manufacturer)::($program)::;...`

The application uses the packages of the specified OFML libraries, which are listed in the same data resp. catalog profile from which the present package has been loaded.

```
Example: catalogs = ::egr::office::;::egr::office_np::
```

[24] ***oap_program* [M/O]**
– specifies the library that contains the OAP data for the articles of the present library

If the key is missing, the OAP data [oap] is expected in the present library itself.

Key syntax: `::($manufacturer)::($program)::`

The application uses the package of the specified OFML library, which is listed in the same data resp. catalog profile from which the present package has been loaded.

[18]The latter normally only applies to basic OFML libraries, because they are not provided by the manufacturer itself.

[25] ***pd_format* [M/O]**
– defines the used product data format

This key is mandatory for packages containing commercial data. All other packages explicitly should use the value `NULL`.

The following formats are defined:

- *OCD_2.1*, *OCD_2.2*, *OCD_3.0*, *OCD_4.0*, *OCD_4.1*, *OCD_4.2*, *OCD_4.3*
- *NULL* (no commercial product data)

```
Example: pd_format = OCD_4.3
```

[26] ***meta_type* [O]**
– indicates whether an instantiation in context of OFML–MetaTypes is possible.

If this key is not empty and an OFML–MetaType DLM licence for the manufacturer series exists, articles might be encapsulated by a OFML–MetaType.

The given function in the second part of the value will be used to identify the article specific OFML–MetaType.

The definition of the OFML type before the function is necessary to guarantee the evaluation of the OFML classes. The given function has to be implemented for the OFML type explicitly or implicitly.

```
Example: meta_type = ::ofml::go::GoMetaType;::ofml::go::goGetMetaType()
```

[27] ***features* [O]**
– defines special functions allowed for this package

Several functions are separated by semicolon.

The following function keys are available:

| Key | Description |
|---|---|
| *showInitProgressDlg* | A progress dialog is displayed during initialization of the product database. |
| *denyFreePos* | Free positioning is denied for elements of this package. |
| *deny3DFdbMove* | Free moving for elements in 3D is denied for elements of this package. |
| *editableCatalog* | Users are allowed to edit the catalog at runtime. |
| *programProperties* | This package provides static properties on the ProgInfo object. |
| *useMetaDialogs* | This package provides specific configuration dialogues. |
| *maySetFinalArticleSpec* | Has to be specified, if the user is enabled to enter (possibly incomplete) final article numbers in order to create specific article variants (i.e., the article will be created with the configuration encoded by the final article number)[a]. |

[a]For reasons of performance and in order to avoid potential problems during evaluation of final article numbers, this feature should be set only if the coding scheme really allows for the processing of entered (possibly incomplete) final article numbers.

```
Example: features = editableCatalog;showInitProgressDlg
```

[28] ***series_type*** **[O]**
– defines specific types for the package

This key controls type specific handlings. Several types can be separated by semicolon.

The following values are defined:

| Value | Description |
|---|---|
| *go_meta* | support of OFML–MetaTypes 1.x |

```
Example: series_type = go_meta
```

[29] ***special_article_scheme*** **[O]**
– determines a specific formatting for article number of special models

The value can contain any alphanumeric characters and the substitution characters '?' and '*'. A '?' will be replaced by a characters of the original basic article number. The replacement take place in order from the first character of the basic article number. A '*' will be replaced by the whole basic article number. By default 'SPECIAL *' is used (language dependent).

```
Example: special_article_scheme = CS??????
```

[30] ***~~progid_3d~~(obsolete)*** **[O]**
– determines the series component for the 3D structure information

This value is used by graphic exports which are supporting object names in their structure (e.g. 3DS).

```
Example: progid_3d = GOI
```

[31] ***~~layer_progid_2d~~(obsolete)*** **[O]**
– determines the series component for the 2D layer names

This value is used by layer oriented graphic exports (e.g. DXF).

```
Example: layer_progid_2d = EGRGOIEX2D
```

[32] ***~~layer_progid_3d~~(obsolete)*** **[O]**
– determines the series component for the 3D layer names

This value is used by layer oriented graphic exports (e.g. DXF).

```
Example: layer_progid_3d = EGRGOIEX3D
```

[33] ***~~block_progid_2d~~(obsolete)*** **[O]**
– determines the series component for the 2D block names

This value is used by block oriented graphic exports (e.g. DXF).

```
Example: block_progid_2d = EGRGOIEX2D
```

[34] ***~~block_progid_3d~~(obsolete)*** **[O]**
– determines the series component for the 3D block names

This value is used by block oriented graphic exports (e.g. DXF).

```
Example: block_progid_3d = EGRGOIEX3D
```

[35] ***~~geo_export_params~~(obsolete)*** **[O]**
– determines special parameters for the export of graphical data.

Several parameters are separable by semicolon. The following values are allowed:

| Value | Description |
|---|---|
| *create_hashed_mat_names* | Converts material names to hash values for the mapping towards generated MLI file. |
| *create_hashed_geo_names* | Converts geometry names to hash values. |

Options for article geometry creation in DWG supporting applications:

| Value | Description |
|---|---|
| *use_proxy_geometries* | Create mesh data from geometry description. Slower and not always solid, but small data and best possible quality. (Default) |
| *use_generated_meshes* | Use supplied pre-generated mesh data. Quick and solid, but larger data and not always at best possible quality. |

Note:
The values *use_proxy_geometries* and *use_generated_meshes* are exclusive, they can not be combined.

```
Example: geo_export_params = create_hashed_mat_names;create_hashed_geo_names
```

[36] ***gtin_id*** **[O]**
– This key specifies the GTIN (Global Trade Item Number) for the package (see [gtin]).

The length of the key may be 8, 12, 13 or 14 digits, according to the different schemes GTIN–14, GTIN–13, GTIN–12 or GTIN–8.

[37] ***cat_type*** **[O]**
– defines the catalog type for this package.

The key structure follows this form:

```
cat_type=<Type>[:<Version>[.<MinorVersion>[.<SubMinorVersion>]]][/<PhysicalFormat>]
```

The following types are defined:

| | |
|---|---|
| `XCF` | catalog in XCF Format (Default) |
| `OAS` | catalog in OAS Format |
| `NULL` | Package without catalog<br>(This explicitly prevents the display of any existing catalog data.) |

The (optional) version specification indicates the version used for catalog types with versioned data formats.

Currently, the following physical formats are defined (optional):

- *CSV* (Default)

```
Example 1: cat_type = XCF:1.8/CSV
Example 2: cat_type = NULL
```

[38] ***masked_catalogs*** **[O]**
– Determines the visibility of a catalog in specific applications

The catalog data of a package can be marked as „not to show“ in certain applications. The applications are defined by the standard module keys and an optional version number. If specified the version number part requires an exact match. The package is still registered and if necessary loaded but not shown regularly.

The formation rule of the key is:

```
masked_catalogs=<ModuleKey>[:<Version>[.<MinorVersion>[.<SubMinorVersion>]]][;...]
```

Version refers to the version of each module.

```
Example: masked_catalogs = P-PL-X:6.3.0;P-XCAD:2
```

[39] ***persistency_form*** **[O]**
– determines the persistent representation of articles of the package.

The following values are allowed:

| Value | Description |
|---|---|
| *SCENE* | Article instances have to be stored as a complete OFML scene. |
| *STATECODES* | Articles can be saved by their state codes, which completely describe the elements. |

Mode *STATECODES* results in a smaller size of the saved project, but assumes that all articles of the package are actually completely described by their state codes.

The default value for `persistency_form` is *STATECODES*.

```
Example: persistency_form = STATECODES
```

Note:
So far, the key has only been evaluated by the pCon.planner. For the release of pCon.planner in autumn 2023 this key will be discontinued! All articles then will be saved uniformly using the OBX format.

[40] ***add_gfx_symbols.<format>*** **[O]**
– defines the OFML library, which contains additional graphic symbols in the specified format

The additional graphic symbols can be used by an application to generate a graphic for articles from the present package in the specified format.

Key syntax: `::($manufacturer)::($program)::`

The application uses the package of the specified OFML library, which is listed in the same data resp. catalog profile from which the present package has been loaded.

Currently, the following formats are supported:

| Value | Description |
|---|---|
| *skp* | Sketchup |

## 5.4 Language dependent keys

The registration files can contain several section of language dependent texts. A section will be started with a section name within squared brackets. The name of the language dependent sections corresponding to the double-digit ISO language codes (ISO–639-1) (see [iso639-1]). The succeeding keys are permitted to use in each of these sections. Each key overwrites a possibly given value in a manufacturer resp. concern registration.
Language independent keys are not found in a language dependent section and are not allowed there.

[41] ***manufacturer_name* [O]**
– defines the name of the manufacturer resp. supplier

[42] ***program_name* [M]**
– defines the name of the library

```
Example: program_name = Office elements
```

[43] ***copyright* [O]**
– contains the copyright note

```
Example: copyright = EasternGraphics 2004
```

[44] ***description* [O]**
– contains a short description for the library

```
Example: description = Examples for the creation of OFML and product data
```

## 5.5 Example

`egr_office2_EGR_1.cfg`:

```
# manufacturer information
manufacturer=egr
manufacturer_id=EG

# package information
program=office2
program_name=Office, V.2
program_id=OFFICE2

# catalog information
type=product
category=furniture
languages=en;de;es;fr;it;nl;pl;ru
distribution_region=EGR
version=1
release_version=1.0.19
release_date=2016-02-23
release_timestamp=20160225135032
cat_type=XCF

# product database
pd_format=OCD_4.0
productdb=::ofml::xoi::xOiNativeOCDProductDB40
productdb_path=egr/office2/EGR/1/db
oam_path=egr/office2/EGR/1/oam

# program information
proginfo=::ofml::xoi::xOiProgInfo
proginfodb_path=egr/office2/1

# relations and dependencies
depend=::ofml::go::1.14.9/ANY

# additional settings
series_type=go_meta
meta_type=::ofml::go::GoMetaType;::ofml::go::goGetMetaType()
geo_export_params=use_proxy_geometries

[de]
program_name=Büroelemente

[en]
program_name=Office elements
```

# 6 Data types and file types

## 6.1 OFML and picture data

OFML data of an OFML package will be stored independent of the distribution region underneath the directory:

```
<data>/($manufacturer)/($program)/($version)
```

This directory defines the name-space. Out of it results the formation regulation for the name of the OFML archive (`*.alb`). The following file formats are among to the OFML and graphic data:

| Use | File name | Extension |
|---|---|---|
| OFML archives | $manufacturer_$program_$version | alb |
| OFML classes* | $name | cls |
| Text resources* | $program_$lang-code | sr |
| 3D geometries* | $name | geo, 3ds, dwg |
| 2D symboles* | $name | egms, fig |
| Textures/image maps* | $name | tga, jpg, bmp, png |
| OFML materials* | $name | mat |
| EBase database | $name | ebase |
| CSV database | $name | csv |
| Material libraries | $name | mli |

The file name is formed by the name of the object ($name) or by values ($manufacturer), ($program) in the registration files.
All types marked with * could and should be stored in the OFML archive. For extended data distribution concepts this will likely get mandatory.
CSV databases should be removed from distribution data after the conversion to the high-performance EBase format[19].

## 6.2 OFML product data

Product data[20] will be stored underneath the directory

```
<data>/($manufacturer)/($program)/($region)/($version)/db
```

in a database.

## 6.3 XCF catalog

The catalog information of the OFML package in XCF format [xcf] will be stored within the directory

```
<data>/($manufacturer)/($program)/($region)/($version)/cat
```

[19]EBase is a database in binary format developed by EasternGraphics for read-only access. An EBase database is generated by means of a table description file from text files with fixed or variable field length (e.g. CSV). The tool for creating an EBase database and the description files for the different types of OFML data can be requested from EasternGraphics.
[20]Here, in the narrower sense, the commercial data are meant.

## 6.4 Specific resources

Specific resources are stored separately in sub-directories of:

```
<data>/($manufacturer)/($program)/($region)/($version)
```

These can include HTML documents, pictures, videos and other data. The use of that data is optional for the software system. The predefined resource types are covered below.

Resources of other types not explicitly mentioned below are to be stored in directory:

```
<data>/($manufacturer)/($program)/($region)/($version)/etc
```

### 6.4.1 Catalog images

The image files referenced from the catalog are stored in the directory:

```
<data>/($manufacturer)/($program)/($region)/($version)/image
```

The image data can also be used for the printout or for a reference in HTML-documents.

The format of the image file has to be JPEG or PNG[21].

The files in this directory can be also stored in a ZIP archive `image.zip` (see also section 2.1).

### 6.4.2 Material images

Material images are picture data that are used for the visualization of material characteristics or, more generally, of property values[22].

The names of the image files (without filename extension) are determined based on the property key and the property value using the OFML interface *Property*.

The graphic file format used is determined by the file name extension. JPEG and PNG formats are allowed for material images, with preference given to a PNG format file[23].

Two ways of storing the material images are supported:

1. This older way supports a small icon with default image size of 50 x 18 pixels (width x height) and a larger square image with default image size of 50 x 50 pixels.

   These are stored in the directory:

   ```
   <data>/($manufacturer)/($program)/($region)/($version)/mat
   ```

   The file name for the square image is formed by appending an underscore to the actual image name.

   Example:

   small:
   `<data>/($manufacturer)/($program)/($region)/($version)/mat/material.jpg`
   square:
   `<data>/($manufacturer)/($program)/($region)/($version)/mat/material_.jpg`

[21] For image format conventions see section 6.5.
[22] e.g. within the property editor
[23] For general image format conventions see section 6.5.

2. In order to give the applications more design options for the presentation[24], in this new way the material images can be provided in following 3 sizes:

    - small icon for compact presentation: 200 x 72 (width x height in pixels)
    - medium (square) icon: 200 x 200 pixels
    - large image for the material resp. property value preview:
        - the image may have a maximum edge length of 2000 pixels
        - as a recommendation, the longest edge should be at least 1000 pixels long
        - there is no specification about the aspect ratio

    The files for the different image sizes are located in corresponding subdirectories of

    `<data>/($manufacturer)/($program)/($region)/($version)/mat`

    `s` – small icons
    `m` – medium icons
    `l` – large images

    The creation of the subdirectory `l` is optional. In addition, it is not necessary to store an image there for all property values. An image only needs to be stored if a material resp. property value preview is required for a given value.

    Example:

    small:
    `<data>/($manufacturer)/($program)/($region)/($version)/mat/s/material.jpg`
    medium:
    `<data>/($manufacturer)/($program)/($region)/($version)/mat/m/material.jpg`
    large:
    `<data>/($manufacturer)/($program)/($region)/($version)/mat/l/material.jpg`

For a visually good representation, the following general recommendations apply:

- Material images should not have unnecessary white space.
- Material images must not have a frame/border or the like.

The display of material images by the applications is optional and the form of the presentation depends on the application. The applications are responsible for scaling and displaying the images correctly.

If there is no subdirectory `s` or no subdirectory `m`, the previous smaller images will continue to be used (according to the older way of data creation).
If there is a subdirectory `s` *and* a subdirectory `m`, only the images stored there will be used by applications that support this new way of data creation. This means that there is no fallback to images according to the older way of data creation if there are no images in subdirectory `s` resp. subdirectory `m` for individual property values.

Thus, if all applications are to be supported with the best possible display quality, the material images must be provided according to both the older and the new way of data creation.

The files in directory (including subdirectories with the second storage variant)

`<data>/($manufacturer)/($program)/($region)/($version)/mat`

can be also stored in a ZIP archive `mat.zip` (see also section 2.1).

[24]including support of high-resolution displays

### 6.4.3 HTML files

HTML files referenced in the catalog data can be stored in the directories:

```
<data>/($manufacturer)/($program)/($region)/($version)/html/($language)
<data>/($manufacturer)/($program)/($region)/($version)/html
```

For each supported language a subdirectory can be created with the respective two-letter ISO language code (ISO 639-1) as the directory name. A referenced HTML file always first is looked for in the language-specific directory according to the catalog language currently set in the application, regardless of whether the associated resource in the catalog itself is language-specific or not.

HTML files can reference image data from the picture data directory.

### 6.4.4 Configurations and geometries

Configurations (.fml), OFML groups (.ogrp), geometries (.3ds, .dwg) as well as container files (.pec) referenced in the catalog data are stored here:

```
<data>/($manufacturer)/($program)/($region)/($version)/etc
```

### 6.4.5 Article specific view setup

A table with article specific view setup can be stored in:

```
<data>/($manufacturer)/($program)/($region)/($version)/etc/artsetups.csv
```

This views are interpreted application specific. The table format is described in [asv]. Primarily the table from that OFML package is employed whose catalog data was used to create the article. Alternatively the table from that OFML package is employed which is referenced in the catalog data.

## 6.5 Image format conventions

Images in **JPEG** format have to comply with the specification of the "JPEG File Interchange Format (JFIF)"[25]

- have to be sequentially structured (not interlaced/progressive)
- have to use Huffman coding (not arithmetic coding)
- have to use the YCbCr color model (no CMYK, no black/white)
- have to use 8 bit per color channel (no more)

Images in **PNG** format have to comply with the "PNG (Portable Network Graphics) Specification, Version 2.2"[26]:

- have to be sequentially structured (not interlaced/progressive)
- have to use the RGB color model (3 color channels, not less, i.e., no black/white)
- have to use 8 bit per color channel (no more)
- optionally an 8 bit alpha channel can be used for transparent images
- animated PNGs (APNG) are not supported

Embedded metadata (thumbnails, EXIF, IPTC, ICC profiles, printing information) are ignored (not evaluated) in both formats.

[25] http://www.w3.org/Graphics/JPEG/jfif3.pdf
[26] http://libpng.org/pub/png/spec/1.2/png-1.2-pdg.html

## 6.6 Price profiles

Price profiles are described and specified in [ppr].

Price profiles installed in conjunction with catalogs of a manufacturer resp. a brand (see key *ppr_version*, section 3.3.2) have to be located in directory

```
<data>/($brand)/priceprofiles/($ppr_version)/
```

This means that for a catalog only one price profile can be specified and this has to be installed under the brand under which the catalog is listed (see key *brand*, section 3.3.2)[27].

Price profiles associated with a conventional dataset of a manufacturer have to be located in directory

```
<data>/($manufacturer)/priceprofiles/
```

If a manufacturer registers data both via a catalog profile and a compatibility data profile, accordingly 2 price profiles have to be provided in the above mentioned paths.

[27]Therefore, if the catalog contains data from multiple manufacturers, possibly existing manufacturer-specific price profiles have to be merged into a common price profile.

# 7 History

| | |
|---|---|
| 3.6.0 | - Clarification on the use of subdirectories in archive `mat.zip`[2.1] |
| | - The registration of manufacturers (section 4) is now described before the registration of packages (section 5) |
| | - More precise specifications in sections 4.1 and 4.2 |
| | - Index suffixes to specify a multiline text now are allowed only in the keys of group *`address`* for manufacturer registration[4.2.2] |
| | - New keys `external_catalog.url` und `external_catalog.name` for the registration of manufacturers[4.2.2] |
| | - Removed obsolete key *`module_depends`*[5.3] |
| | - Removed keys *`supplier_id`* and *`distributor_name`*[4.2.2,5.3,5.4] |
| | - Removed keys *`concern_id`*, *`ppr_region_id`* and *`concern_name`* from the package registration[5.3,5.4] |
| | - Key *`manufacturer_name`* is now optional in package registration[5.4] |
| 3.5.1 | - Marked keys *`progid_3d`*, *`layer_progid_2d`*, *`layer_progid_3d`*, *`block_progid_2d`*, *`block_progid_3d`* and *`geo_export_params`*[5.3] as obsolete (the corresponding behavior is part of the graphic export of the respective application) |
| 3.5.0 | - Removed obsolete data profile section *`[libdef]`*[3.2] |
| | - Removed obsolete value *`ISO-8859-2`* (ISO–Latin-2) for key *`encoding`*[5.3] |
| | - Key *`release_timestamp`*[5.3] is no longer obsolete (but now is optional) |
| | - Removed obsolete value `attrsel` for key *`type`*[5.3] |
| | - New key *`oap_program`*[5.3] |
| | - Updated supported product data formats in key *`pd_format`*[5.3] |
| | - Removed obsolete keys *`visibility`* and *`insertion_mode`*[5.3] |
| | - Removed not supported physical formats *`EBASE`*, *`DBF`* and *`SQLITE`* with key *`cat_type`* entfernt[5.3] |
| | - Default value for key *`persistency_form`* is now *`STATECODES`*[5.3] |
| | - Updated section 5.5 – Example(s) |
| | - The small manufacturer and concern logo is no longer supported, but a larger logo is now supported instead[4.2.3,4.3.3] |
| | - PNG support for manufacturer and concern logos by the applications is now mandatory[4.2.3,4.3.3] |
| | - Removed invalid reference to `xcf.ebase`[6.3] |
| | - Specification of the location of non-predefined resources[6.4] |
| | - Material images now may be provided also in PNG format[6.4.2] |
| | - A third image size now is supported for large material images[6.4.2] |
| | - New sizes for small and medium material images[6.4.2] |
| | - New structure for storing material images[6.4.2] |
| | - The location of container files is now specified[6.4.4] |
| | - Clarifications regarding image format conventions[6.5] |
| 3.4.1 | - More precise specification regarding default for key *`persistency_form`*[5.3] |
| 3.4.0 | - New value *`maySetFinalArticleSpec`* for key *`features`*[5.3] |
| | - Removed obsolete feature *`freeArticles`* (by now, conversion into special articles always is allowed)[5.3] |
| | - More precise specifications regarding keys *`encoding`*, *`distribution_region`*, *`visibility`* and *`persistency_form`*[5.3] |
| | - Recommendation regarding usable characters in key *`program_id`*[5.3] |
| | - More precise specification regarding HTML files referenced in catalog data[6.4.3] |
| | - Some minor corrections |
| 3.3.0 | - Changed regulations regarding price profiles in conjunction with catalogs[6.6] |
| | - Specified set of characters allowed in keys *`manufacturer`*, *`manufacturer_id`* and *`concern_id`*[5.3] |
| 3.2.0 | - New key *`add_gfx_symbols`*[5.3] |
| | - Corrected usage of terms "library" und "package" |
| | - Removed obsolete key *`extpath`* in data profiles[3.2.3] |
| | - More precise specification regarding declaration of packages in data resp. catalog profiles[3.2.3,3.3.4] |
| | - More precise specification regarding usage of packages in key *`catalogs`*[5.3] |
| | - Some minor corrections |
| 3.1.1 | - Clarification regarding usage of packages from several manufacturers in a catalog profile[3.3.4] |
| | - Clarifications regarding keys *`catalogs`* and *`pd_format`*[5.3] |
| 3.1.0 | - Clarification regarding usage of packages from several manufacturers in a catalog profile[3.3.4] |
| | - Clarification/correction regarding examination of dependencies between packages (key *`depend`*[5.3]) |
| | - Product data format EPL no longer is supported (key *`pd_format`*[5.3]) |
| | - Logo images optionally may be given in PNG format[3.1.3,4.2.3,4.3.3] |
| | - Catalog images may be given in PNG format[6.4.2] |
| | - New section regarding image format conventions[6.5] |
| 3.0.1 | - Correction regarding evaluation of release version given in key *`depend`* |
| 3.0.0 | - Introduction of catalog profiles[3] |
| | - Clarifications regarding registration of maufacturers[4.2.1,4.2.2] |
| | - Clarification regarding usable characters in key *`program`*[5.3] |
| | - Clarifications regarding keys *`release_version`*, *`depend`* and *`persistency_form`*[5.3] |
| | - Marked key *`release_timestamp`*[5.3] as obsolete |
| | - library type `attrsel` marked as deprecated |
| | - Removed key *`extension`* (use *`depend`* instead!) |
| | - Removed obsolete keys *`gf_version`* and *`xcf_format`* |
| | - Clarifications regarding the location of price profiles[6.6] |
| | - Reorganization of document structure |
| | - Minor corrections |

For older history see older versions.