EAIWS 4.16
Manual

for EAIWS 4.16 and EAILS 1.1

2024-10-22

Table of Contents

1 Prerequisites

1.1 Operating System

Please read the system requirements for more details. They can be found in the EasternGraphics Download Center https://download-center.pcon-solutions.com/?cat=7.

1.2 Java Runtime Environment

The EasternGraphics Online Configurator requires a Java 8 Runtime Environment. It has been tested with both the Java HotSpot(TM) 64-Bit Server VM (build 23.2) and the OpenJDK 64-Bit Server VM (build 23.2).

1.3 Port Mapper

For the EasternGraphics Online Configurator to be able to communicate with the license server the ONC/RPC port mapper must be running.

Red Hat Enterprise Linux 6 / CentOS 6: The ONC/RPC port mapper is implemented by `rpcbind(8)`. In the default configuration of the port mapper (operating in secure mode) an RPC service must not only use the loopback interface to register itself, but also a privileged port. As the license server is not supposed to be run as root, and contains no code to use a privileged port for registration with the port mapper, `rpcbind(8)` must be switched into insecure mode. One way to achieve this is to create the file `/etc/sysconfig/rpcbind` and add a line consisting of `RPCBIND_ARGS=-i` to this file.

2 Installation

A detailed installation guide can be found in the EasternGraphics Download Center https://download-center-.pcon-solutions.com/?cat=7

3 Licensing

The EasternGraphics Online Configurator requires various licenses which must be enabled for the host it is running on. Identification of the host is done through the name of the host and a host ID, short for host identifier.

3.1 Obtaining the Host Name and Host ID

The host name is usually known and equal to the output of the `hostname(1)` command. However, the host ID, as used by the license server of the Online Configurator, is different from the output of the `hostid(1)` command[1]. Assuming the license server is not already running the following can be done to obtain the host ID:

```
bash$ cd EAILS
bash$ java -jar LicenseServer.jar -log cons
NOTICE: Starting LicenseServer version 1.0.3
CONFIG: platform=amd64-linux
CONFIG: hostname=vaio, hostid=HHM5-N6MA-9ES2-DSEX
```

The license server can then be terminated using Control-C. Of course, the host name and host ID will be different from the values shown above.

If the license server is already running then the values can be obtained from the log file of the license server. The log files of the license server are usually in `EAILS/var/log`.

3.2 Installation of License Files

License files for a particular host must be copied into the `EAILS/etc/licenses` directory of the license server running on this host.

License files for hosts different than the host the license server is running on are ignored by the license server after a message is written to the log file.

3.3 License Features

A license file contains one or more license features licensed for the host identified in the license file. A license feature consists of the feature name, a version number, the expiration date (or 0000-00-00 if the feature does not expire), a number indicating how often the feature is available, and optional attributes, in that order.

The license features of all valid license files are combined by the license server into a pool of license features. This pool is used to serve license requests of the Online Configurator.

Once a feature has been requested by and leased to a particular instance of the Online Configurator it is no longer available for use by other instances of the Online Configurator. Leases are refreshed periodically by the Online Configurator. A leased license feature is moved back into the pool of available features if it is either explicitly released by the Online Configurator holding its lease, or if its lease is not refreshed for the duration of five minutes. Thus, the license features leased by a crashed instance of the Online Configurator eventually become available again without the need to restart the license server.

3.3.1 Feature `egr.eai.server`

Each instance of the Online Configurator (i.e. each process) leases this feature during it is starting up. If the feature is not available then the process terminates immediately.

1 On Linux, the host ID is computed based on the hardware address of `eth0`. If `eth0` does not exist then it is computed based on the hardware address of the first network interface found other than the loopback interface or a point-to-point interface.

Example:

```
    FEATURE egr.eai.server 1.0 0000-00-00 2
```

The example above allows two (additional) instances of the Online Configurator to run on the local host.

3.3.2 Feature `egr.eai.server.session`

Support for the license feature egr.eai.server.session was added in late 2016.

Important: Removed in EAIWS 4.1.

This feature is leased by the Online Configurator whenever a new session is opened using the `openSession` operation of the `SessionService`. It is released when the session is explicitly closed by the `closeSession` operation or after the session expired.

If this feature is not available (that is if the maximum allowed number of open sessions over all instances of the Online Configurator running on this host has been reached) then the `openSession` operation fails with a `SessionServiceFault`.

Example:

```
    FEATURE egr.eai.server.session 1.0 0000-00-00 20
```

The example above allows twenty (additional) concurrent sessions over all instances of the Online Configurator running on the local host.

3.3.3 Feature `egr.eai.server.throughput`

This feature has an attribute named `operations_per_minute` whose value is a positive integer specifying the allowed number of non-trivial state-changing web service operations[2] per minute. When the Online Configurator is starting up it tries to lease one or more instances of this feature to meet the required number of operations per minute as configured in the server start-up file (§4.1.4). To do so it first fetches all available throughput features, sorts them in descending order according to the number of operations per minute, iterates over the resulting list and leases the current feature if its number of operations per minute is less than or equal to the remaining number of required operations per minute. This approach has two important implications:

- A throughput feature will always be ignored if it supports more operations per minute than configured in the server start-up file. Thus, if the server is configured to support 200 operations per minute, but the only available throughput feature supports 500 operations per minute, the Online Configurator will fail to start as it would be unable to process any non-trivial state-changing web service operation.
- The operations per minute provided by a single throughput feature cannot be split between multiple instances of the Online Configurator. For instance, if there were three throughput features available, each providing 100 operations per minute, and two instances of the Online Configurator, each configured to process 150 operations per minute, the allowed throughput of each instance of the Online Configurator would be reduced to 100 operations per minute.

***Note:** Given these implications, one could be tempted to define license templates with many instances of the throughput feature, each one allowing a single operation per minute. This would be a bad idea, however, as leasing a single instance of a license feature requires a round trip to the license server, and such an approach could severely affect the start-up time of the Online Configurator (in the order of a second or so). So each instance of the throughput feature should probably allow at least 10 operations per minute, a smaller value being of questionable use anyway.*

If the number of allowed operations per minute had to be reduced during start-up due to insufficient throughput features then it will not ever be raised again for the current instance of the Online Configurator even if

2 Right now these are the `insertOCDArticle` and `setPropertyValue` operations of the `BasketService`.

more instances of the throughput feature become available.

Example:

```
FEATURE egr.eai.server.throughput 1.0 0000-00-00 10
operations_per_minute=100
```

The example above provides 10 instances of the throughput feature, and each instance allows 100 operations per minute, so in total this example allows 1000 operations per minute.

3.3.4 Feature `egr.eai.dsr.ofml_catalog`

The Online Configurator must be able to lease one instance of this feature for each registered catalog profile[3]. If no instance of this feature is available then a message will be written to the log file and the catalog profile will be ignored.

For an old-style manufacturer profile the number of required instances of this feature is equal to the number of different OFML manufacturers used by all product and catalog packages referenced from that profile.

If different session start-up files share the same catalog or manufacturer profile then this profile is counted only once.

The current implementation never releases an instance of this feature once it has been leased, even if the profile is no longer referenced by one of the session start-up files.

Example:

```
FEATURE egr.eai.dsr.ofml_catalog 1.0 0000-00-00 5
```

The example above allows the use of five catalog profiles over all instances of the Online Configurator running on the local host.

3.3.5 Feature `egr.eai.ws.project.ProjectStore`

The EAI-Server instantiates the project store if and only if the application feature 'egr.eai.server.ProjectStore' is available.

For the project web service to be able to use the project store, the application feature 'egr.eai.ws.project.ProjectStore' must be available and the session startup property with the same name must not be set to 'false'. Furthermore, the EAI-Server must support the project store (see above).

Both application features are mapped to license feature 'egr.eai.server.project_store'. So for the project store to be available, this license feature must be available and the session startup property 'egr.eai.ws.project.ProjectStore' must not be set to 'false'.

3.3.6 Further License Features

The Online Configurator tries to lease a single instance of the features below the first time they are used. If no instance of the feature is available at this point the feature will be permanently disabled for this instance of the Online Configurator.

Feature	Description
`egr.eai.server.ofml.multiple_catalogs`	Several catalogs of one manufacturer can be processed. Without this feature only one catalog per manufacturer will be processed.
`egr.eai.server.ofml.prices`	Prices will only be available if this feature is active.

3 The canonical path name of the profile is used to identify a profile for the purpose of licensing.

`egr.eai.server.export.image`	Image Export Feature (§5.6.3.40)
`egr.eai.server.export.gltf`	Export of GLTF geometry (format=GLTF)
`egr.eai.server.export.usd`	Export of USD geometry (format=USD)
`egr.eai.server.export._3ds`	Export of 3DS geometry (§5.6.3.43)
`egr.eai.server.export.dwg`	Export of DWG and DXF geometry (§5.6.3.43)
`egr.eai.server.export.skp`	Export of SKP geometry (§5.6.3.43)
`egr.eai.server.export.fbx`	Export of FBX geometry (§5.6.3.43)
`egr.eai.server.export.obj`	Export of OBJ geometry (§5.6.3.43)
`egr.eai.server.export.dae`	Export of Collada geometry (§5.6.3.43)
`egr.eai.server.export.gfx`	Export of GFX (§5.6.3.43)
`egr.eai.server.export.gfx_obx`	Export of GFX including OBX for commercial information (§5.6.3.43)
`egr.eai.server.export.igxc`	Export of IGXC (§5.6.3.43)
`egr.eai.server.export.gfj`	Export of GFJ (§5.6.3.43)
`egr.eai.server.export.ccl`	Export of CCL
`egr.eai.server.basket`	Basket functions are provided like application specific data (§5.3), project settings (§5.4.1.6), further basket item types (§5.6.1.7).
`egr.eai.basket.set_articles`	Set Article functions are available. (§5.6.3.25)
`egr.eai.basket.alternative_positions`	Convert ariticles to alternative positions (§5.6.1.27)
`egr.eai.server.save_load`	Enables the OBK save / load function (§5.4.3.14, §5.4.3.15)
`egr.eai.basket.copy_paste`	Enables the OBX copy / paste function for more than one article. (§5.6.3.51, §5.6.3.52)
`egr.eai.server.multiple_positions`	If this feature is available more than one article / aggregate can be inserted into a session.
`egr.eai.server.image_cache`	Activates the global file cache (§4.1.16)
`egr.eai.basket.group_calculation`	Enables group calculation.
`egr.eai.basket.move_items`	Enables functions for structural changes of the basket item tree (indent, unindent, move up, move down). (§5.6.3.23)
`egr.eai.server.pricing`	Enables basket calculation.
`egr.eai.server.ofml.package_groups`	Enables package groups.
`egr.eai.server.statistics_events`	Enable delivery of statistics events.

3.4 Running the License Server

The directory `EAILS` contains the shell script `elicsrv` that can be used to start, stop and restart the license server.

3.4.1 Starting the License Server

The license server can be started with the `start` action of the `elicsrv` script:

```
$ elicsrv start
```

Before the license server registers itself with the port mapper it checks whether another license server is already running. If so, it writes a message to the log file and terminates[4].

3.4.2 Stopping the License Server

The license server can be stopped with the `stop` action of the `elicsrv` script:

```
$ elicsrv stop
```

While the script terminates immediately the license server will take about five seconds before it terminates.

Note: *The license server must not be stopped while a local Online Configurator is still running as otherwise the Online Configurator would not be able to lease new license features, effectively preventing new sessions from being opened. Furthermore, a future version of the Online Configurator may choose to stop servicing requests if the license server is no longer accessible or has been replaced by a new instance of the license server.*

3.4.3 Restarting the License Server

The license server can be restarted with the `restart` action of the `elicsrv` script:

```
$ elicsrv restart
```

The `restart` action should be used instead of the `stop` action followed by the `start` action as it ensures that the old license server has terminated before the new one is started[5].

4 Repeated starts of the license server should be avoided if a license server is already running as this would cause the log file of the running license server to be deleted while it is still running.

5 Right now this is done by simply waiting for ten seconds after the old license server has been told to terminate itself and restarting the new one.

4 Configuration

4.1 Server Start-Up File

The server start-up file contains settings that affect the Online Configurator as a whole, as opposed to session specific settings that are placed in one or more session start-up files.

Server start-up files are always stored in the `$EAIWS/etc/startup` directory. By default, the name of the server start-up file is `server.cfg`. It can be changed using a command line option (§4.3.2.1).

Empty lines, lines consisting entirely of white space, and lines starting with a number sign (`#`) after a possibly empty sequence of white space are ignored.

The options available for use in the server start-up file are described in the following sections.

4.1.1 Default Locale

Synopsis: `app.framework.locale=`*`<locale>`*

Default Value: not set

The value of this option specifies the name of the default locale used for new sessions. It may be overridden by a locale specification (using the same key) in a session start-up file, or by the `-locale` argument of the `openSession` operation (§5.4.3.2). The format of valid locale names is described in section 5.4.3.10.

In addition to the effects of a sessions current locale on the behavior of the session as described in section 5.4.3.10, the default locale of a session also determines the currency used by the basket service at the beginning of a new session.

4.1.2 Fixer Compatible web-service

Synopsis: `egr.eai.basket.currency.fixer.api_base=`*`<url>`*

Register other Fixer-compatible web service implementations. For instance, to use the open source Foreign exchange rates API that provides exchange rates published by the European Central Bank, the server start-up file should contain one or both of the following entries:

```
egr.eai.basket.currency.fixer.api_base=http://api.exchangeratesapi.io/

egr.eai.basket.currency.fixer.api_base=https://api.exchangeratesapi.io/
```

See the following chapter for more details.

4.1.3 Known Currencies

Synopsis: `app.basket.currencies=`*`<currencies>`*

Default Value: `currencies`

The default value of this option is the name of a file describing all currencies known by the Online Configurator, including their localized names and exchange rates. The currency configuration file must be accessible as `$EAIWS/etc/`*`<currencies>`*`.cfg`.

The value of this property must be an URI-reference. If this reference is relative (i.e. does not contain a scheme) and consists of a single path segment that does not contain a period then .cfg is appended to the reference. Relative references are then resolved against the URI representing the etc/ directory.

The URI is then matched against the pattern returned by method getSourceURIPattern() of currency provider factories to select the factory used to produce a currency provider.

To use a Fixer-compatible currency provider (see https://fixer.io), the application start-up file property `app.basket.currencies` must be set to the request URL used to access the Fixer-compatible web service. The following restrictions apply:

- The path of the request URL must end with either `/latest` or with a slash followed by an ISOdate.
- The query component may contain only the following parameters:

  `access_key`

  the access key; required with data.fixer.io, but not necessarily required by alternative implementations of the service

  `base`

  the base/reference currency; The default value is EUR. Note that this is also the only base currency supported by the free pricing plan of the fixer.io service.

  `symbols`

  a comma-separated list of currency codes used to restrict the set of conversion rates provided by the web service; Without this parameter the fixer.io service returns about 170 conversion rates, and the exchangeratesapi.io service (see below) returns about 30 conversion rates.

Example: `http://data.fixer.io/api/latest?access_key=<access-key>`

If the request path ends with `/latest` then the provider’s conversion rates are regularly updated a quarter past every full hour. This results in about 700 to 750 updates per month which is within the limit of the maximum number of monthly requests allowed by the free pricing plan of the fixer.io service as long as the access key is used by only one instance of EAIWS.

Responses from successful requests are cached in `var/ebasket/currency/fixer/cache` using a digest computed from the request URL as key (file name). The cache is used when the provider is first accessed after an EAIWS (re)start. If a matching entry is found then the behavior is as follows:

- If the request path ends with an ISO date then the cached data is used.
- If the request path ends with /latest then the connect and read timeouts for the request are both reduced to 5 seconds (instead of the default 30 seconds) and the cached data is used if the request does not produce a valid result (due to timeout or some other error).

To use this provider with other Fixer-compatible web service implementations they must be registered using application start-up property `egr.eai.basket.currency.fixer.api_base`. For instance, to use the open source Foreign exchange rates API that provides exchange rates published by the European Central Bank, the server start-up file should contain one or both of the following entries:

```
egr.eai.basket.currency.fixer.api_base=http://api.exchangeratesapi.io/
```

```
egr.eai.basket.currency.fixer.api_base=https://api.exchangeratesapi.io/
```

Then, the application or session start-up file property `app.basket.currencies` can be set to something like

```
https://api.exchangeratesapi.io/2019-03-18?base=USD&symbols=EUR,USD,RUB
```

The example references a currency provider that uses USD as base currency and provides the conversion rates for EUR, USD and RUB as published by the EZB for Mach 18, 2019.

4.1.4 Operations per Minute

Synopsis: `egr.eai.server.operations_per_minute=`*`<ops-per-min>`*

Default Value: `unlimited`

The value of this option must be either a positive integer or the string `unlimited`[6].

6 Prior to version 2.0 RC 2, there was no default for this option (i.e. it always had to be specified in the start-up file), and the special

For the purpose of licensing, the number of non-trivial state-changing operations per minute the Online Configurator is able to serve is limited. The number of operations per minute specified with this option may further be reduced during start-up due to insufficient availability of throughput license features (§3.3.3).

Informational: The current algorithm used to limit the number of operations per minute works as follows (although the implementation is quite different): Assuming *n* is the number of operations per minute, there is a blocking queue of tickets with a maximum size of *n*. One thread inserts a ticket at the end of the queue every *60 / n* seconds, blocking if the queue is full. Threads servicing operations take a ticket from the front of the queue before they start one of the operations in question, blocking if necessary until a ticket becomes available. If more than one thread is waiting for a ticket to become available, then the longest waiting thread gets the next ticket that becomes available.

4.1.5 Maximum Number of Concurrent Sessions

Synopsis: `egr.eai.server.max_sessions=`*`<max-sessions>`*

Default Value: `100`

The value of this option specifies the maximum number of concurrent sessions supported by one instance of the Online Configurator.

Note: The actual maximum for the number of concurrent sessions may further be limited by the number of available session license features (§3.3.2) and vary over time if more than one instance of the Online Configurator is running on the same host, and if the sum of the configured maximum number of concurrent sessions over all instances of the Online Configurator is greater than the number of available session license features.

4.1.6 Timeout for Sessions

Synopsis: `egr.eai.server.session_timeout=`*`<timeout-in-seconds>`*

Default Value: `300`

The value of this option specifies the number of seconds of inactivity the Online Configurator waits until it automatically closes the session and eventually deletes session specific data from disk (see also $4.1.7).

In addition to the timeout in seconds, the properties can be set to a duration similar to ISO 8601 (see https://en.wikipedia.org/wiki/ISO_8601#Durations), with the difference that 'Y' (year), 'M' (month) and 'W' (week) are not supported, nor are the basic and extended formats PYYYYMMDDThhmmss and P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].

4.1.7 Session Suspend

Synopsis: `egr.eai.server.session_suspend_timeout=`*`<timeout-in-seconds>`*

The value of this option specifies the number of seconds of inactivity the Online Configurator waits until it automatically moves the session specific data from RAM to disk. For session suspend to work, this property must be set to a value less than the session timeout.

In addition to the timeout in seconds, the properties can be set to a duration similar to ISO 8601 (see https://en.wikipedia.org/wiki/ISO_8601#Durations), with the difference that 'Y' (year), 'M' (month) and 'W' (week) are not supported, nor are the basic and extended formats PYYYYMMDDThhmmss and P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].

4.1.8 Allowed File Access

Synopsis: `egr.eai.server.file_access=<path-list>`

Default Value: empty string

value `unlimited` was not supported.

The value of this option must be a semicolon-separated[7] list of absolute path names in the Online Configurator's local file system. The Online Configurator does not check whether the paths point to existing directories or files.

The client may pass a file URI as the `uri` argument of the `loadSession` operation, or as the `uri` option of the `saveSession` operation. For the Online Configurator to be able to load the session from this file, or save it to this file, the file's path or one of the files parent directories must be listed by this option.

4.1.9 OFML Debug

Synopsis: `egr.eai.gf.cobra.enable_file_io=<true|false>`

Default Value: false

If set to `true`, the `-enableFileIO` flag is passed to fConfigure(), allowing Cobra to do file IO operations (necessary to write the OFML debug log into a file).

4.1.10 Rotation of Log files

If rotation of log files is not enabled, a log file named `app_YYYY-mm-dd_HHhMMmSS.LLL.log` is created when the server starts and written to until the server terminates.

If rotation of log files is enabled, log messages are written to <appkey>.log until the log file is rotated. Rotation of a log file renames it into <appkey>.log-YYYYmmdd, optionally followed by a minus sign and a serial number and, if compressed, by suffix '.gz'. The serial number is used if more than one rotation happens per day.

The logging subsystems checks whether or not to rotate whenever (and prior to) a log message is written unless the last check was less than five minutes ago. If the check determines that the log file should be rotated, the current log file is closed, renamed, and optionally compressed, a new log file is created, and old log files are deleted if the current number of rotated log files exceeds the configured number.

The following application properties can be used to control rotation of log files:

Synopsis: `egr.eai.logrotate.ngen=<integer>`

Default Value: -1

Controls the maximum number of rotated log files to keep until they are automatically deleted. Allowed values are within -1 and 2^31-1, both inclusive. If set to -1, rotation of log files is disabled. Otherwise it is enabled.

Synopsis: `egr.eai.logrotate.min_size=<integer>`

Default Value: 0

Log files smaller than the size (in kilo bytes) specified by this option are never rotated.Allowed values are within 0 and 2^63-1, both inclusive.

Synopsis: `egr.eai.logrotate.max_size=<integer>`

Default Value: `0x7fffffffffffffff`

Log files greater than the size (in kilo bytes) specified by this option are always rotated (as soon as the check is done as described above), no matter what the value of property 'egr.eai.logrotate.when'. Allowed values are within the value specified as the minimum size and 2^63-1, both inclusive.

Synopsis: `egr.eai.logrotate.when=<string:>`

May be used to specify the minimum time between log file rotations and a time when rotation shall take place. The syntax, in EBNF, is as follows:

```
when ::= interval? timespec?
```

[7] On Unix-like operating systems, the colon may be used too.

```
interval ::= digit+

timespec ::= ( '@' date-time? ) | ( '$' period? )

date-time ::= ( ( year? month )? day )?
              'T'
              ( hour ( minute ? second? )? )?

period ::= ( 'D' hour-of-day )
         | ( 'W' day-of-week ( 'D' hour )? )
         | ( 'M' day-of-month ( 'D' hour )? )

year ::= digit digit digit digit    // 0001..9999
month ::= digit digit               // 01..12
day ::= digit digit                 // 01..31
hour ::= digit digit                // 00..23
minute ::= digit digit              // 00..59
second ::= digit digit              // 00..59

hour-of-day ::= digit? digit        // 0..23
day-of-week ::= digit               // 0..7, with both 0 and
                                    // 7 representing Sunday
day-of-month ::= (digit? digit)     // 1..31
               | [Ll]               // 'L' and 'l' indicate
                                    // last day of month

digit ::= [0-9]
```

If the value of this property is an empty string (the default), the the decision whether or not to rotate the log file is solely based on the maximum size (thus, with both 'max_size' and 'when' are set to their default values, no rotation takes place).

Otherwise, the time the log file should be rotated next is determined as follows:

1. The time of next rotation is set to the time of the last rotation (the modification time of the most recently rotated log file, or the time the application has been started if no rotated log file exists).
2. If an interval has been specified, the interval is treated as a number of hours and added to the next rotation time.
3. If a rotation time has been specified, the time of next rotation is adjusted to the smallest possible value greater than the current value that matches specified time. Unspecified year, month, day(-of-month) and day-of-weak do not cause an adjustment, whereas unspecified hour, minute and second set the corresponding field to zero.

Finally, if the time of next rotation is less than or equal to the current time, the log file is rotated.

Examples:

`10`
rotate every ten hours

`@01T01`
rotate the first day of each month at 01:00 AM

`47$D23`
rotate every 47 hours, but wait until 11:00 PM

`$W5D23`
rotate every Friday at 11:00 PM

`$M1`
rotate the first day of each month at midnight

`$MLD23`
rotate the last day of each month at 11:00 PM

4.1.11 HTTP Server Options

4.1.11.1 Port Number

Synopsis: `egr.eai.server.http.port=`*`<port-number>`*

Default Value: `8080`

The port number to use instead of 8080. May be overwritten by the port specified as part of egr.eai.server.ht-tp.listen.

4.1.11.2 HTTP Server

Synopsis: `egr.eai.server.http.listen=<host>[':'<port>]`

Create an HTTP server for each address specified by <host>. <host> must be an IPv4 address, an IPv6 address enclosed in square brackets ('[' and ']'), or a host name.

If <host> is a host name, the name is resolved to determine all addresses of the host. If resolution of the host name fails, EAIWS terminates with an error. Otherwise, an HTTP server is created for each address produced by resolution.

This option may be specified multiple times to listen on multiple internet addresses. Use of this option prevents the creation of an HTTP server listening on the wildcard address.

4.1.11.3 HTTP Request Executer

The initialization of the HTTP request executor can now be configured through the following application properties:

Synopsis: `egr.eai.server.http.thread_pool.max_pool_size`

The maximum number of threads in the pool. The value must be within -1 and 1024 (both inclusive). The default value is -1.

If the value is -1, the actual maximum pool size is equal to the number of processor available to the Java runtime at startup, multiplied by two. Note that in case of Intel Hyper-Threading, this is the number of visible CPUs, not the number of CPU cores (i.e., in the most common case, 8 instead of 4).

If the value is zero, no thread pool is used. Instead, all requests are serviced by the HTTP server thread. This may be useful for special cases where it is known that the server services requests for a single single-threaded client.

Synopsis: `egr.eai.server.http.thread_pool.keep_alive`

The number of seconds to keep an idle thread alive. The value must be withn -1 and 2^31-1 (both inclusive). The default value is 5.

If the value is -1, threads are kept alive infinitely. In general, a value of zero should be avoided as it would result in continual thread replacement.

This property is ignored if no thread pool is used (i.e. if the maximum pool size is zero).

Synopsis: `egr.eai.server.http.thread_pool.queue_size`

The upper bound on the size of the queue used to store pending requests. The value must be within -1 and 65536 (both inclusive). The default value is 1024.

A value if -1 results in a queue with unlimited length. Otherwise, when the HTTP server adds a request to the queue, and the queue size has reached its upper bound, the HTTP server waits until a request has been removed from the queue by one of the threads servicing requests.

The special value zero is allowed and results in the use of a queue where each add operation of the HTTP server waits for a corresponding remove operation of one of the request service threads.

This property is ignored if no thread pool is used (i.e. if the maximum pool size is zero).

4.1.12 Automatic Generation of Article Images

Synopsis: `egr.eai.server.export.image=<true|false>`

Default Value: `true`

This option controls whether the automatic generation of article images is supported, provided the necessary license feature is available (§3.3.6). If the value of this option is `false`, or if the license feature is not available, the operation `getGeneratedImage` of the `BasketService` always returns an empty string.

4.1.13 3DS Geometry Export

Synopsis: `egr.eai.server.export._3ds=<true|false>`

Default Value: `true`

This option controls whether the export of 3DS geometries for article positions is supported, provided the necessary license feature is available (§3.3.6). If the value of this option is false, or if the license feature is not available, the operation `getExportedGeometry` of the `BasketService` always returns an empty string when called with `format=3DS`.

4.1.14 OFML File Extensions

Synopsis: `egr.eai.http.ofml_extensions=<list-of-extensions>`

Default Value: `.bmp,.jpg,.jpeg,.rgb,.tga,.3ds,.dwg,.dxf,.geo,.pec,.svg,.html,.pdf`

The set of OFML data files served by the HTTP server has been limited to files whose name ends with an extension from a configurable set of extensions. The HTTP server ignores case when it compares the extension of the path name of a GET request with the extensions from this set[8].

The default set of extensions consists of the extensions listed above as the default value for this option.

The set of extensions can be overwritten or augmented in the server start-up file with one or more uses of this option. They are processed in the same order they appear in the server start-up file. The value for each of these options must start with an optional PLUS SIGN (U+002B) followed by a possibly empty list of extensions. Extensions must consist of a PERIOD (U+002E) followed by one or more alphanumeric ASCII characters. Multiple extensions must be separated by COMMA (U+002C). ASCII white space in front of the PLUS SIGN (if present) and around extensions is ignored. If the value starts with a PLUS SIGN, then the currently configured list of extensions is augmented with the list of extensions contained in this entry. Otherwise the list of extensions in the current entry replaces the currently configured set of extensions.

4.1.15 HTTP Server Root

Synopsis: `egr.eai.http.server_root=<context>:<path>`

Default Value: not applicable

Besides its ability to handle SOAP calls and serve URLs returned as part of the response message of SOAP calls (like catalog images, generated article images, and exported geometries) the integrated HTTP server also provides a very simple general purpose HTTP server whose functionality is limited to serving GET requests for plain files.

The value of this option consists of a context and an absolute path, separated by a colon character. The `<context>` is matched against the beginning of the URL paths from incoming GET requests, and the `<path>` is interpreted as the absolute path of a directory within the local file system. Together they define a server root. The option can be used multiple times to define multiple server roots.

The `<context>` must begin and end with a slash character. The `<path>` must be absolute and should use the platforms default file name separator. If it does not end with a file name separator, a file name separator

[8] The comparison has been case sensitive in EAIWS versions prior to 3.0 RC 1.

is appended.

When a GET request comes in, the HTTP server decodes escaped octets within the path of the request URL[9] and replaces sequences of slash characters with a single slash. It then selects the server root whose context is the longest matching[10] prefix of the request path. If no such server root is found, the server sends a 404 (Not Found) response message. Otherwise, it removes the context from the request path to form a relative request path.

The server then substitutes slashes within the relative request path with the platform's default file name separator. If the resulting path contains two consecutive dots as an element, then the request is rejected with an 403 (Forbidden) response message. Otherwise, the path is appended to the absolute path specified for the server root to form the absolute path of the file to retrieve. If the resulting path does not represent a file, or cannot be opened for reading, the server sends a 404 (Not Found) response. Otherwise it sends a 200 (OK) response containing the file.

4.1.16 Global File Cache

The primary purpose of the global file cache is the reduction of the average time spent in the `getGeneratedImage` operation (§5.6.3.40) of the basket service. To do so, this operation computes a hash over the 3D geometry of an OFML object and the rendering and camera settings as well as all other settings passed as options to the image generation operation. The hash is converted to a file name. If the cache contains a file with this name, the file is expected to contain the requested image[11]. If not, the image is generated and stored in the cache using the computed file name.

This algorithm has some advantages and drawbacks. It is quite simple and efficient, having nearly constant complexity, even for very large caches. On the other hand, all cached images become useless whenever there is a change to the aforementioned hash function[12], while modifications to the rendering algorithm do not result in old images being invalidated[13].

By default, the cache directory is `var/cache` within the Online Configurator's installation directory. If the cache is to be stored somewhere else, `var/cache` may be replaced by a symbolic link to either an empty directory or a complete copy of an existing cache.

It is not allowed to share a cache between multiple instances of the Online Configurator.

The cache must not be modified by a program other than the Online Configurator. Individual files must not be deleted, nor copied into the cache (either from another cache or from a backup). The only allowed operations are deleting the entire cache (all files and sub-directories within `var/cache`, and optionally `var/cache` itself), and copying or restoration of the entire cache.

4.1.16.1 Maximum Cache Size

Synopsis: `egr.eai.fscache.size=`*`<size-in-KiB>`*

Default Value: unlimited

This option sets the target cache size. The actual amount of disk space used may be larger because:

- The cache cleaner does not delete objects that are in use. Therefore, if the target size of the cache is quite small, the total size of all used objects may be greater than the target size.
- The cache performs its size computations in KiB (number of bytes divided by 1024 and rounded up to the next integral value). If the block size of the file system is greater than one KiB, files may actually use more disk space than assumed by the cache.
- The size of file system meta data (i-nodes, directories) is not considered by the cache at all.

9 Escaped octets that do not represent an ASCII character are expected to be part of a sequence of escaped octets that forms the UTF-8 encoding of an Unicode character. If this is not the case, the behavior is undefined.
10 The comparison of the context with the beginning of the request path is case sensitive.
11 In theory, the hash function (MD5) could compute the same hash for different article configurations and options. In practice, however, this is expected not to happen.
12 While this reduces the efficiency of the cache, other than that it should not be a problem as these image will be removed from the cache once it has reached it's maximum size.
13 Thus, whenever a new version is installed which is known to have significant improvements to the image rendering algorithm, the entire cache should be deleted by hand.

The default value `unlimited` is used if the option is not specified, or if the option value is an empty string. It must not be specified literally as the option value.

If the cache size is not limited, the Online Configurator does not remove unused images as the cache fills up. If disc space is not an issue, the cache size may be left unlimited without significant impact on the cache's performance[14].

4.1.16.2 Use of Extended Attributes

Synopsis: `egr.eai.fscache.use_extended_attributes=<`*`true|false`*`>`

Default Value: `false`

By default, the data stream of cache files contains both metadata about the files content (like image width and height) as well as the actual file data. Therefore, the cache files can not be used as ordinary image files.

If the file system used to store the cache supports extended attributes, this option can be set to `true` to store the metadata as an extended attribute. This is a prerequisite for optimizations enabled with the `egr.eai.fscache.transfer_modes` option.

If this option is set to `true`, the Online Configurator ensures that the file system used to store the cache actually supports extended (user) attributes. If it does not, the Online Configurator terminates with an error message.

For more information about Linux file systems and their support for extended attributes, see `attr(5)`, section FILESYSTEM DIFFERENCES.

4.1.16.3 Transfer Modes

Synopsis: `egr.eai.fscache.transfer_modes=<`*`list-of-transfer-modes`*`>`

Default Value: `copy`

The value of this option must be a possibly empty, comma-separated list of `copy`, `symlink`, `hardlink` and `link`, with `link` being a shortcut for both `symlink` and `hardlink`.

The option lists the transfer modes considered for use by the Online Configurator whenever a file must be transferred between the cache and a sessions working directory.

The `copy` mode is always considered, whether or not it is listed by this option.

If the use of extended attributes is disabled (see §4.1.16.2), the link transfer modes are not considered, even if they are listed by this option.

From the list of configured transfer modes, the cache selects the transfer mode it deems best for the operation in question. It does not, however, consider the capabilities of the underlying operating system or file system. In particular, if the file cache and the session cache are stored on different file systems, the `hardlink` mode must not be used.

From a performance point-of-view, `hardlink` is the most efficient, shortly followed by `symlink`. The `copy` mode is the least efficient, as in addition to the metadata operations necessary to write or update the directory and i-node, it also requires the complete file data to be read and written, which may be quite resource consuming, especially for large files.

4.1.17 Java System Properties

Synopsis: `system:<`*`property`*`>=<`*`value`*`>`

Default Value: not applicable

Lines starting with `system:` can be used to set the value of Java system properties. Note that this is similar,

[14] Assuming a cache size of one terabyte, and an average file size of 50 kilobyte, the leaf directories storing the cached files contain, on average, somewhat more than 300 files. Even for file systems with linear directory search this is still a reasonable size. File systems that use tree structures for their directories (XFS, EXT3, EXT4, among others) should be able to accommodate much larger caches.

but not identical, to using the `-D<property>=<value>` command line option of the Java Virtual Machine as with the command line option the property is set from the very beginning of the execution of the Online Configurator, whereas with a start-up file entry the property is set early during start-up. In particular, there is no point to set the `egr.eai.fw.cmdline.AppRootDir` property within the server start-up file, because the server start-up file will not be found unless the application root directory is known.

4.1.18 Version Number of License Feature

Synopsis: `egr.eai.fw.licensing.feature_versions=<list-of-feature-version>`

Default Value: not applicable

The version number used when requesting a license feature can be configured. The value is a list of feature version specifications, separated by backslash followed by newline. Each feature version specification consists of a feature pattern, a colon, and the version number. The feature pattern is a glob-style pattern (see fnmatch(3)), supporting ?, *, ** and [...] placeholders, and using dot instead of slash as delimiter (** behaves like *, but also matches delimiters).

4.1.19 Color Space

Synopsis: `egr.eai.gf.color_space=<LinearRGB|sRGB>`

Default Value: sRGB

Defines the color space the OpenGL rendering should use. Can also be defined in the session start-up file, whereas a specification in a session configuration file takes precedence of a specification in the server start-up file.

4.1.20 Stop-Word Filters

Synopsis: egr.eai.product.catalog.search.disable_stop_filter=`<list-of-languages>`

Default Value: not applicable

Disables stop-word filters by language.

The value of the property is a possibly empty comma-separated list of language tags (like 'de', 'de-DE', 'de-Latf'). The stop-word filter for a language with a particular tag is disabled if this language tag or one of its parent tags is listed in the value of this property. Thus, to disable stop-word filters altogether, the language tag 'und' may be used, representing the undetermined language.

While probably not of any practical importance, it should be noted that language tags are normalized. Thus, 'de-Latn' becomes just 'de', disabling the stop-word filter for all German languages, even if they use other scripts.

(The parent tag of a language tag is the language tag with the least significant subtag (i.e. the last subtag) removed, except in case the language tag consists of the primary language subtag only, where the parent tag is 'und', representing the undetermined language. The language tag for the undetermined language has no parent tag.)

4.1.21 GZIP-compressed Geometries

If a geometry file format has been selected for compression (based on its file name extension), the geometry export creates, in addition to the original geometry file, a second compressed file that has the same name as the original file, but with suffix `.gz` added.

The following two properties can be specified in the server start-up file to control compression of generated geometry files:

Synopsis: `egr.eai.server.export.gzip.level=<level>`

Default Value: `0`

`Level` must be an integer between zero and nine, both inclusive. Compression is disabled (no compressed file will be generated) if the specified level is zero. Otherwise, `level` controls the speed of compression, whereas 1 indicates the fastest and 9 the slowest compression method. Obviously, the slower the compression method, the better the compression ratio will (usually) be.

Synopsis: `egr.eai.server.export.gzip.extensions=<extension-list>`

Default Value: `empty`

`Extension-list` must start with an optional plus sign (+), followed by zero or more file extensions separated by comma (,), with each file extension consisting of a period (.) followed by one or more US-ASCII letters or digits. US-ASCII white space in front of the plus sign and around extensions is ignored.

The server start-up file may contain multiple occurrences of this property. If so, the second and subsequent ones should probably start with a plus sign as the plus sign causes subsequent extensions to be added to the current set of extensions (a non-empty extension list that does not start with a plus sign clears the current set of extensions).

4.1.22 Property References

Property references consist of a previously defined property name enclosed in `${` and `}`. The property name must consist of a non-empty sequence of alphanumeric US-ASCII characters, underline (_) and/or period (`.`).

Property values that do not contain `${` are left unmodified. Property values that do contain `${` are processed as follows:

- `${` must start a valid property reference or processing of the start-up file will fail. There must be a previous definition of the referenced property in the same start-up file. The whole property reference is replaced by the referenced properties current value.
- `$$` is substituted by a single `$`, which will not be considered as a leading dollar sign in `${` or `$$`.
- Any other dollar sign is left unchanged.

One possible use of property references is easy switching between multiple copies of mirrored user application (e.g. pCon.box) data sets. The relevant part of the start-up file may look as follows:

```
.data=/home/data/ofml
#.data=/home/data/ofml/.zfs/snapshot/2017-10-07

app.gf.data.profile.registration=${.data}/pCon.box/catalogs/default.profil
es
app.gf.data.profile.path=${.data}/pCon.box/catalogs/profiles
app.gf.data.path.substitution=,(^|[;:])/opt/pCon/_data/,$1${.data}/,
```

4.1.23 OFML catalog cache database

Synopsis: `egr.eai.product.catalog.oas.cachedb=<`*`ebase`*`|`*`ods`*`>`

Default Value: `ebase`

The application property `egr.eai.product.catalog.oas.cachedb` can be used to select the back-end of the OFML catalog cache database.

4.1.24 OFML data cache backend

Synopsis: `egr.eai.server.odc.backend=<h2mvs|bdb>`

Default Value: `bdb`

Select OFML Data Cache backend. Value `h2mvs` selects the H2 Multi Version Store, whereas value `bdb` selects Berkely DB Java Edition.

4.1.25 FAPI-shell Command Timeout

Synopsis:

`egr.eai.gf.min_cmd_timeout=<`*`time-in-seconds`*`>`

`egr.eai.gf.max_cmd_timeout=<time-in-seconds>`

Default Value: 0

Startup file (usually etc/startup/server.cfg) has two new optional entries `egr.eai.gf.min_cmd_timeout` and `egr.eai.gf.max_cmd_timeout`. Values must be non-negative decimal, hexadecimal (prefix 0x or 0X) or octal (prefix 0) integers less than 2^31 representing lower and upper timeout limits in seconds.

The default value is `zero`, meaning 'unspecified' (i.e. no minimum or maximum).

Invalid values result in use of the default value and in a log message whenever an FAPI-shell is started. Similarly, a minimum greater than the maximum (other than zero) results in a log message and the minimum set to the maximum.

4.1.26 Default image options (server start-up file)

Synopsis:

`egr.eai.server.export.default_image_options=<key/value list>`

Default Value: empty

Added support for session startup option
`egr.eai.server.export.default_image_options=<key/value list>` It can be specified, with decreasing priority, as part of the options returned by
`ISessionManagerListener.getStartupProperties()`, in the session startup file, or in the server startup file. A higher priority startup option value, even if empty, completely replaces a lower priority value.

The startup option value is decomposed into a possibly empty list of strings representing image export options as accepted by operation getGeneratedImage. Individual options are US-ASCII whitespace separated quoted strings or key/value pairs. In case of key/value pairs, key and value must be separated by a single equal sign (U+003D). Keys start with an optional dollar sign (U+0024) followed by a sequence of one or more identifiers separated by a single full stop / period (U+002E). Values are either a quoted string or a possibly empty sequence of non-whitespace characters. Key and value (decoded in case of a quoted string) are concatenated, separated by a single equal sign, before appended to the list of image export options.

Identifiers and quoted strings must adhere to ISO C89 rules.

The list of image export options is then validated the same way as done by operation getGeneratedImage.

If the startup option value does not conform to the rules described above, or the resulting list of image export options contains an invalid option, the openSession operation fails.

4.1.27 Support for response headers

Synopsis:

egr.eai.http.response_headers==<path pattern: HTTP response header field>

Default Value: empty

The option value consists of a `path` pattern and a HTTP response header field, separated by a path separator character (semicolon (U+003B) on Windows and semicolon or colon (U+003A) on Unix-like systems).

The `path` consists of a non-empty prefix and a possibly empty pattern (even though an empty pattern doesn't make much sense as of now as URLs ending with a slash produce a 404 Not Found response).

The path prefix consists of the longest sequence of characters other than `asterisk` (U+002A) starting and ending with a `slash`. The path prefix must literally match the beginning of an URL path.

The pattern consists of a sequence of characters other than `asterisk` (which must literally match a corresponding characters in the URL path), interspersed by `zero` or more sequences of one or more `asterisk`. A single `asterisk` matches a sequence of `zero` or more characters other than `slash`, whereas a sequence of two or more `asterisks` matches a possibly `empty` sequence of any character.

The `response` header field consists of field `name` and file `value` according to RFC 9112, Section 5, Field Syntax.

The field `name` must match one of the supported field names (`ignoring case)`, and the field `value` must conform to the rules for that field `name`.

The only field `name` currently supported is `Cache-Control`. The rules for the `field` value are defined in RFC 9111, Section 5.2, Cache Control, Subsection 5.2.2, Response Directives. In case of different rules for sender and receiver, the rules for senders apply.

The startup file may contain multiple `egr.eai.http.response_headers` option lines. The effective line is determined as follows:

Only lines whose `path` prefix and `pattern` match the normalized `path` component of the request URL are considered.

Lines with a `longer` matching `path` prefix take precedence over lines with a `shorter` matching `path` prefix.

If there are `multiple` matching lines with equal `path` prefix but different `patterns`, the line that appears last in the startup file is used.

Example:

egr.eai.http.response_headers=/**:Cache-Control: public, max-age=3600
egr.eai.http.response_headers=/session-cache/**:Cache-Control: public, max-age=86400
egr.eai.http.response_headers=/OFML/**:Cache-Control: public, max-age=2592000

Examples for paths:

/prefix/*.foo
All files ending with `.foo` in directory /prefix/ (but not in subdirectories of /prefix/)

/prefix/**.foo
All files ending with `.foo` in directory /prefix/ and subdirectories of /prefix/ at any `depth`.

/prefix/**/*.foo
All files ending with `.foo` in subdirectories of /prefix/ at any `depth`.

/prefix/*/*.foo
All files ending with `.foo` in immediate subdirectories of `prefix`.

4.1.28 Statistics Event Manager

4.1.28.1 Block Size and Queue Size

A local event queue is basically a large ring buffer consisting of one or more dynamically allocated blocks. Posted events are appended to the buffer (serialized as CBOR), whilst the consumer reads events from the front of the buffer.

The maximum size of the ring buffer (the queue size) is limited. If serialization of a group of events results in a buffer overflow, all events in the group are silently discarded. The consumer can use `IStatsEventQueue.getDiscardedEventCount()` to obtain the number of discarded events.

Two application properties can be used to configure block and buffer/queue size:

- `egr.eai.fw.statistics.local_queue.block_size` -- the block size; Valid values are between 64 and 524256 (slightly less than 512 KB), both inclusive. The default value is 4096.
- `egr.eai.fw.statistics.local_queue.size` -- the queue size; Valid values are between 1024 and 2147483647 (0x7fffffff), both inclusive.

The queue size must be at least as large as the block size, and is rounded up to a multiple of the block size.

The ring buffer consists of at most n + 1 blocks, where n is the queue size divided by the block size. The extra block is necessary to prevent the effective queue size from becoming smaller than the configured queue size (the effective queue size is the total queue size minus the number of bytes already read from the first block).

4.1.28.2 HTTP/HTTPS server

In addition to local event queues, the event manager allows events to be forwarded to an HTTP/HTTPS server using POST requests with content type `application/json`. The content of each request consists of a single JSON array, with each element representing an event. Each event consists of a two-element array. The first element is an UUID identifying the event type. The second element is a JSON object containing the actual event data (see §6).

Configuration of the HTTP (client) event queue is primarily done using the following application properties:

- `egr.eai.fw.statistics.http_client.url` -- The http or https URL of the server; There is no default value. If the property is not defined, the HTTP event queue is not active.
- `egr.eai.fw.statistics.http_client.certificate_validation` -- In case of an HTTPS server using a self-signed certificate, the property may be set to `false` to suppress certificate validation on the client side. The default value is `true`.

If an HTTP (client) event queue has been configured, posted events are serialized into a message object. The message object will eventually be handed over to a separate upload thread using a queue of message objects. The upload thread sends the message to the server, using it as the content (entity body) of a POST request.

The message currently being constructed from newly posted events is completed and handed over to the upload thread if the age of the first event in the message, or the total size of the message after serialization of an event (or a group of events that have been posted together), exceeds the following configurable values:

- `egr.eai.fw.statistics.http_client.max_event_delay` -- the maximum time an event spends in a partially constructed message before the message is completed and handed over to the upload thread; The value must be specified as a number of seconds or as an ISO-8601 duration (limited to D, H, M, and S components). The default value is one hour.

- `egr.eai.fw.statistics.http_client.target_message_size` -- The maximum message size limit in bytes. The default value is `0x10000` (64 KB). Note that this is not a hard limit (see above). The maximum allowed value is 1 GByte.

  If this property is set to zero, each message consists of a single event (or of a group of events that have been posted together), and there is no delay between the serialization of events to the message and the handing over of the message to the upload thread.

The upload thread takes messages from the message queue fed by the event manager. The message queue may contain multiple messages if they are generated at a faster rate than they can be uploaded to the server. To avoid excessive growth of the message queues, the maximum size of the message queue can be configured with

- `egr.eai.fw.statistics.http_client.max_queue_size` -- the maximum size of the message queue in bytes. The default value is `0x100000` (1 MB).

  If the event manager completes a message and attempts to feed it to the message queue, but doing so would result in the total size of all messages in the queue to exceed this value, the message is rejected by the queue. The event manager writes an INFO log message and discards the message.

Once the upload thread has taken a message from the message queue, it compares the age of the message with the value of property

- `egr.eai.fw.statistics.http_client.max_message_age` -- the maximum age of a message before it is discarded by the upload thread. The value must be specified as a number of seconds or as an ISO-8601 duration (limited to D, H, M, and S components). The default value is two hours.

  The upload thread writes an INFO log message and discards a message taken from the message queue if the age of the message (the time elapsed since the posting of the first event in the message) exceeds the value configured by this property.

If the message is not expired, the upload thread establishes a connection to the server and constructs a request with POST as its request method. The request property Content-Type is set to `application/json`. The following application property can be used to define additional request properties:

- `egr.eai.fw.statistics.http_client.extra_request_properties` -- one additional request property to be sent to the server. The property value must conform to the production field-line of RFC 9112.

  This application property may be used repeatedly to specify multiple request properties.

The request is sent to the server, using the message as the request's content. The server should respond with a 200 (OK) status code or, preferably, with a 204 (No Content) status code (as any content sent as part of the response is discarded anyway).

If there is an IO error sending the request or receiving the response, or if the response contains a status code which does not indicate success, the upload thread either discards the message or puts it back at the head of the message queue. In general, the message is discarded unless

- it can be said with absolute certainty that the server hasn't received the complete message, or
- the server responded with status code 404 (Not Found), 408 (Request Timeout), or 503 (Service Unavailable).

In either case (discard or put back), the upload thread pauses further processing for a configurable amount of time:

- `egr.eai.fw.statistics.http_client.error_retry_delay` -- the time to wait after a failed upload attempt encoded as a number of seconds or an ISO-8601 duration (limited to D, H, M, and S components); The default value is ten minutes.

While (mostly) sleeping for the configured amount time, the upload thread still removes expired messages (messages whose age is greater than the maximum message age) from the message queue so the event manager doesn't have to discard messages it wants to hand over to the upload thread just because the message queue has filled up with expired messages.

- `egr.eai.fw.statistics.http_client.encoding` -- the method used by the HTTP client event queue to encode statistics event sent to the server; possible values are:
  - `json` -- data is sent as a single JSON array, with each element representing one event (the default)
  - `json-lines` -- data is sent using JSON Lines encoding, with each line containing one event
  - `cbor` -- data is sent as a single CBOR array, with each element representing one event
  - `cbor-records` -- data is sent using a proprietary binary record format, with each record containing one CBOR-encoded event

Depending on the encoding, the Content-Type of the POST request is set to either `application/json`, `application/jsonl`, `application/cbor`, or `application/x.egr-cbor-records`.

Use of JSON lines and CBOR records may be necessary (depending on the implementation) for the server to recover from decoding errors, as may happen (again depending on the implementation) in case of unknown event types.

Use of CBOR instead of JSON reduces the size of serialized data by about factor two. Most notably, this affects the amount of data which can be buffered by the client (EAIWS) in case of temporary unavailability of the server (assuming equal maximum queue size). It also reduces the frequency of POST requests issued by the client and/or content size of request messages (assuming equal maximum message size and event delay).

Compression allows more events to be buffered on the client (EAIWS) side in case of temporary server failure. If compression is enabled, the content of HTTP POST messages used to transfer events to the server is also compressed.

- `egr.eai.fw.statistics.http_client.compression` -- specifies the compression method:
  - `none` -- no compression (the default)
  - `deflate` -- "zlib" data format (RFC1950) containing a "deflate" compressed data stream (RFC1951) that uses a combination of the Lempel-Ziv (LZ77) compression algorithm and Huffman coding; The Content-Encoding header field of the POST request is set to deflate.
  - `gzip` -- an LZ77 coding with a 32-bit CRC that is commonly produced by the gzip file compression program (RFC1952). The Content-Encoding header field of the POST request is set to gzip.
- `egr.eai.fw.statistics.http_client.compression_level` -- the compression level used in case compression is enabled; The compression level must be a value between 1 and 9, both inclusive. The default value is 6.

  The configured compression level is ignored if compression is not enabled.

Given the difficulties to determine the final compressed message size while compression is still in progress (a considerable amount of the data about to be compressed is still buffered in the compressor), compressed messages use a different algorithm to compute the message size which compared against the maximum message size (which actually behaves as a target message size) as configured with application property `egr.eai.fw.statistics.http_client.max_message_size`. In case of compressed messages, the current estimated message size is computed as the larger of the number of bytes of compressed data produced so far, and the number of bytes of uncompressed data divided by the weighted average of recent compression rates.

Enabling of compression has no immediate effect on the performance of operations emitting statistics events because serialization of these events is done by a separate thread.

4.2 Session Start-Up File Options

The session start-up file contains options that can vary between sessions.

Session start-up files are always stored in the `$EAIWS/etc/startup` directory. By default, the name of the session start-up file is `session.cfg`. It can be changed using an option passed to the `openSession` operation of the `SessionService` (§5.4.3.2).

If a session start-up file is added to a running Online Configurator, then it is evaluated the next time an `openSession` operation referencing this start-up file is executed.

If a session start-up file is modified while the Online Configurator is running, existing sessions will continue to use the configuration specified by the old version of the start-up file, but otherwise the Online Configurator behaves as if the start-up file has been newly added.

4.2.1 Allowed File Access

Synopsis: `egr.eai.server.file_access=<path-list>`

Default Value: not set

This option overrides the option of the same name from the server start-up file. If this option is set, the option in the server start-up file is ignored by this session. If it is not set, the value of the server start-up file's option is used. For more information, see 4.1.8)

4.2.2 Unit Format Settings

Synopsis:

```
egr.eai.gf.UnitFormatter.LengthUnit=<length-unit>
egr.eai.gf.UnitFormatter.LengthRepresentation=<unit-repr>
egr.eai.gf.UnitFormatter.LengthPrecision=<integer>
egr.eai.gf.UnitFormatter.LengthGrouping=<boolean>
egr.eai.gf.UnitFormatter.LengthUnitDisplayed=<boolean>
egr.eai.gf.UnitFormatter.LengthUpperCase=<boolean>
egr.eai.gf.UnitFormatter.AngleUnit=<angle-unit>
egr.eai.gf.UnitFormatter.AnglePrecision=<integer>
egr.eai.gf.UnitFormatter.AngleZeroDirection=<floating-point>
egr.eai.gf.UnitFormatter.MaxAngle=<floating-point>
egr.eai.gf.UnitFormatter.AngleUnitDisplayed=<boolean>
egr.eai.gf.UnitFormatter.AngleMathPositive=<boolean>
egr.eai.gf.UnitFormatter.AreaUnit=<area-unit>
egr.eai.gf.UnitFormatter.AreaRepresentation=<unit-repr>
egr.eai.gf.UnitFormatter.AreaPrecision=<integer>
egr.eai.gf.UnitFormatter.AreaGrouping=<boolean>
egr.eai.gf.UnitFormatter.AreaUnitDisplayed=<boolean>
egr.eai.gf.UnitFormatter.AreaUpperCase=<boolean>

<length-unit>:    mm|cm|dm|m|in|ft|in_dq|ft_sq|architectural|yd
<angle-unit>:     degrees|radians
<area-unit>:      mm2|cm2|dm2|m2|sq_in|sq_ft|sq_yd
<unit-repr>:      scientific|fixed|auto|fraction
```

'in_dq' stands for inch represented with double quotes. 'ft_sq' means feet represented as single quote and 'architectural' is the combination of 'ft_sq' and 'in_dq' with '-' as delimiter.

The length representation 'scientific' stands for floating-point numbers. 'fixed' represents fixed-point numbers. 'auto' shows the numbers depending on the value either as floating-point number or fixed-point number. 'fraction' represents the number as fraction.
'LengthUnitDisplayed ' defines whether units should be displayed. 'LengthUpperCase' defines whether the exponent should be represented with capital or small 'E' (only in case of 'scientific' or 'auto' with corresponding value).

'AngleZeroDirection' defines the direction of the 0-degree-angle in radian measure. It is subtracted from an absolute angle before formating. 'MaxAngle' defines the maximum angle in radian measure. After converting the angle which should be displayed into]-2pi,+2pi[2pi is subtracted in case the angle is bigger as the maximum angle or 2pi is added in case the angle is negative and after adding 2pi smaller or equal the maximum angle.

4.2.3 Rounding in Basket Service

Synopsis:
```
egr.eai.basket.BeginOfSwedishRounding=<swed-rnd-begin>
egr.eai.basket.RoundedDiscounts=<boolean>

<swed-rnd-begin>:   <empty>
                  | None
                  | SingleSalesPrice
                  | TotalPositionNetPrice
<empty>:
```

XXX (not used yet)

4.2.4 Article Number Prefix for Conversion to User Article

Synopsis: `egr.eai.basket.UserArticleNumberPrefix=<string>`

Default Value: empty string

XXX (not used yet)

4.2.5 Color Space

Synopsis: `egr.eai.gf.color_space=<LinearRGB|sRGB>`

Default Value: LinearRGB

Defines the color space the OpenGL rendering should use. Can also be defined in the server start-up file, whereas a specification in a session configuration file takes precedence of a specification in the server start-up file.

4.2.6 Session Features

A `session feature` is an application feature that can be disabled by the session configuration.

Application features are defined in the application configuration file and are set to a license feature, `true`, or `false`. An application feature is enabled if it is set to '`true`' or if it is set to a license feature and the license feature is enabled.

For a `session feature` to be enabled, it must be enabled as an application feature and it must not be disabled by the session configuration file. It is disabled by the session configuration file if and only if the session configuration file sets it to `false`.

A feature that is designated as a session feature may also be queried in a context where no session configuration is available, in which case it is treated like an ordinary application feature.

All application features are now treated as session features when they are used in a session-specific context.

Among possibly other uses, this change can be used to prevent the server to send any price data back to the client by setting the following keys to 'false' in the session start-up file:

```
egr.eai.basket.obx.ReadPriceData
egr.eai.basket.PriceCalculation
egr.eai.basket.obx.WritePriceData
egr.eai.ws.basket.ReturnPriceInfo
```

4.2.7 Path Substitution

Synopsis: `app.gf.data.path.substitution=<string>`

Entries with this key may be used to substitute substrings of the values of the `path` and `extpath` keys in package registration files. This key may be useful if an installation uses a mirrored/rsynced data directory whose local path differs from the original path.

The value is similar to the argument of the `s` commands in sed(1) and vi(1). It must consist of delimiter, pattern string, delimiter, replacement string, delimiter, and optional "i" flag.

The character used as delimiter is determined by the value's first non-whitespace character. The pattern string is a Java regular expression (see http:// docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html).

The replacement string is the string used to replace all matching substrings of the values of the `path` and `extpath` keys in package registration files. Note that backslash ('\') and dollar signs ('$') in the replacement string are treated specially. A dollar sign may be treated as a reference to a captured subsequence, and a backslash is used to escape the following character. Look for `appendReplacement` in http://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html for more information.

Note that the substitution is performed for the whole value of the `path` and `extpath` keys. This may make a difference as EAIWS allows the specification of multiple paths separated by semicolon (or colon on Unix-like operating systems). This should be kept in mind even if the installed data profiles contain only one path per entry, because EAIWS merges data profiles pertaining to the same manufacturer or concern into a single data profile to create a single catalog, if possible, and processes the merged data profile the same way as all other data profiles.

Example:

```
app.gf.data.path.substitution=,(^|[;:])/opt/pCon/_data/,$1/home/jpo/egr/data/,
```

4.2.8 Encoding

Synopsis: `app.gf.data.default_encoding=`<*string*>

Its value must be a character set name understood by Java. The default value is 'windows-1252' (a superset of ISO-8859-1 with graphical characters in the C1 range).

4.2.9 Property References

Property references consist of a previously defined property name enclosed in `${` and `}`. The property name must consist of a non-empty sequence of alphanumeric US-ASCII characters, underline (_) and/or period (`.`).

Property values that do not contain `${` are left unmodified. Property values that do contain `${` are processed as follows:

- `${` must start a valid property reference or processing of the start-up file will fail. There must be a previous definition of the referenced property in the same start-up file. The whole property reference is replaced by the referenced properties current value.
- `$$` is substituted by a single `$`, which will not be considered as a leading dollar sign in `${` or `$$`.
- Any other dollar sign is left unchanged.

One possible use of property references is easy switching between multiple copies of mirrored user application (e.g. pCon.box) data sets. The relevant part of the start-up file may look as follows:

```
.data=/home/data/ofml
#.data=/home/data/ofml/.zfs/snapshot/2017-10-07

app.gf.data.profile.registration=${.data}/pCon.box/catalogs/default.profil
es
app.gf.data.profile.path=${.data}/pCon.box/catalogs/profiles
app.gf.data.path.substitution=,(^|[;:])/opt/pCon/_data/,$1${.data}/,
```

4.2.10 Session Startup Modification Time

Synopsis: `egr.eai.ws.session.startup.modified_time`

Default value: the time the session start-up file has last been modified at the time it was read

This property is read-only. The value of this property is an ISO instant in UTC, such as 2011-12-03T10:15:30Z.

The property may be explicitly specified in a session start-up file, overriding its default value, although there is probably not much use in doing so.

4.2.11 Currencies

Synopsis: `app.basket.currencies`

Default value: `currencies`

The default value is the base name of the currency configuration file, without suffix .cfg, located in directory etc/.

The value of this property must be an URI-reference. If this reference is relative (i.e. does not contain a scheme) and consists of a single path segment that does not contain a period then .cfg is appended to the reference. Relative references are then resolved against the URI representing the etc/ directory.

The URI is then matched against the pattern returned by method getSourceURIPattern() of currency provider factories to select the factory used to produce a currency provider.

To use a Fixer-compatible currency provider (see https://fixer.io), the session start-up file property `app.basket.currencies` must be set to the request URL used to access the Fixer-compatible web service. The following restrictions apply:

- The path of the request URL must end with either `/latest` or with a slash followed by an ISOdate.
- The query component may contain only the following parameters:

  `access_key`

  the access key; required with data.fixer.io, but not necessarily required by alternative implementations of the service

  `base`

  the base/reference currency; The default value is EUR. Note that this is also the only base currency supported by the free pricing plan of the fixer.io service.

  `symbols`

  a comma-separated list of currency codes used to restrict the set of conversion rates provided by the web service; Without this parameter the fixer.io service returns about 170 conversion rates, and the exchangeratesapi.io service (see below) returns about 30 conversion rates.

Example: `http://data.fixer.io/api/latest?access_key=<access-key>`

If the request path ends with `/latest` then the provider's conversion rates are regularly updated a quarter past every full hour. This results in about 700 to 750 updates per month which is within the limit of the maximum number of monthly requests allowed by the free pricing plan of the fixer.io service as long as the access key is used by only one instance of EAIWS.

Responses from successful requests are cached in `var/ebasket/currency/fixer/cache` using a digest computed from the request URL as key (file name). The cache is used when the provider is first accessed after an EAIWS (re)start. If a matching entry is found then the behavior is as follows:

- If the request path ends with an ISO date then the cached data is used.
- If the request path ends with /latest then the connect and read timeouts for the request are both reduced to 5 seconds (instead of the default 30 seconds) and the cached data is used if the request does not produce a valid result (due to timeout or some other error).

To use this provider with other Fixer-compatible web service implementations they must be registered using application start-up property `egr.eai.basket.currency.fixer.api_base`. For instance, to use the open source Foreign exchange rates API that provides exchange rates published by the European Central Bank, the server start-up file should contain one or both of the following entries:

```
egr.eai.basket.currency.fixer.api_base=http://api.exchangeratesapi.io/
```

```
egr.eai.basket.currency.fixer.api_base=https://api.exchangeratesapi.io/
```

Then, the application or session start-up file property `app.basket.currencies` can be set to something like

```
https://api.exchangeratesapi.io/2019-03-18?base=USD&symbols=EUR,USD,RUB
```

The example references a currency provider that uses USD as base currency and provides the conversion rates for EUR, USD and RUB as published by the EZB for Mach 18, 2019.

4.2.12 Timeout for Sessions

Synopsis: `egr.eai.server.session_timeout=<`*`timeout-in-seconds`*`>`

Default Value: `300`

The value of this option specifies the number of seconds of inactivity the Online Configurator waits until it automatically closes the session and eventually deletes session specific data from disk (see also $4.1.7).

In addition to the timeout in seconds, the properties can be set to a duration similar to ISO 8601 (see https://en.wikipedia.org/wiki/ISO_8601#Durations), with the difference that 'Y' (year), 'M' (month) and 'W' (week) are not supported, nor are the basic and extended formats PYYYYMMDDThhmmss and P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].

4.2.13 Session Suspend

Synopsis: `egr.eai.server.session_suspend_timeout=<`*`timeout-in-seconds`*`>`

The value of this option specifies the number of seconds of inactivity the Online Configurator waits until it automatically moves the session specific data from RAM to disk. For session suspend to work, this property must be set to a value less than the session timeout.

In addition to the timeout in seconds, the properties can be set to a duration similar to ISO 8601 (see https://en.wikipedia.org/wiki/ISO_8601#Durations), with the difference that 'Y' (year), 'M' (month) and 'W' (week) are not supported, nor are the basic and extended formats PYYYYMMDDThhmmss and P[YYYY]-[MM]-[DD]T[hh]:[mm]:[ss].

4.2.14 FAPI-shell Command Timeout

Synopsis: `egr.eai.gf.cmd_timeout=<timeout-in-seconds>`

Default Value: `0`

Optional entry, `egr.eai.gf.cmd_timeout`. Again, the value must be a non-negative integer less than 2^31. It represents the actual timeout in seconds.

A value of `zero` (the default value) means 'no timeout'. However, if a maximum timeout is configured in the server startup file, that maximum timeout is used instead.

If the timeout is greater than the maximum timeout, the maximum timeout is used. Similarly, if the timeout is less than the minimum timeout, the minimum timeout is used.

The value of `egr.eai.gf.cmd_timeout` from the session startup file may be overridden by a property with the same name returned by `ISessionManagerListener.getStartupProperties()`.

4.2.15 Default image options (session start-up file)

Synopsis:

`egr.eai.server.export.default_image_options=<key/value list>`

Default Value: empty

Added support for session startup option `egr.eai.server.export.default_image_options=<key/value` list> It can be specified, with decreasing priority, as part of the options returned by ISessionManagerListener.getStartupProperties(), in the session startup file, or in the server startup file. A higher priority startup option value, even if `empty`, completely replaces a lower priority value.

The startup option value is decomposed into a possibly empty list of strings representing image export options as accepted by operation getGeneratedImage. Individual options are US-ASCII whitespace separated quoted strings or key/value pairs. In case of key/value pairs, key and value must be separated by a single equal sign (U+003D). Keys start with an optional dollar sign (U+0024) followed by a sequence of one or more identifiers separated by a single full stop / period (U+002E). Values are either a quoted string or a possibly empty sequence of non-whitespace characters. Key and value (decoded in case of a quoted string) are concatenated, separated by a single equal sign, before appended to the list of image export options.

Identifiers and quoted strings must adhere to ISO C89 rules.

The list of image export options is then validated the same way as done by operation getGeneratedImage.

If the startup option value does not conform to the rules described above, or the resulting list of image export options contains an invalid option, the openSession operation fails.

4.3 Command Line Options

In general, the Online Configurator is started as

```
$ java $JVM_OPTIONS -jar $EAI_SERVER_JAR $OPTIONS
```

where `$JVM_OPTIONS` are options passed to the Java Virtual Machine, `$EAI_SERVER_JAR` is the absolute or relative path to `EAI-Server.jar`, and `$OPTIONS` are options passed to the Online Configurator.

4.3.1 JVM Options

4.3.1.1 Application Root Directory

Synopsis: `-Degr.eai.fw.cmdline.AppRootDir=<EAIWS-dir>`

Default Value: current working directory

This option must be used if the installation directory of the Online Configurator (the `$EAIWS` directory) is not the current working directory when the Online Configurator is started. Otherwise the Online Configurator will not be able to find it's configuration files.

4.3.2 Online Configurator Options

4.3.2.1 Server Start-Up File

Synopsis: `-startup <startup-file-name>`

Default Value: `server`

This option is used to specify the base name of the server start-up file. If specified the Online Configurator uses `$EAIWS/etc/startup/<startup-file-name>.cfg` as the path of the server start-up file.

4.3.2.2 Destination of Log Messages

Synopsis: `-log <destination>`

`<destination>` is any non-empty sequence of `file` and `cons`, separated by comma

Default Value: `file`

This option controls where log messages are written to. If `file` is specified, log messages are written to a log file that can be found in the `$EAIWS/var/log` directory. If `cons` is specified, log messages are written to the console. If both are specified, log messages are written both to the log file and the console.

4.4 Configuration files

4.4.1 Media Types

The configuration file named 'mime.types' is located in etc/ directory of application. Each non-comment line (starting with '#') consists of a MIME/Media type followed by space and a non-empty space-separated case sensitive list of file extensions (without the period ('.') separating base name and extension).

Right now, this file defines media types for OBK and OBX files.

Version 4.7: Added etc/mime.types with pdf suffix mapped to application/pdf.

5 Web Service Interfaces

The EasternGraphics Online Configurator exposes it's functionality as a set of three web services accessible through the SOAP web service protocol:

Session Service: The sole purpose of the session service is to open and close user sessions.

Catalog Service: The catalog service is used to access the XCF catalogs of registered product data.

Basket Service: The basket service is used to create and modify a hierarchical structure of folders and article positions and to allow the configuration of article positions.

Project Service: The project service is used to handle project based header information like addresses.

The WSDL files that describe the SOAP protocol used to access the services of the Online Configurator can be downloaded from `http://`*`<host>`*`:`*`<port>`*`/EAI/`*`<service>`*`?wsdl`, where *`<host>`* is the name or IP address of the host the Online Configurator is running on, *`<port>`* is the configured HTTP port number of the Online Configurator, and *`<service>`* must be replaced by either `Session`, `Catalog`, `Basket` or `Project`.

A single instance of the Online Configurator is prepared to process a certain number of web service operations in parallel. Right now this number is hard-coded to 20 parallel operations. This limit is more or less independent from the maximum allowed number of configured and licensed operations per minute, as the mechanism used to throttle the throughput allows the accumulation of tickets in periods of inactivity, which can then be used for a short time to service requests at a much higher rate than suggested by the throughput limit.

Operations that work on a single session, however, are always serialized by the Online Configurator.

5.1 Web Service Definition Syntax

Few developers will implement their clients solely by reading the WSDL files and referenced XSD files. Instead, they will use tools to generate the client side stubs used to call web service operations. Furthermore, there is no rule to prevent future versions of the Online Configurator from using different web service protocols like JSON-RPC. For these reasons the specifications of the Online Configurator web service operations merely define the semantics of the operations and use a syntax that is agnostic about the web service protocol and client implementation.

The grammar used in the following sub-sections uses the same EBNF notation as is used in Extensible Markup Language (XML) 1.0 (Fifth Edition).

5.1.1 Types

Synopsis:	Type	::=	NamedType
		|	ConstructedType
	NamedType	::=	PrimitiveType
		|	DefinedType

5.1.1.1 Primitive Types

Synopsis:	PrimitiveType	::=	'boolean'	§5.1.1.1.1
		|	'int'	§5.1.1.1.2
		|	'string'	§5.1.1.1.3
		|	'decimal'	

Entities of primitive types always hold an value of the type. They are never null, even if the language binding uses reference types.

5.1.1.1.1 Boolean Type

Synopsis: PrimitiveType ::= 'boolean'

XSD: `type="xs:boolean"`

Entities of type `boolean` can assume the values `true` and `false`.

5.1.1.1.2 Integer Type

Synopsis: PrimitiveType ::= 'int'

XSD: `type="xs:int"`

Entities of type `int` can assume any integral value between -2147483648 and 2147483647, inclusive.

5.1.1.1.3 String Type

Synopsis: PrimitiveType ::= 'string'

XSD: `type="xs:string"`

Entities of type `string` consist of a possibly empty sequence of Unicode characters.

5.1.1.1.4 Decimal Type

Synopsis: PrimitiveType ::= 'decimal'

XSD: `type="xs:decimal"`

The type `decimal` represents the subset of real numbers that can be expressed by an optional plus (`+`) or minus (`-`) sign and two non-empty sequences of decimal digits (`0` through `9`), delimited by the decimal point (`.`), where the total number of significant digits must not be greater than 18[15]. Significant digits are all digits except leading zero left of the decimal point and trailing zero right of the decimal point.

5.1.1.2 Defined Types

Synopsis:	DefinedType	::=	EnumerationType	§5.1.1.2.1
		|	StructureType	§5.1.1.2.2
		|	AliasType	§5.1.1.2.3

5.1.1.2.1 Enumeration Type

Synopsis: EnumerationType ::= EnumTypeName

XSD: `type="`*`<enumTypeName>`*`"`

Entities of enumeration types always hold an value of the type. They are never null, even if the language binding uses a reference type to represent enumerations.

See §5.1.2.1 for how to define a new EnumTypeType.

5.1.1.2.2 Structure Type

Synopsis: StructureType ::= StructTypeName

XSD: `type="`*`<structTypeName>`*`"`

Entities of structure types always hold an instance of the type. They are never null, even if the language binding uses reference types to implement structure types.

15 This limitation is imposed by XSD for minimally conforming processors.

See §5.1.2.2 for how to define a new StructTypeName.

5.1.1.2.3 Alias Type

Synopsis: AliasType ::= AliasTypeName

XSD: In XSD, alias type names are replaced by their original type names, recursively if necessary.

See §5.1.2.3 for how to define a new AliasTypeName.

5.1.1.3 Constructed Types

Synopsis: ConstructedType ::= NillableType
| SequenceType
| NillableValueSequenceType

5.1.1.3.1 Nillable Type

Synopsis: NillableType ::= NamedType '*'

XSD: `<xs:element ... type="<type>" minOccurs="0"/>`

Entities of nillable types can hold a null value in addition to the values allowed by the underlying type.

In XSD, nillable types are implemented with `minOccurs="0"` instead of `nillable="true"` as this is more efficient, unless the nillable type is the element type of a sequence type. If the nillable type is the element type of a sequence type, `nillable="true"` must be used as `minOccurs` and `maxOccurs` are used to identify the element as a sequence.

5.1.1.3.2 Sequence Type

Synopsis: SequenceType ::= NamedType '[' ']'
| NillableType '[' ']'

XSD: `<xs:element ... type="<type>" minOccurs="0" maxOccurs="unbounded"/>`

XSD:
```
<xs:element ...
            type="<type>"
            nillable="true"
            minOccurs="0"
            maxOccurs="unbounded"/>
```

5.1.2 Type Definitions

Synopsis: TypeDefinition ::= EnumTypeDefinition
| StructTypeDefinition
| AliasTypeDefinition

5.1.2.1 Enumeration Type Definitions

Synopsis: EnumTypeDefinition ::= 'enum' Name '{' Enumerator (',' Enumerator)* '}'

Enumerator ::= Name

XSD:
```
<xs:simpleType name="<enumTypeName>">
<xs:restriction base="xs:string">
<xs:enumeration value="<Enumerator>"></xs:enumeration>
...
</xs:restriction>
</xs:simpleType>
```

The enumeration type definition defines Name as a new EnumTypeName (§5.1.1.2.1).

5.1.2.2 Structure Type Definitions

Synopsis: StructTypeDefinition ::= 'struct' Name '{' (Declaration ';')* '}'

XSD:

```
<xs:complexType name="<structTypeName>" final="extension restriction">

<xs:sequence>
<xs:element name="<fieldName>" ... />
...
</xs:sequence>
</xs:complexType>
```

The structure type definition defines Name as a new StructTypeName (§5.1.1.2.2).

5.1.2.3 Alias Type Definitions

Synopsis: AliasTypeDefinition ::= 'typedef' Type Name ';'

XSD: In XSD, all alias types are replaced by the original types, recursively if necessary.

The alias type definition defines Name as a new AliasTypeName (§5.1.1.2.3).

5.1.3 Declarations

Synopsis: Declaration ::= Type Name

XSD: `<xs:element name="<name>" type="<type>" ... />`

If Type is a constructed type, the attributes `minOccurs="0"`, `maxOccurs="unlimited"` and `nillable="true"` may be added as described in §5.1.1.3. Otherwise, they keep their default values of `1`, `1` and `false`.

A declaration declares a new entity (structure field, operation argument) with the given type and name.

5.1.4 Operations

Synopsis:

```
Operation        ::= ResultType OperationName '(' ArgumentList? ')'
                        ThrowsClause? ';'

ResultType       ::= Type
                   | 'void'

OperationName    ::= Name

ArgumentList     ::= Argument ( ',' Argument )*

Argument         ::= Declaration
```

5.1.5 Names

Synopsis:

```
Name             ::= NameStartChar NameChar*
NameStartChar    ::= Letter | '_'

NameChar         ::= NameStartChar | Digit

Letter           ::= [A-Z] | [a-z]

Digit            ::= [0-9]
```

Boolean Values

Synopsis: **`boolean`**

Entities of type `boolean` can assume the values `true` and `false`.

5.1.5.1 Integers

Synopsis: **`int`**

Entities of type `int` can assume any integral value between -2147483648 and 2147483647, inclusive.

Boolean Values

Synopsis: **`boolean`**

Entities of type `boolean` can assume the values `true` and `false`.

5.1.5.2 Integers

Synopsis: **`int`**

Entities of type `int` can assume any integral value between -2147483648 and 2147483647, inclusive.

Strings

Synopsis: **`string`**

Entities of type `string` consist of a possibly empty sequence of Unicode characters.

5.2 Common Type Definitions

5.2.1 `UUID`

Synopsis:

```
typedef string UUID;
```

An Universally Unique Identifier (UUID) is a 128-bit number that is meant to uniquely identify a resource over time and space[16] without the need for central coordination in the generation of UUIDs.

The Online Configurator web service interfaces use strings to represent UUIDs. For this purpose UUIDs are formatted as 32 hexadecimal digits (using lower case letters), separated by hyphens (`-`) into five groups of 8, 4, 4, 4, and 12 digits.

In some contexts, a special null-UUID is required to indicate special semantics. Because the Online Configurator web service interfaces prohibit the use of `null` for entities of type `string`, either the empty string (`""`) or the string `"00000000-0000-0000-0000-000000000000"` may be used as a null-UUID. The Online Configurator always uses the second form for null-UUIDs returned by web service operations.

5.2.2 URL

Synopsis:

```
typedef string URL;
```

Entities of the URL type are either a HTTP URL or an empty string. The host and port of the URL refer to the

16 In theory, due to the limited value range of about 3 x 10^{38}, it could happen that the same UUID is accidentally used to identify different resources. In practice, however, this is quite unlikely and, given a reasonable generator for UUIDs, one can assume that one such identifier will never be unintentionally used for another resource.

same host and port as used to access the Online Configurator web services.

5.3 Application-Specific Data

The client is able to attach application-specific data to the project, the basket, and individual basket items. This data is written to and read from project files by the `saveSession` (§5.4.3.14) and `loadSession` (§5.4.3.15) operations of the `SessionService`. The operations used to access this data are:

- `setProjectAppData` (§5.4.3.18) and `getProjectAppData` (§5.4.3.19) of the `SessionService` for data attached to the project. This data is written to the `metainf.xml` project file member as child elements of the `<appData>` element.
- `setBasketAppData` (§5.6.3.46) and `getBasketAppData` (§5.6.3.47) of the `BasketService` for data attached to the basket. This data is written to the `1.bsk` project file member as child elements of the `<appData>` element that is a child of the `<basket>` element.
- `setItemAppData` (§5.6.3.48) and `getItemAppData` (§5.6.3.49) of the `BasketService` for data attached to individual items. This data is written to the `1.bsk` project file member as child elements of the `<appData>` element that is a child of the element representing the basket item. It is also part of the OBX stream that is used for cut/copy/paste and may be used by some export routines exporting commercial data (§5.6.3.43, GFX export).

All operations that access application data have an argument of type `string` named `appKey`. This argument is intended to identify the application managing this particular kind of data. For each application key, the Online Configurator uses an `<application>` element as child of the `<appData>` element, with an attribute named `key` whose value is the specified application key. The `<application>` elements are created by operations that set the application data. Operations that get the application data do not create this (or any other) element.

5.3.1 Setting Application Data

The operations that set application data take an argument named `data` of type `string[]`. Each element of this sequence of strings must consist of a location path and the value to set, separated by an equal sign (`'='`). The value consists of all characters right of the equal sign, including any leading or trailing white space. The operation fails if one of the elements of `data` does not contain an equal sign separating the location path and the value, or if the location path does not adhere to the syntax for location paths specified below (§5.3.3.1). If the operation fails at this point, no application data has been modified.

Once `data` has been successfully validated, the operation evaluates each location path. If the location path references an XML element, all children of the element are removed, and if the new value is not an empty string, replaced by a single Text node containing the value. If the location path references an XML attribute, the value of this attribute is set to the new value.

If the location path does not reference an existing node, evaluation of the location path creates new nodes as required so evaluation of the location path always produces at least one node. Therefore, once the elements of `data` have been successfully validated, the operations setting application data should not fail. Thus, the operations setting application data are atomic, setting either all referenced nodes to their new values, or none.

5.3.2 Getting Application Data

The operations that get application data take an argument named `paths` of type `string[]`. The return value is of type `string*[]`. The return value has always as many elements as the argument `paths`.

Each element of `paths` must consist of a single location path that adheres to the syntax for location paths as specified below (§5.3.3.1), or the operation will fail. Once all location paths have been validated, each location path is evaluated. If evaluation produces an empty node set, the corresponding element in the return value is set to *null*. Otherwise, the first node in document order from the node set is chosen and its string value is assigned to the corresponding element in the return value. The string value of an XML element node consists of the concatenation of all Text and CDATASection descendant nodes in document order. The string value of an XML attribute node consists of the attribute’s value.

5.3.3 Location Paths

5.3.3.1 Syntax of Location Paths

Location paths supported by the operations that set and get application data are a very limited declarative[17] subset of XPath location paths. They must adhere to the following grammar:

```
LocationPath        ::=  ElementTest  ( '/'  ElementTest  )*  ( '/'  AttributeTest  )?
ElementTest         ::=  NCName  Predicate?
AttributeTest       ::=  AttributeRef
Predicate           ::=  '['  PredicateExpr  ']'
PredicateExpr       ::=  AndExpr
AndExpr             ::=  EqualityExpr  (  'and'  EqualityExpr  )*
EqualityExpr        ::=  FunctionCall
                      |  AttributeRef  '='  PrimaryExpr
FunctionCall        ::=  'not'  '('  AttributeRef  ')'
AttributeRef        ::=  '@'  NCName
PrimaryExpr         ::=  Literal
                      |  Number
NCName              ::=  NCNameStartChar  NCNameChar*
NCNameStartChar     ::=  [A-Z]  |  "_"  |  [a-z]  |  [#xC0-#xD6]  |  [#xD8-#xF6]  |  [#xF8-#x2FF] |
                         [#x370-#x37D]  |  [#x37F-#x1FFF]  |  [#x200C-#x200D]  |
                         [#x2070-#x218F]  |  [#x2C00-#x2FEF]  |  [#x3001-#xD7FF]  |
                         [#xF900-#xFDCF]  |  [#xFDF0-#xFFFD]  |  [#x10000-#xEFFFF]
NCNameChar          ::=  NameStartChar
                      |  "-"  |  "."  |  [0-9]  |  #xB7  |  [#x0300-#x036F]  |  [#x203F-#x2040]
Literal             ::=  '"'  [^"]*  '"'
                      |  "'"  [^']*  "'"
Number              ::=  [+-]?  Digits  (  '.'  Digits?  )?
                      |  [+-]?  '.'  Digits
Digits              ::=  [0-9]+
```

Whithin literals, the delimiting character (QUOTATION MARK or APOSTROPHE) and AMPERSAND (&) must be represented by character references. The value of a numeric character reference must represent the code point of a valid XML character.

```
CharacterReference  ::=  NumericCharacterReference
                      |  '&amp;'  |  '&quot;'  |  '&apos;'  |  '&lt;'  |  '&gt;'
NumericCharacterReference  ::=
                         '&'  [0-9]+  ';'
                      |  '&x'  [0-9A-Fa-f]+  ';'
```

[17] Location paths used by the operations that set and get application data are declarative because each step of the location path can not only be used to match against existing nodes, but must also declare how to insert a new node if no match is found (operations that set application data must be able to create nodes). In XPath, a relative location path consisting of a single step may look like `Foo[@bar!='abc']`. If this syntax would be supported by operations that set application data, and the application data does not contain a matching node, the operation would have to create a new node, but would not know which value to use for the attribute `'bar'`, or whether to add this attribute at all. On the other hand, if the location path is `Foo[@bar='abc' and not(@baz)]`, and the operation does not find a matching node, it knows to create an element named `<Foo>` with an attribute `'bar'` whose value is `'abc'` and without an attribute `'baz'`.

XMLCharacter	::=	#09 | #0A | #0D | [#0020-#D7FF] | [#E000-#FFFD] | [#10000-#10FFFF]

White space (SPACE, CHARACTER TABULATION, LINE FEED and CARRIAGE RETURN) is allowed between individual grammar symbols except within the productions of NCName, Literal, Number and NumericCharacterReference.

WhiteSpace	::=	#x20 | #x09 | #x0A | #x0D

5.3.3.2 Evaluation of Location Paths

Location paths are split into a sequence of *n* individual node tests (ElementTest, AttributeTest). Each node test has a set of input nodes and a set of output nodes. The output nodes of node test *n* become the input nodes of node test *n+1*. The input set of the first node tests consists of the `<application>` element selected by the application key. The output set of the last node test represents the result of the location path's evaluation.

The output set of an element test consists of the immediate child nodes of all nodes from the input set for which the following conditions are true:

- The node is an element node.
- The tag name of the element is equal to the name specified by the node test.
- If the element test contains a predicate, the predicate expression is true.

Predicate expressions are used to test attributes of the (element) node for their value. They contain one or more equality expressions separated by the token `and`. An equality expression may have one of three forms:

`@`*`<NCName>=<Number>`*

This expression is true if and only if the currently tested node has an attribute named *<NCName>* whose value can be converted into a number, and the resulting number has the same value as *<Number>*. The conversion of the attribute value to a number accepts at least the same character sequences as the rule for Number in the grammar above.

`@<NCName>=<Literal>`

This expression is true if and only if the currently tested node has an attribute named *<NCName>* whose value contains the same sequence of Unicode code points as *<Literal>*.

`not(@`*`<NCName>`*`)`

This expression is true if and only if the currently tested node does not have an attribute named *<NCName>*.

The output set of an attribute tests consists of the immediate child nodes of all nodes from the input set for which the following conditions are true:

- The node is an attribute node.
- The name of the attribute is equal to the name specified by the node test.

If the operation sets application data, and the output set of a node test is empty, the node test creates a new node that matches the node test and appends it to the input set's first node (in document order). The output set then contains the newly created node.

5.4 Session Service

5.4.1 Type Definitions

5.4.1.1 `SessionId`

Synopsis:

```
typedef UUID SessionId;
```

Session IDs are generated by the `openSession` operation to uniquely identify the newly opened session. They remain valid until the session has been explicitly closed or has timed out.

5.4.1.2 `StringPair`

Synopsis:

```
     struct StringPair {
         string  first;
         string  second;
     }
```

Instances of type `StringPair` are used to specify option arguments of the `openSession` operation. The value of `first`, which must start with a hyphen (`-`), indicates the option. The value of `second` is the actual value of the option. If the option has no value, then `second` must be the empty string (`""`).

5.4.1.3 `ProjectData`

Synopsis:

```
struct ProjectData {
    string* number;
    string* name;
    string* description;
    string* personInCharge;
    string* extRefNr;
    string* extRefText;
}
```

5.4.1.4 `ClientMessage`

Synopsis:

```
     final struct ClientMessage{
                              string       target;
                              string       messageId;
                              StringPair[] data;
                              }
```

```
    target
```

Identifies the target of the message. It consists of a prefix and a prefix-specific string. As of now, the prefix must be plugin: and the substring following the prefix must be empty or equal to a plugin ID.

```
    messageId
```

identifies the message. It must adhere to the rules for QName as specified in Namespaces in XML 1.1, Second Edition). For plugin-specific messages it should be just a NCName.

```
    data
```

Is the payload of the message, a possibly empty list of key/value pairs. Keys must adhere to the rules for NCName.

5.4.1.5 `CustomerData`

Synopsis:

```
struct CustomerData {
    string* customerNumber;
    string* salutation;
    string* firstName;
    string* lastName;
    string* emailAddr;
    string* company;
    string* organization;
    string* streetAddr;
    string* city;
    string* stateOrProvince;
    string* cityCode;
    string* poBox;
    string* postalCode;
    string* poBoxPostalCode;
    string* country;
    string* language;
    string* phone;
    string* fax;
}
```

The format of `postalCode` and `poBoxPostalCode` follows the OEX (OFML Business Data Exchange) Global specification:

Maximum Length:	10 characters
Allowed Characters:	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ space and – within. Whereas it is not permitted for several spaces/hyphens to follow one another.

5.4.1.6 `ProjectSettings`

Synopsis:

```
struct ProjectSettings {
    ProjectData* projectData;
    CustomerData* customerData;
}
```

The `ProjectSettings` structure corresponds to the `<settings>` element within the `metainf.xml` file of pCon.basket project (OBK) files. It is used as result type of the `getProjectSettings` (§5.4.3.17) and argument type of the `setProjectSettings` (§5.4.3.16) operations of the `SessionService`.

5.4.1.7 `SendMessageStatus`

Synopsis:

```
enum SendMessageStatus{
                    OK,
                    UnexpectedError,
                    UnknownTargetType,
                    UnknownPlugin,
                    NoPluginImplementation,
                    NoMessageHandler,
                    UnknownMessageId,
                    InvalidMessageData
                   }
```

5.4.1.8 `SessionCacheFileType`

Synopsis:

```
enum SessionCacheFileType {
    Project,
    CutBuffer,
    Import
}
```

Defines the valid session cache file types. Is used for example by operation getUploadURL. `Project` files have the suffix .obk and `CutBuffer` files end with.obx.

The enumeration value `Import` should be used with operation getUploadURL to get an URL that can be used with HTTP PUT to upload a file that can later be imported into the session with operation `importFile` (§5.4.3.21).

5.4.1.9 `ImportFileOptions`

Synopsis:

```
struct ImportFileOptions {boolean  deleteAfterImport}
```

The complex type `ImportFileOptions` contains optional elements representing options that control the behavior of the importFile (§5.4.3.21) operation. Right now, there is only one option of type boolean with name `deleteAfterImport`. If set to true, the importFile operation deletes the file after a successful import.

5.4.1.10 `SaveSessionOptions`

Synopsis:

```
struct SaveSessionOptions {
    string*  suffix;
    boolean* overwrite;
    boolean  generateImages,
    boolean  omitPriceData,
    boolean* saveCalculations;
    boolean* saveLegacyCalculation;
    string*  legacyCalculationName;
    boolean  export
}
```

The complex type `SaveSessionOptions` contains optional elements representing options that control the behavior of the `saveSession` (§5.4.3.14) operation.

If the `suffix` option is not specified, the operation uses ".obk" as the suffix of the project file. If the client sets the suffix option to an empty string, then no suffix is appended to the project file name. Otherwise, the Online Configurator checks whether the suffix starts with a dot, prepending one if it does not, and appends the resulting suffix to the project file name.

The optional element `saveCalculations` controls whether or not (normal) calculation data is saved. Default value is `true`.

The optional element `saveLegacyCalculation` controls whether or not legacy calculation data (used by pCon.basket) is saved. Default value is `false`. The optional element `legacyCalculationName` explicitly specifies the pricing procedure name of the calculation to use for legacy calculation data. Default value is the empty string. Ignored if `saveLegacyCalculation` is `false`.

If access to the file is allowed, and the file already exists, the operation fails unless the `overwrite` option has been set to `true`.

export: prevents, when set to true, the project's URI set to be updated to reflect the location where the project has been saved to. (The default value of this option is false.)

5.4.1.11 `OperatingSystemInformation`

Synopsis:

```
struct OperatingSystemInformation {
    string name;
    string architecture;
    string version;
    int    availableProcessors;
    double systemLoadAverage;
}
```

`name (element)`

the operating system name

`architecture (attribute)`

the operating system architecture

`version (attribute)`

the operating system version

`availableProcessors (attribute)`

the number of processors available to the Java virtual machine

This value may change between subsequent invocations of the `getSystemInformation` (§5.4.3.23) operation for the same instance of the Online Configurator.

`systemLoadAverage (attribute)`

the system load average for the last minute; The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over a period of time. The way in which the load average is calculated is operating system specific but is typically a damped time-dependent average.

If the load average is not available, a negative value is returned.

5.4.1.12 `RuntimeInformation`

Synopsis:

```
struct RuntimeInformation {
    string name;
    string vmName;
    string vmVendor;
    string vmVersion;
    string specName;
    string specVendor;
    string specVersion;
    long   uptime;
    long   startTime;
    string compilerName;
    long   totalCompilationTime;
    long   loadedClassCount;
    long   totalLoadedClassCount;
    long   unloadedClassCount;
}
```

`name (element)`

the name representing the running Java virtual machine; The returned name string can be any arbitrary string and a Java virtual machine implementation can choose to embed platform-specific useful information in the returned name string. Each running virtual machine could have a different name.

`vmName (element)`

the Java virtual machine implementation name

`vmVendor (element)`

the Java virtual machine implementation vendor

`vmVersion (attribute)`

the Java virtual machine implementation version

`specName (element)`

the Java virtual machine specification name

`specVendor (element)`

the Java virtual machine specification vendor

`specVersion (attribute)`

the Java virtual machine specification version

`uptime (attribute)`

the uptime of the Java virtual machine in milliseconds

`startTime (attribute)`

the start time of the Java virtual machine in milliseconds; This method returns the approximate time when the Java virtual machine started.

`compilerName (element)`

the name of the Just-in-time (JIT) compiler

`totalCompilationTime (attribute)`

the approximate accumulated elapsed time (in milliseconds) spent in compilation; If multiple threads are used for compilation, this value is summation of the approximate time that each thread spent in compilation.

`loadedClassCount (attribute)`

the number of classes that are currently loaded in the Java virtual machine

`totalLoadedClassCount (attribute)`

the total number of classes that have been loaded since the Java virtual machine has started execution

`unloadedClassCount (attribute)`

the total number of classes unloaded since the Java virtual machine has started execution

5.4.1.13 `MemoryUsage`

Synopsis:

```
struct MemoryUsage {
    long init;
    long used;
    long committed;
    long max;
}
```

`init (attribute)`

the amount of memory in bytes that the Java virtual machine initially requests from the operating system for memory management; This value is -1 if the initial memory size is undefined.

`used (attribute)`

the amount of used memory in bytes

`committed (attribute)`

the amount of memory in bytes that is committed for the Java virtual machine to use; This amount of memory is guaranteed for the Java virtual machine to use.

`max (attribute)`

the maximum amount of memory in bytes that can be used for memory management; This value is -1 if the maximum memory size is undefined.

This amount of memory is not guaranteed to be available for memory management if it is greater than the amount of committed memory. The Java virtual machine may fail to allocate memory even if the amount of used memory does not exceed this maximum size.

5.4.1.14 `GarbageCollectorInformation`

Synopsis:

```
struct GarbageCollectorInformation {
    string name;
    long   collectionCount;
    long   collectionTime;
}
```

`name (element)`

the name representing this garbage collector

`collectionCount (attribute)`

the total number of collections that have occurred; This value is -1 if the collection count is undefined for this collector.

`collectionTime (attribute)`

the approximate accumulated collection elapsed time in milliseconds; This value is -1 if the collection elapsed time is undefined for this collector.

The Java virtual machine implementation may use a high resolution timer to measure the elapsed time. This method may return the same value even if the collection count has been incremented if the collection elapsed time is very short.

5.4.1.15 `MemoryInformation`

Synopsis:

```
struct MemoryInformation {
    int         objectPendingFinalizationCount;
    MemoryUsage heap;
    MemoryUsage nonHeap;
    GarbageCollectorInformation[] collectors;
}
```

`objectPendingFinalizationCount (attribute)`

the approximate number of objects for which finalization is pending

`heap (element)`

the current memory usage of the heap that is used for object allocation. The heap consists of one or more memory pools. The used and committed size of the returned memory usage is the sum of those values of all heap memory pools whereas the init and max size of the returned memory usage represents the setting of the heap memory which may not be the sum of those of all heap memory pools.

The amount of used memory in the returned memory usage is the amount of memory occupied by both live objects and garbage objects that have not been collected, if any.

`nonHeap (element)`

the current memory usage of non-heap memory that is used by the Java virtual machine; The non-heap memory consists of one or more memory pools. The used and committed size of the returned memory usage is the sum of those values of all non-heap memory pools whereas the init and max size of the returned memory usage represents the setting of the non-heap memory which may not be the sum of those of all non-heap memory pools.

`collectors (element)`

information about one or more garbage collectors used by the Java virtual machine.

The number of collectors may change between subsequent invocations of the`getSystemInformation` (§5.4.3.23) operation for the same instance of the Online Configurator.

5.4.1.16 `ApplicationInformation`

Synopsis:

```
struct ApplicationInformation {
    string appName;
    string appKey;
    string appVersion;
    string vendorKey;
}
```

`appName (element)`

the display name of the application, or the application key if no display name is available

`appKey (attribute)`

symbolic identifier for the application

`appVersion (attribute)`

the version number of the application; The returned version number adheres to the following BNF grammar:

```
version_number = major
               | major "." minor
               | major "." minor "." micro
               | major "." minor "final"
               | major "." minor level micro

major = 1*digit

minor = 1*digit

micro = 1*digit

level = "alpha" | "beta" | "rc"

digit = "0" | "1" | "2" | "3" | "4"

    | "5" | "6" | "7" | "8" | "9"
```

`vendorKey (attribute)`

symbolic identifier for the vendor distributing the application

5.4.1.17 `SendMessageResult`

Synopsis:

```
final struct SendMessageResult{
            boolean          sessionAlive;
```

```
                    ServerResponse[] messages;
                    }
```

5.4.1.18 `ServerInformation`

Synopsis:

```
struct ServerInformation {
    int  maxConcurrentSessionCount;
    int  openSessionCount;
    int  activeSessionCount;
    long sessionTimeout;
    long sessionSuspendTimeout;
}
```

`maxConcurrentSessionCount (attribute)`

The configured maximum number of concurrent sessions

`openSessionCount (attribute)`

The number of sessions open at some point during the execution of the `getSystemInformation` (§5.4.3.23) operation

`activeSessionCount`

The field `activeSessionCount` is filled by the session service operation `getSystemInformation` to return the number of sessions active at some point during the execution of that operation.

`sessionTimeout (attribute)`

The configured session time out in milliseconds

`sessionSuspendTimeout (attribute)`

The configured session suspend timeout in milliseconds. The attribute's value is -1 if no session suspend timeout has been configured.

5.4.1.19 `SystemInformation`

Synopsis:

```
struct SystemInformation {
    OperatingSystemInformation *operatingSystem;
    RuntimeInformation         *runtime;
    MemoryInformation          *memory;
    ApplicationInformation     *application;
    ServerInformation          *server;
}
```

This complex type acts as a container for information returned by operation `getSystemInformation` (§5.4.3.23). Its elements are optional. Whether or not they are present depends on the `options` argument passed to `getSystemInformation` (§5.4.3.23).

5.4.1.20 `GetItemPropertiesTextMode`

Synopsis:

```
enum GetItemPropertiesTextMode {
       Legacy,
       Simple,
       Multi,
       Mixed}
```

5.4.1.21 `GetSystemInformationOptions`

Synopsis:

```
struct GetSystemInformationOptions {
    boolean *operatingSystem;
    boolean *runtime;
    boolean *memory;
    boolean *application;
    boolean *server;
}
```

Used by `getSystemInformation` (§5.4.3.23). Allows restriction of the returned information to the subset that is actually needed.

5.4.1.22 `LogConfig`

Synopsis:

```
struct LogConfig {
    int       *maxRecordCount;
    boolean   *discardOldest;
    boolean   *storeStackTrace;
    string    *timeZoneId;
    LogFilter *filter;
}
```

`maxRecordCount`

the maximum number of records to store in the buffer

default: `1000`

`discardOldest`

affects the behavior if a new log message is to be added to the buffer in case the buffer is already full (already contains `maxRecordCount` messages); If true, the oldest message from the buffer is discarded, and the new message is added. If false, the new message will be discarded.

default: `true`

`storeStackTrace`

whether or not to store the stack trace of exceptions that are the cause of log messages

default: `false`

`timeZoneId`

the time zone ID to use for log message time stamps returned to the client. See https://en.wikipedia.org/wiki/List_of_tz_database_time_zones for a list of supported time zones (column 'TZ').

default: UTC

`filter`

the filter to select the messages to be added to the session-specific log buffer

default: no filter

5.4.1.23 `LogFilter`

Synopsis:

```
struct LogFilter {
    LogFilterSpec[] include;
    LogFilterSpec[] exclude;
}
```

`include, exclude`

A message is added to the session-specific log buffer if

- either the list of include specs is empty or the message matches at least one include spec,

AND

- the message does not match an exclude spec.

5.4.1.24 `LogFilterSpec`

Synopsis:

```
struct LogFilterSpec {
    string *level;
    string *facility;
    string *origin;
}
```

`level, facility, origin`

optional POSIX Extended Regular Expressions that must match, if specified as a non-empty string, the whole level, facility or origin of the message.

Matching of the level is done case-insensitive, whereas matching of facility and origin is case sensitive.

A log filter spec that contains neither level, facility nor origin matches any log message.

5.4.1.25 `LogData`

Synopsis:

```
struct LogData {
    int         discarded;
    int         suppressed;
    LogRecord[] data;
}
```

5.4.1.26 `LogRecord`

Synopsis:

```
struct LogRecord {
    string      timeStamp;
    string      level;
    string      facility;
    CallSite    *origin;
    string      message;
    FaultInfo[] cause;
}
```

`timeStamp`

the time when the log message has been generated, such as

2017-12-03T10:15:30+01:00[Europe/Paris]

`level`

`Fatal` (should never happen), `Error`, `Warning`, `Notice`, `Info`, `Config`, or `Debug`

`facility`

a sequence of one or more identifiers separated by single or double colon; Usually, the facility specified the module that produced the log message. Special facilities are OFML for OFML-related problems detected by EAIWS, OBX for syntactic errors when reading BSK/OBX streams, and GF::* for messages generated by FAPI-Shell.

`origin`

if present, the origin specifies the source code location where the log message has been generated.

`message`

the actual log message

`cause`

if the log message was caused by an exception, then this element contains information about the exception.

If cause contains *N* elements with *N* >= 2, then fault *I* (with *I* >= *0* and *I+1* < *N*) is caused by fault *I+1*, i.e. the faults and indices greater than or equal to one represent nested (or inner) exceptions. Suppressed exceptions are not reported to the client.

5.4.1.27 `FaultInfo`

Synopsis:

```
struct FaultInfo {
    string     type;
    string     message;
    CallSite[] stackTrace;
}
```

5.4.1.28 `CallSite`

Synopsis:

```
struct CallSite {
    string className;
    string methodName;
    string *fileName;
    int    *lineNumber;
}
```

`className, methodName`

the name of the method, and the name of the class containing the method, that generated the log message

`fileName, lineNumber`

if present, the source file name and line number where the log message has been generated; This information may be missing as it requires the availability of debug information for the method that generated the log message.

5.4.1.29 `ResolveURIsOptions`

Synopsis:

```
struct ResolveURIsOptions {string serverBase}
```

serverBase

The server base, if specified, must be an absolute hierarchical URI that is used instead of the server base derived from the HTTP header of the SOAP request. It affects the URLs returned by the operation as well as URLs embedded in files referenced by these URLs.

5.4.1.30 `ServerResponse`

Synopsis:

```
final struct ServerResponse{
                                string            origin;
                                string            messageId;
                                StringPair[]      data;
                                SendMessageStatus statusCode;
                                string[]          errorDetail;
                                string           targetErrorCode
                         }
```

```
origin
```

Identifies the origin of the response. Same rules as for ClientMessage.target, but the substring following plugin: is always non-empty.

```
messageId
```

Identifies the message. It is always equal to the messageId of the corresponding ClientMessage.

```
data
```

Is the payload of the response, a possibly empty list of key/value pairs. Keys must adhere to the rules for NCName. data is always empty if statusCode is not OK.

```
statusCode
```

Is used to signal and identify error during message delivery.

```
errorDetail
```

Is used to provide additional information for some kinds of errors.

5.4.1.31 `LoadSessionOptions`

Synopsis:

```
LoadSessionOptions (
      string suffix,
      boolean assignNewProjectId,
)
```

5.4.2 Faults

5.4.2.1 `SessionServiceFault`

Synopsis:

```
    struct SessionServiceFault {
        string* message;
    }
```

A `SessionServiceFault` is returned by operations of the session service in case of errors the Online Configurator was prepared to detect and deal with (i.e. to recover cleanly).

5.4.3 Operations

5.4.3.1 `hasOpenSession`

Synopsis:

```
    boolean hasOpenSession();
```

The `hasOpenSession` operation can be called any time to test whether the Online Configurator has an open session. Naturally, the client should not have an open session when it calls this operation, albeit doing so would cause no harm other than the waste of computing resources.

5.4.3.2 `openSession`

Synopsis:

```
    SessionId openSession(StringPair[] args) throws SessionServiceFault;
```

The `openSession` operation is used to create a new session. As virtually all other web service operations of the Online Configurator require an open session, this operation is usually the first operation called by a client of the Online Configurator.

It is not an error if a single client simultaneously opens and works with multiple sessions.

The following options are defined for `openSession`:

`-startup <startup-file-name>`

The `-startup` option specifies the base name of the session start-up file. The full path name of the session start-up file is `$EAIWS/etc/startup/`*`<startup-file-name>`*`.cfg`.

`-locale <locale-name>`

The `-locale` option may be used to specify the locale used by the session. It overrides the locale specified in the session or server start-up files. For more information see sections 4.1.1 and 5.4.3.10.

Unknown options are ignored.

A `SessionServiceFault` is returned if any of the following conditions occurs:

- The maximum number of configured or licensed sessions has been reached.
- The `<startup-file-name>` does not name an existing session start-up file.
- Numerous other errors, usually caused by incorrect configuration of the Online Configurator.

5.4.3.3 `closeSession`

Synopsis:

```
    void closeSession(SessionId sessionId) throws SessionServiceFault;
```

The `closeSession` operation closes the session identified by the `sessionId` argument. Once this operation has executed, any other operation using the same `sessionId` will fail.

The invocation of the `closeSession` operation has no effect if the `sessionId` argument does not identify an open session.

The `closeSession` operation can't be used with a session context ID as their first argument.

A `SessionServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID.

5.4.3.4 `keepAlive`

Synopsis:

```
boolean keepAlive(SessionId sessionId) throws SessionServiceFault;
```

In order to avoid a session being closed due to an expired session timeout the client may periodically invoke the `keepAlive` operation. If the `sessionId` argument identifies an open session then the session's timeout timer is reset to it's initial value (§4.1.6) and the operation returns `true`. Otherwise, the operation returns `false`.

The `keepAlive` operation can't be used with a session context ID as their first argument.

A `SessionServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID.

5.4.3.5 `newSessionContext`

Synopsis:

```
SessionId newSessionContext( SessionId              sessionId,
                             SessionContextData?    ContextData)
        throws SessionServiceFault;

struct SessionContextData {
        UUID? projectKey;                        // attribute
}
```

Session contexts are a mechanism to to allow clients to pass implicit parameters to (web service) operations that are required by most operations. As of now, the only such parameter is the project key (a kind of handle for an in-memory instance of a project).

To use a session context, a client will create a session context, set the implicit parameters stored in the session context, and use the session context ID instead of the session ID as the sessionId argument of web service operations. The implementations of these operations will then use the session ID stored in the session context as the actual session ID, and use implicit parameters stored in the session context if and when needed (in other words, an operation which does not require a project should not fail just because the session context contains a project key which does not reference one of the session's open projects).

Once no longer needed, a client may dispose a session context. All session contexts of a session are automatically disposed when the session is closed.

In general, and unless explicitly specified otherwise, all operations which take a session ID as their first argument may be invoked with a session context ID. If so, the session ID stored in the session context will be used as the actual session ID.

Furthermore, if the operation requires a project, the project key stored in the session context will be used to identify the project to operate on. The operation will fail if the project key does not identify one of the projects currently held open by the session. On the other hand, if the project key of the session context is not specified, the operation will operate on the session's current project (the same project as used with operations invoked with an actual session ID).

The operation `newSessionContext` creates a new session context for the session specified by the first argument.

Depending on whether argument `sessionId` is a session ID or a session context ID, the default value for the project key stored in the new session context is either the key of the session's current project, or the project key from the original session context, or unspecified if the original session context does not specify a project key (in which case operations on the session context will operate on the session's current project unless a project key has been set for the session context).

The optional `contextData` argument, if specified, is used to initialize the session context. See operation `updateSessionContext` for more information.

The operation returns a session context ID which can be used as the `sessionId` argument of most EAIWS web service operations until the context is disposed, either explicitly with operation `disposeSessionContext`, or implicitly when the session is closed.

5.4.3.6 disposeSessionContext

Synopsis:

```
void disposeSessionContext( SessionId contextId)
      throws SessionServiceFault;
```

Dispose the session context specified by argument `contextId`. The operation has no effect if an actual session ID is passed as `contextId`. Disposing a session context with a valid project key does not close the project identified by this key.

5.4.3.7 getSessionContext

Synopsis:

```
SessionContextData? getSessionContext( SessionId contextId)
      throws SessionServiceFault;
```

Get information about the session context specified by argument `contextId`.

If argument `contextId` is a session context ID, the returned instance of `SessionContextData` contains all information stored in the session context (except the session ID).

If argument `contextId` is a session ID, the operation doesn't fail, but doesn't return an instance of `SessionContextData` either.

5.4.3.8 getAllSessionContexts

Synopsis:

```
SessionContext[] getAllSessionContexts( SessionId sessionId)
      throws SessionServiceFault;

 struct SessionContext : SessionContextData {
       SessionId contextId;                           // attribute
 }
```

Get information about all currently existing session contexts of the session specified by argument `sessionId` (which may be one of the session's session context IDs).

5.4.3.9 `updateSessionContext`

Synopsis:

```
void updateSessionContext(    SessionId                contextId,
                              SessionContextData       contextData)
      throws SessionServiceFault;
```

Update the specified session context.

Argument `contextId` must be a session context ID. The operation fails if a session ID is specified.

If field `projectKey` of argument `contextData` is specified, the project key stored in the session context is updated to the specified project key, or reset (becomes unspecified) if the specified project key is the NIL-UUID (may be represented by an empty string).

Replacing the project key in a session context does not close a previously referenced project.

The operation does not ensure that the project key, if specified and not NIL, actually matches the key of one of the session's open projects. However, since the project keys used in the web service APIs are a special kind of V8 UUIDs, attempts to set the project key of the session context to a randomly created UUID will fail.

5.4.3.10 `setLocale`

Synopsis:

```
void setLocale(
            string sessionId,
            string name,
            string timeZone,
            int tzOffset
            )
```

If invoked with a session context ID as first argument (see also §5.4.3.5), the session ID stored in the session context will be used as the actual session ID. So the locale is changed for the whole session including all projects loaded in the session.

Optional parameters 'timeZone' and 'tzOffset' (types string and integer)

TmeZone: if specified and not empty, should be a time-zone accepted by ZoneId.of(String)

If it is valid the time-zone will be used as the session's new time-zone, regardless of whether or not 'tzOffset' has been specified.

If the time-zone is not valid and 'tzOffset' has not been specified then the operation will fail.

If the time-zone is not valid and 'tzOffset' has been specified then the session's new time-zone will be set according to the zone offset.

tzOffset: if specified, must be an integer between -840 and +840, inclusive, regardless of whether or not the value is actually used to set the time zone.

5.4.3.11 `getLocale`

Synopsis:

```
string getLocale(
            string sessionId,
            boolean withTimeZone
                  ) throws SessionServiceFault;
```

Optional parameter 'withTimeZone' (type boolean).

If this parameter is true, the return value consists of the locale name and time-zone ID, separated by a comma.

A `SessionServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.

5.4.3.12 `setSessionProperty`

Synopsis:

```
void setSessionProperty(SessionId sessionId, string name, string value)
        throws SessionServiceFault;
```

This operation is used to set session-specific properties that affect the operation of the Online Configurator. The `name` argument must be set to the property's name, and the value argument to the property's value, converted to a string if numeric.

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. The properties will be changed for the whole session including all projects loaded in the session.

The following properties are supported:

`egr.eai.ws.catalog.preferred_icon_size`

Setting this property affects the size of icons returned by the catalog service. This property can also be set through the `setPreferredIconSize` operation of the `CatalogService`. For more information, see §5.5.3.9.

`egr.eai.ws.basket.material_icon_format`

Setting this property affects the format of material images returned by operations of the basket service. The value must be a possibly empty comma-separated list of file name extensions. White space around file name extensions is ignored. Each file name extension must consist of a dot, followed by a non-empty sequence of ASCII letter, digit or underscore.

For each material image to be returned, the basket web service will iterate over the list of extensions until it finds an image file using the current extension. If the list of extensions is empty, no material images will be returned.

In this version:

Changed default value of session property 'egr.eai.ws.basket.material_icon_format' from '.jpg' to '.png,.svg,.jpg'.

`egr.eai.basket.preserve_price_date`

If set to `true` always preserve price date when reading a BSK/OBX file if it has been written by the same session, thus preserving the price date during copy/paste in the same session. It also ensures suspend/resume preserves the price date in case the default value of session property egr.eai.basket.preserve_price_date ever changes.

If set to `false` the price date is ignored, when articles read from OBK/BSK, or from OBX originating from another session.

A `SessionServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The property name is unknown.
- The format of the property value is invalid (e.g. the value for a property of type integer cannot be parsed as an integer).

- The property value is out of range or otherwise invalid.

5.4.3.13 `getSessionProperty`

Synopsis:

```
string* getSessionProperty(SessionId sessionId, string name)
        throws SessionServiceFault;
```

This operation is used to get the current value of session-specific properties, converted to a string.

See §5.4.3.12 for a list of supported properties. The operation returns *null* if the property name is unknown.

If a property has not been set by the `setSessionProperty` operation, the `getSessionProperty` operation returns the default value.

A `SessionServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.

5.4.3.14 `saveSession`

Synopsis:

```
string saveSession(SessionId sessionId,
                   string uri,
                   SaveSessionOptions* options)
        throws SessionServiceFault;
```

The `saveSession` operation saves the state of the current session into a pCon.basket compatible project (OBK) file. If the `uri` argument is an empty string, the project file is written to the working directory of the current session, and the operation returns an HTTP URL that can be used by the client to download the project file. The URL may be passed to the `loadSession` operation to load the project into the same session. If the project file is no longer needed, the client may use an HTTP DELETE request to delete the project file on the server.

If the `uri` argument is set to a FILE URL, the Online Configurator treats the URL's path as a path into its local file system, and attempts to save the project file under this path. The `egr.eai.server.file_access` property of the session or server configuration file must be properly configured for this to work (§4.2.1).

URI schemes other than "file" are currently not supported.

The `suffix` option is ignored if an URI is specified.

A `SessionServiceFault` is returned if one of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `uri` argument is not empty and the syntax of the URI is invalid.
- The `suffix` option is specified (and the `uri` argument is empty), and the suffix contains a character not allowed in file names, or would result in an otherwise invalid file name (like a file name whose length exceeds the maximum length of file names supported by the underlying file system).
- The `uri` argument is not an empty string and the URI uses a scheme other than "file".
- The FILE URL can not be converted into a path name.
- The server has not been configured to be allowed access to the file referenced by the FILE URL.
- The file already exists and the `overwrite` option has not been set to `true`.
- `legacyCalculationName` is not empty and does not specify the name of a calculation added to the project
- `legacyCalculationName` specifies a calculation added to the project, but the underlying pricing

procedure is not considered eligible for export as a legacy calculation.
- `legacyCalculationName` is empty, the project contains at least one calculation, and the operation is unable to identify a single calculation that is eligible for export as a legacy calculation (eligibility of multiple calculations is reason for failure).
- The Online Configurator encountered some other problem while saving the project file (input/output error, file system full, inconsistent session state, …).

For a pricing procedure to be considered eligible for export, all of the following conditions must be met:

- The line tags `GROSS_SALES_PRICE` (preferred) or `PD_SALES_PRICE` and `NET_VALUE` identify a usable range of lines.
- None one of the line tags `GROSS_PURCHASE_PRICE`, `PD_PURCHASE_PRICE` and `PURCHASE_VALUE` identifies a line, or the identify a usable range of lines (`GROSS_PURCHASE_PRICE` is preferred over `PD_PURCHASE_PRICE`).
- The lines identified by the aforementioned tags must be subtotal lines, or price conditions with line insert mode `Always`.
- The lines between the first and last line of the ranges specified above must be subtotal or text lines (both will be ignored by the export), or condition lines with condition classes `Price` or `PriceModifier`. (Condition class `Tax` and line type `CalculationBreak` cause an error.)

All conditions are exported as item discounts, and the rounding of item discounts is disabled that the total net value of the exported legacy calculation is equal to the net value of the original calculation. For taxes this can not be guaranteed due to limitations of the legacy calculation scheme.

5.4.3.15 `loadSession`

Synopsis:

```
string loadSession(SessionId sessionId,
                   string uri,
                   LoadSessionOptions* options)
        throws SessionServiceFault;

struct LoadSessionOptions {
    string* suffix;
}
```

The `loadSession` operation is used to load a pCon.basket compatible project (OBK) file into the current session.

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. When invoked with a session context ID, the operation sets the project key of the session context to the key of the new or loaded project, but never close the session context's old 'current project', regardless of whether it had been identified by a project key stored in the session context or, in the absence of such a key, was (and remains) the session's current project.

On the other hand, if these operations are invoked with a session ID, and if the new project can be created or loaded successfully, then the new project will always become the session's current project, and the old current project is always closed, regardless of whether there is a session context referencing this project. If there is such a session context, operations invoked for this session context will subsequently fail if they require a valid project until the project key of the session context has been reset or is set to the key of one of the projects currently held open by the session.

If the `suffix` option is not specified, the operation uses ".obk" as the suffix of the project file. If the client sets the suffix option to an empty string, then no suffix is appended to the project file name. Otherwise, the Online Configurator checks whether the suffix starts with a dot, prepending one if it does not, and appends the resulting suffix to the project file name.

The client should use an HTTP PUT request with this URL to upload the project file into the session's working directory. The upload does not actually load the project into the current session.

The client may use an HTTP DELETE request to delete the uploaded file, either before or after it has been loaded into the current session.

If the `uri` argument is not an empty string, it must be one of the following:

- a HTTP URL returned by a previous invocation of the `loadSession` operation on the same session, and uploaded by the client
- a HTTP URL returned by a previous invocation of the `saveSession` operation on the same session
- a FILE URL pointing to a file within the local file system of the Online Configurator

The `loadSession` operation loads the project file referenced by the URI into the current session. If the project file could be loaded successfully, the current state of the session is replaced with the state read from the project file. If there was an error, the session's state is left unchanged.

If the `uri` argument is a FILE URL, The `egr.eai.server.file_access` property of the session or server configuration file must be properly configured (§4.2.1) or the Online Configurator will reject access to the project file.

The `suffix` option is ignored if the `uri` argument is not an empty string.

A `SessionServiceFault` is returned if one of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `uri` argument is not empty and the syntax of the URI is invalid.
- The `suffix` option is specified (and the `uri` argument is empty), and the suffix contains a character not allowed in file names, or would result in an otherwise invalid file name (like a file name whose length exceeds the maximum length of file names supported by the underlying file system).
- The `uri` argument is not an empty string and the URI is neither an URL returned by a previous invocation of the `loadSession` or `saveSession` operations on the same session, nor a FILE URL.
- The FILE URL can not be converted into a path name.
- The server has not been configured to be allowed access to the file referenced by a the FILE URL.
- The Online Configurator encountered some other problem while loading the project file (input/output error, corrupted project file, file system full, …).
- The BSK/OBX stream contains a set-article and the license feature `egr.eai.basket.set_articles` is not enabled.
- After load, the number of items, not counting the top folder and basket sub-article items, is greater than one and the license feature `egr.eai.server.multiple_positions` is not enabled.

5.4.3.16 `setProjectSettings`

Synopsis:

```
void setProjectSettings(SessionId sessionId, ProjectSettings settings)
        throws SessionServiceFault;
```

The `setProjectSettings` operation is used to set individual project settings that will eventually be returned by an invocation of the `getProjectSettings` operation or be written to the project file by the `saveSession` operation.

All fields of the `ProjectSettings` structure (§5.4.1.6) and the referenced structures may be null (§5.1.1.3.1). Each field of the referenced structures (`ProjectData` and `CustomerData`) corresponds to one project settings property.

The client should set only those fields whose corresponding properties it wants to change. If a field is null, the operation keeps the current value for the property.

A `SessionServiceFault` is returned if one of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.

5.4.3.17 getProjectSettings

Synopsis:

```
ProjectSettings getProjectSettings(SessionId sessionId)
        throws SessionServiceFault;
```

The getProjectSettings operation returns the current project settings that have previously been read from a project file by the loadSession operation or set by the setProjectSettings operation.

Although all fields of the ProjectSettings structure (§5.4.1.6) and the referenced structures may be null (§5.1.1.3.1), the operation sets all fields of the ProjectSettings structure and the referenced structures (ProjectData and CustomerData) to non-null values, even if the corresponding project settings properties have not been explicitly set (in this case, the fields are set to an empty string).

A SessionServiceFault is returned if one of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.

5.4.3.18 setProjectAppData

Synopsis:

```
void setProjectAppData(SessionId sessionId, string appKey, string[] data)
        throws SessionServiceFault;
```

The setProjectAppData operation is used to attach application-specific (or client-specific) data to the project. This data can be accessed using the getProjectAppData (§5.4.3.19) operation and is part of the project file saved and loaded by the saveSession (§5.4.3.14) and loadSession (§5.4.3.15) operations. For more information, see §5.3.

A SessionServiceFault is returned if one of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.
- One of the elements of data contains an invalid location path, or the location path is not followed by an equals sign (=).

5.4.3.19 getProjectAppData

Synopsis:

```
string*[] getProjectAppData(SessionId sessionId,
                            string    appKey,
                            string[]  paths)
        throws SessionServiceFault;
```

The getProjectAppData operation is used to fetch application-specific (or client-specific) data attached to the current project which has either been read by the loadSession (§5.4.3.15) operation as part of the project file, or previously attached to the current project by the setProjectAppData (§5.4.3.18) operation. For more information, see §5.3.

A SessionServicFault is returned if one of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.
- One of the elements of data contains an invalid location path.

5.4.3.20 getUploadURL

Synopsis:

```
string getUploadURL(SessionId               sessionId,
                    SessionCacheFileType    fileType,
                    string*                 suffix)
        throws SessionServiceFault;
```

This operation may be used to get an URL that can be used with HTTP PUT to upload a file into the cache directory of the current session. Once the file has been uploaded, the URL can be used with other operations (`loadSession`, `paste` of basket web service) for further processing of the file.

The `fileType` parameter must be passed to the operation and must be a valid session cache file type (as defined by the enum type). The `suffix` parameter is optional. If it is missing (or null in many language bindings), the operation uses the default suffix for the file type ('.obk' for 'Project' and '.obx' for 'CutBuffer'). Otherwise, if the suffix is not empty and does not start with a period ('.'), a period is prepended to the suffix. The suffix is then used as the file name suffix of the path component of the returned URL.

5.4.3.21 importFile

Synopsis:

```
string importFile(SessionId               sessionId,
                  string                  url,
                  string[]                attributes,
                  ImportFileOptions*      options)
    throws SessionServiceFault;
```

The `importFile` operation is used to import a file into the session. Importing a file means to copy the file into the session cache and compute an URI that can be used subsequently to identify the imported file.

The `url` argument must be either an HTTP URL previously returned by the `getUploadURL` operation, or a FILE URL referencing a file within the local file system of the Online Configurator.

In case of a FILE URL, the `egr.eai.server.file_access` property of the session or server configuration file must be properly configured or the Online Configurator will reject access to the file.

The `attributes` argument is a list of attribute strings to be stored together with the file within the session cache. Each attribute string consists of a key and a value, separated by a single '=' (EQUALS SIGN, U+003D). White space around key and value are not stripped. Keys must consist of Basic Latin letters, digits and '_' (LOW LINE, SPACING UNDERSCORE, U+005F) only, and must not start with a digit. The format of values depends on the key.

There are the following well known attributes:

`suffix`

> the file suffix, not including the leading dot (may be empty); If this attribute is not specified, the attribute is added automatically, using the suffix of the supported file. If specified, it takes precedence over the suffix of the imported file. In any case, the suffix is converted to lower case before stored in the session cache.
>
> If the suffix is that of a recognized image file format (see below), or the file is identified as having a recognized image file format, both must match or the operation fails.

`width, height`

> the width and height (in pixels) of raster image files; The value must be a decimal number greater than zero. If one is specified, both must be specified. A client should always specify these attributes for raster image files. If they are not specified, and the file is recognized as a raster image file, the operation tries to set these attributes. If it is unable to do so, the operation fails.

Recognized image file formats and corresponding suffixes are JPEG (suffixes "jpeg" and "jpg"), PNG (suffix "png"), GIF (suffix "gif"), BMP (suffix "bmp") and TIFF (suffixes "tiff" and "tif"). The operation should be able to determine width and height for all recognized image file formats except TIFF.

The `options` argument as well as individual options are optional. If not specified, the default value applies. Right now, the following option is defined:

```
boolean *deleteAfterImport;
```

> If `true`, the file specified by the `url` argument is deleted once it has been successfully imported into the session cache. The default value of this option is `false`.
>
> If the file is to be deleted, but the delete operation fails for some reason, a message is written to the log file but the `importFile` operation succeeds anyway.

If the operation succeeds, the return value is an URI with scheme `imp` that identifies the imported file. The URI may be used with the resolveURIs operation to get an URL that can be used to access the cached file.

The operation fails if

- the specified session ID is not a valid string representation of an UUID or does not represent an open session
- the specified URL is not an HTTP or FILE URL or does not identify an accessible file
- one of the attribute strings does not start with a valid attribute key immediately followed by an EQUAL SIGN
- the file to be imported is empty
- the file is recognized as an raster image file and the `width` and/or `height` attributes are specified and have an invalid value, only one of them has been specified, or none of them has been specified and the operation is unable to determine their values
- the file format is not recognized by the suffix matches one of the suffixes of file formats the operation should be able to recognize
- the file format is recognized but the suffix (either explicitly specified derived from the file name) does not match the file format
- there was some input/output error reading the file or storing it in the session cache

5.4.3.22 `resolveURIs`

Synopsis:

```
string resolveURIs(SessionId              sessionId,
                   string[]               uris,
                   ResolveURIsOptions     options)
    throws SessionServiceFault;
```

The `resolveURIs` operation is used to convert URIs with URI scheme `imp` or `gen` into URLs that can be used to access the files identified by these URIs.

The `uris` argument must be a possibly empty list of strings representing URIs with scheme `imp` or `gen`.

The operation resolves only URIs for files that have been generated by or imported into this session.

If the operation succeeds, the return value is a list of HTTP URLs with the same number of entries as the `uris` argument.

The URL returned at index i of the result list corresponds to the URI at index i of the argument list. If an URI is unknown, the corresponding return value for this URI is an empty string.

The operation fails if

- the specified session ID is not a valid string representation of an UUID or does not represent an open session
- one of the specified URI arguments does not adhere to the URI syntax, is a relative URI, or uses a scheme other than `imp` or `gen`

5.4.3.23 `getSystemInformation`

Synopsis:

```
SystemInformation getSystemInformation(GetSystemInformationOptions *options)
      throws SessionSerivceFault;
```

The operation may be used to obtain various information about this instance of the Online Configurator.

The `options` argument allows restriction of the returned information to the subset that is actually needed. The `options` argument, if present, has one attribute for each element of return type `SystemInformation`. The name of the attribute is the same as the name of the corresponding element. An element will be returned if the `options` argument is missing, or the corresponding attribute has value `true`.

5.4.3.24 `configureSessionLog`

Synopsis:

```
void configureSessionLog(SessionId sessionId,
                         boolean   enable,
                         LogConfig *config)
        throws SessionServiceFault;
```

The operation enables/disables and configures logging of session-specific log messages.

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. The changes will be made for the whole session including all projects loaded in the session.

Log messages generated during the execution of a web-service operation that specifies a session ID may be stored in a session-specific buffer for later retrieval by the client. To enable this feature, `configureSessionLog` must called with parameter `enable` set to true.

If parameter `enable` is true, the optional parameter `config` may be used to adjust some properties of the session log handler and to select a subset of the session-specific log-messages to be stored in the session log buffer.

All parameters are optional. If not specified, their default values (or previous values in case of repeated calls of `configureSessionLog` with `enable` set to true) will be used. A call of `configureSessionLog` with `enable` set to false will revert all previously set parameters to their default vaues.

5.4.3.25 `getSessionLog`

Synopsis:

```
LogData *getSessionLog(SessionId sessionId,
                       boolean   reset)
        throws SessionServiceFault;
```

The operation retrieves and optionally clears the content of the session specific log buffer.

`getSessionLog` returns the log messages accumulated since the last time the session log has been switched from disabled to enabled or the last time `getSessionLog` has been called with parameter `reset` set to true.

If the session log is currently disabled, no log data is returned. Otherwise a `LogData` element is returned, containing the number of discarded (in case of buffer overflow) and suppressed (i.e. filtered out) log messages, as well as a sequence of log records, each one representing a single log message.

5.4.3.26 `SendMessage`

Synopsis:

```
SendMessageResult sendMessage(
                                SessionId sessionId,
                                       ClientMessage[] messages
                                   )
   throws SessionServiceFault;
```

If the specified session ID is not the NIL UUID, the `keepAlive` operation is called first and its result will be returned in the `sessionAlive` field of `SendMessageResult`. (If `sessionId` is the NIL UUID, `sessionAlive` will be `false`.)

The operation calls `SessionManager.sendMessage()` to deliver the messages to the specified targets. Response messages, if any, are returned in field `messages` of `SendMessageResult`

Operation `sendMessage` should not be invoked with a session context ID. If it is, field `sessionAlive` of the return value is always be `false`, even if the corresponding session is alive, and the session context ID is passed as the first argument of method `IPlugin.sendMessage()`, which may confuse plugins.

5.4.3.27 `loadEmptySession`

Synopsis:

```
   Void loadEmptySession(string sessionId)
```

Its effect is basically the same as loading an empty OBK file (an OBK file immediately written after `openSession`), except that certain values like project ID, create date and last-modified date are set to new values.

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. When invoked with a session context ID, the operation sets the project key of the session context to the key of the new or loaded project, but never close the session context's old 'current project', regardless of whether it had been identified by a project key stored in the session context or, in the absence of such a key, was (and remains) the session's current project.

On the other hand, if these operations are invoked with a session ID, and if the new project can be created or loaded successfully, then the new project will always become the session's current project, and the old current project is always closed, regardless of whether there is a session context referencing this project. If there is such a session context, operations invoked for this session context will subsequently fail if they require a valid project until the project key of the session context has been reset or is set to the key of one of the projects currently held open by the session.

5.5 Catalog Service

The catalog web service API presents registered manufacturer catalogs as a set of trees[18] of catalog nodes. Each tree represents a single manufacturer catalog[19]. Trees consist of folder, article and information nodes.

To uniquely identify a catalog node one must either use a catalog path or the pair of catalog ID and catalog node key.

The catalog path is similar to a file system path. It consists of a list of node names, starting with the name of the root node, followed by the name of one of it’s child nodes, an so on. The last name in the list is the name of the node to identify.

When a catalog node is identified by catalog ID and catalog node key, then the catalog ID identifies the manufacturer catalog containing the node, and the catalog node key uniquely identifies the node within this catalog.

The catalog implementation supports XCF and OAS catalogs. It allows mixing of XCF and OAS catalogs within a single data or catalog profile.

18 Strictly speaking, it is a directed acyclic graph of catalog nodes, as a) the edges between nodes are directed (from parent node to child node), and b) some types of nodes (nodes that can’t have outgoing edges) may have multiple incoming edges (i.e. parent nodes).
19 Old-style manufacturer profiles may result in more than one manufacturer catalog.

5.5.1 Type Definitions

5.5.1.1 `ArticleCatalogItem`

Synopsis:

```
object ArticleCatalogItem : CatalogItem {
   string        articlePackageId;
   string        baseArticleNumber;
   VarCodeType   varCodeType;
   string        variantCode;
}
```

`articlePackageId`

This field contains the package ID of the OFML package containing the product data for the article.

`baseArticleNumber`

This field contains the base article number of the article.

`varCodeType`

The type of the variant code stored in an article catalog item.

`variantCode`

This field contains the initial, possibly partial, variant code of the article.

5.5.1.2 `CatalogImage`

Synopsis:

```
struct CatalogImage {
    string      purpose;
    LanguageTag language;
    boolean     highRes;
    int         width;
    int         height;
    string      name;
    URL         url;
}
```

A reference to an image stored in the catalog.

`purpose`

The image purpose. Predefined values are 'Icon', 'SmallIcon', 'Image' and 'Info'. Other values may be reported depending on the catalog data and text purposes specified as part of the lookup options.

In case of an XCF catalog, resources with type 'IT' are mapped to image purpose 'Icon', and resources with type 'IF' are mapped to image purpose 'Image'.

`language`

The language as specified in the catalog data.

`highRes`

The value of the high-resolution flag as specified in the catalog data.

In case of an XCF catalog, the value of 'highRes' is always 'false'.

`width`
`height`

The image width and height as specified in the catalog data.

In case of an XCF catalog, both values are always zero.

`name`

The name of the image as specified in the catalog data, with possibly some cleanup applied.

`url`

The HTTP url that can be used to access the image. May be an empty string if the Online Configurator knows that it wont find the image (a returned URL does not necessarily mean that the Online Configurator will actually find the image when a HTTP request is used to download the image).

5.5.1.3 `CatalogItem`

Synopsis:

```
object CatalogItem {
    string               catalogId;
    string               catalogPackageId;
    string               catalogNodeKey;
    string               name;
    string               manufacturerIds;
    CatalogItemType      type;
    ItemDescriptor[]     descriptors;
    string               label;
    URL                  icon;
    CatalogText[]        texts;
    CatalogImage[]       images;
    CatalogResource[]    resources;
}
```

The `CatalogItem` type is used by the catalog web service to return information about a catalog node to the client. It contains the following fields:

`catalogId`

Depending on the type of data profile used to register the product data (old-style manufacturer profile or new catalog profile) the catalog ID consists of either

- the OFML manufacturer ID, followed by a colon, followed by a non-empty sequence of decimal digits, or
- the brand ID as specified in the catalog profile (often identical to the commercial manufacturer ID), a colon, and the catalog ID as specified in the catalog profile.

The catalog ID uniquely identifies the catalog within one instance of the Online Configurator. The second type of catalog ID, consisting of brand ID and manufacturer-specific catalog ID, is supposed to be an strictly unique identifier for the catalog, i.e. the same catalog ID should never be used for another catalog, including another version of the same catalog.

`catalogPackageId`

Except for root nodes, the `catalogPackageId` field contains the package ID of the OFML package whose catalog data contains the catalog node. For root nodes, this field contains an empty string.

`catalogNodeKey`

The catalog node key is some string used to uniquely identify the catalog node within the manufacturer catalog. Other than that, the actual content of the string is implementation defined. In particular, a client must not try to parse the string to get at the base article number and/or XCF specific variant key.

For root nodes, this field contains an empty string.

There is no guarantee that a specific catalog node key remains valid for a later version of the same catalog (or identifies the same node), although some effort is taken to keep catalog node keys valid as long as the catalog does not change in some incompatible way.

`name`

The name of a catalog node uniquely identifies the node relative to it's parent. Node names are used to form catalog paths. Other than that, the actual content of the string is implementation defined. In particular, a client must not try to parse the string to get at the base article number and/or XCF specific variant key.

There is no guarantee that a specific catalog node name remains valid for a later version of the same catalog (or identifies the same node), although some effort is taken to keep catalog node names valid as long as the catalog does not change in some incompatible way.

`manufacturerIds`

For catalog items representing the root of a manufacturer catalog, field `manufacturerIds` contains a list of commercial manufacturer IDs of all packages referenced by the underlying catalog profile.

(Each manufacturer catalog shows catalog data from all packages referenced by its underlying catalog profile, and each catalog profile is represented by a single manufacturer catalog.)

For catalog profiles constructed from data profiles, the set of referenced packages consists of a subset of packages referenced by the underlying data profile, with the subset constructed as follows:

- All packages referenced by a data profile that contain catalog data are separated into groups, with all packages in a group having the same OFML and commercial manufacturer IDs, and a non-conflicting set of manufacturer names (as taken from the package registration file).
- A catalog profile is constructed for each group, with the set of packages in each group augmented by other packages from the underling data profile reachable by a dependency graph whose edges are determined by DSR keys `depend` and `catalogs`.

  (Note 1: Multiple (or none at all) catalog profiles may be constructed from a data profile, and packages that don't contain catalog data may be referenced by more than one of these catalog profiles.)

  (Note 2: The process of construction of catalog profiles is further complicated by the fact that data profiles for manufacturers of the same unique concern are combined into a single data profile, which is then used to construct the catalog profile(s). The same applies to data profiles for the same unique manufacturer.)

Given that all packages with catalog data in a catalog profile are required to have the same OFML manufacturer ID, and the mapping between OFML and commercial manufacturer IDs is supposed to be bijective, and given the mechanism described above for the construction of catalog profiles from data profiles, all packages with catalog data referenced by a catalog profile have the same commercial manufacturer ID. This ID is considered the primary manufacturer ID of the catalog profile, and returned as the first item in the list of manufacturer IDs for catalog items representing manufacturer catalogs.

`type`

The `type` field contains the type of the catalog node. For more information about node types see section 5.5.1.4.

`label`

The `label` field contains a short text describing the catalog node. It is supposed to be displayed as part of the catalog presented to the user.

`icon`

The `icon` field contains the URL of a small image that should be displayed as part of the catalog presented to the user. If no icon is available, then this field contains an empty string.

In case the installed catalog data support multiple icon sizes, the `setPreferredIconSize` operation (§5.5.3.9) may be used to specify the preferred icon size.

`texts`

A reference to a catalog text.

`images`

A reference to an image.

`resources`

If resource keys were passed to the operation returning this catalog item, this field contains a possibly empty sequence of the catalog node's resources matching the specified resource keys. The order of entries in this sequence is undefined, and their number is not necessarily equal to the number of resource keys.

5.5.1.4 `CatalogItemType`

Synopsis:

```
enum CatalogItemType {
    Undefined,
    CatalogView,
    Folder,
    Article,
    Information,
    Graphics,
    MethodCall,
    MetaPlanning,
    Container
}
```

The `CatalogItemType` specifies the type of a catalog node. The following node types are defined:

`Undefined`

If a catalog contains a node of some type that cannot be mapped to any of the other types, then the `Undefined` type may be used as the type of such nodes. Right now such nodes are ignored, so the client should never see nodes of this type, but should nevertheless be prepared to handle them[20]. Nodes of type `Undefined` may have either child nodes or multiple parent nodes, but not both.

`CatalogView`

OAS has the concept of catalog views. Basically, views are an additional level at the root of the catalog structure. Applications are supposed to represent only one view to the user. Usually this is the default view, but if the catalog defines multiple views, and an application recognizes one of the views as intended for use with that application, then the application will represent this view to the user.

Catalog view nodes will usually have child nodes.

`Folder`

Nodes of type `Folder` are used to group other nodes which are direct or indirect child nodes of the folder node. Obviously, a folder node may have multiple child nodes, but it must have at most[21] one parent node.

`Article`

A node of type `Article` represents an article that can be inserted into the basket structure provided by the basket web service, and can then be configured (if so intended by the product data). Article nodes cannot have child nodes, but may have multiple parent nodes.

`Information`

Information nodes are added to the catalog to represent additional information. Right now, the API of the catalog web service does not allow access to this information, so the client should probably ignore these nodes.

Like article nodes, information nodes may have multiple parent nodes, but no child nodes.

20 The best way to 'handle' a node of type `Undefined` probably is to ignore the node, together with it's possible child nodes.
21 Root folders have no parent node.

`Graphics`

`MethodCall`

`MetaPlanning`

Nodes of these types are not of much interest for the Online Configurator. For more information, see the OAS specification.

`Container`

The container referenced by the catalog item is returned as a resource whose type depends on the container type, and whose value is an URL for the container file[22]. The URL may be used to download the container file, or it may be used directly with operation `pasteContainer`.

The actual type of catalog items returned by operations of the catalog web service depends on the catalog item type as follows:

CatalogItemType	**Return Type**
Undefined	CatalogItem
CatalogView	CatalogItem
Folder	CatalogItem
Article	ArticleCatalogItem
Information	CatalogItem
Graphics	CatalogItem
MethodCall	MethodCallCatalogItem
MetaPlanning	MetaPlanningCatalogItem
Container	CatalogItem

Future versions of the Online Configurator may return specialized types instead of the `CatalogItem` type.

5.5.1.5 CatalogResource

Synopsis:

```
struct CatalogResource {
    LanguageTag language;
    string      type;
    string      value;
    string      url
}
```

Instances of the `CatalogResource` type are used to store selected resources of catalog nodes found by the `lookupArticle` (§5.5.3.5), `getCatalogItem` (§5.5.3.6) and `listCatalogItems` (§5.5.3.7) operations. The resources are stored in the resources field of the `CatalogItem` (§5.5.1.31) type returned by the aforementioned operations. The fields of the `CatalogResource` type are as follows:

`language`

The language as specified in the catalog data.

`type`

The resource type. The resource type starts with either `XCF:` or `OAS:`, depending on whether the resource originates from an XCF or OAS catalog.

Example: The type of the XCF addon resource is `XCF:AD`, and the type of the OAS graphics resource is `OAS:Graphics`.

[22] The only container format supported right now is `PEC (pCon Exchange Container)`, and the corresponding resource type is `XCF:PEC`.

`value`

The resource value.

5.5.1.6 `CatalogText`

Synopsis:

```
struct CatalogText {
    string      purpose;
    LanguageTag language;
    string      text;
}
```

A text stored in the catalog.

`purpose`

The text purpose. Predefined values are `CatTextShort` and `CatTextLong`. Other values may be reported depending on the catalog data and text purposes specified as part of the lookup options.

In case of an XCF catalog, `CatTextShort` is used as the text purpose for text from the text table.

`language`

The language as specified in the catalog data.

`text`

The actual text.

5.5.1.7 `DescriptorType`

Synopsis:

```
enum DescriptorType {
    Undefined,
    Keyword,
    Category,
    Designer,
    Characteristic
}
```

`Specifies the type of a descriptor (see LookupOptions §5.5.1.14).`

The enumerator `Undefined` has no equivalent in OAS. The catalog web service should never return a descriptor with this type. The enumerator `Keyword` corresponds to OAS type `Untyped`.

5.5.1.8 `DisplayMode`

Synopsis:

```
enum DisplayMode {
       Undefined,
       All,
       Planning2D,
       Planning3D,
       Configuration,
       CAD,
       AllVisible,
       AllVisibleBasket
 }
```

The `DisplayMode` is used ((together with the list of item types) to control the visibility of nodes within catalogs presented to the user.

Display modes are a deprecated XCF concept. They are not supported by OAS. Nevertheless, they are not totally useless. Clients of the Online Configurator should usually pass 'Configuration' in case an XCF catalog uses the display mode to control the visibility of article nodes with insert mode 'S' (standard).

The following display modes are defined:

`All`
: The set of visible catalog nodes is not limited by the display mode.

`Planning2D`
: The set of visible catalog nodes is limited to those nodes relevant to 2D planning applications.

`Planning3D`
: The set of visible catalog nodes is limited to those nodes relevant to 3D planning applications.

`Configuration`
: The set of visible catalog nodes is limited to those nodes relevant to applications that operate on isolated article instances, like application that manage an order list of configurable articles.

`CAD`
: The set of visible catalog nodes is limited to those nodes relevant to CAD applications.

`AllVisible`
: Selects all XCF catalog entries whose display mode has at least one of 3D, 2D, ‘planning’ or ‘configuration’ set.

`AllVisibleBasket`
: Behaves like `AllVisible` but omits catalog items whose display mode has ‘not basket’ set.

5.5.1.9 `DisplayText(Catalog)`

Synopsis:

```
struct DisplayText extends string (string lang)
```

5.5.1.10 `GetPackageInfoOptions`

Synopsis:

```
struct GetPackageInfoOptions (
                             boolean? allData,
                             boolean? allLanguages,
                             boolean? useManufacturerConfig,
                             boolean? manufacturerId,
                             boolean? supplierId,
                             boolean? seriesIds,
                             boolean? releaseVersion,
                             boolean? releaseDate,
                             boolean? releaseState,
                             boolean? languages,
                             boolean? type,
                             boolean? category,
                             boolean? dependencies,
                             boolean? catalogs,
                             boolean? features,
                             boolean? seriesType,
                             boolean? priceProfileRegionId,
                             boolean? specialArticleScheme,
                             boolean? globalTradeItemNumber,
                             boolean? maskedCatalogs,
                             boolean? releaseText,
                             boolean? manufacturerName,
                             boolean? programName,
                             boolean? distributorName,
                             boolean? copyright,
                             boolean? Description,
                             boolean? releaseTimestamp
                                   )
```

5.5.1.11 `ItemCategory`

Synopsis:

```
struct ItemCategory {
      string  subcategoryIds
}
```

`ItemCategory` is an extension of ItemDescriptor. Whenever a descriptor with type `Category` is returned, the type of the returned descriptor element is actually `ItemCategory`.

`subCategoryIds`

Is an optional child-element of type 'string'. Sub-categories are returned if the lookup option (§ `§5.5.1.14`) subCategories is true.

5.5.1.12 `ItemDescriptor`

Synopsis:

```
struct ItemDescriptor {
    Synonym[]              synonyms;
    string               id;
    DescriptorType       type;
}
```

`synonyms`

`synonyms` are optional child-elements. `synonyms` are returned for item descriptors if the lookup option (§ `§5.5.1.14`) `synonyms` is true.

Standard synonyms are returned in front of non-standard synonyms. The number of standard synonyms returned for a particular descriptor depends on the catalog data. EAIWS does not try to ensure that there is at least one or no more than one standard synonym.

5.5.1.13 `LanguageTag`

Synopsis:

```
typedef string LanguageTag;
```

Language tags supported by the Online Configurator are a subset of language tags described by RFC 5646.

Language tags returned by the web services interfaces consist of the primary language subtag and optionally the script and region subtags. Es an exception, the undetermined language is represented by an empty string instead of "und".

Language tags accepted by the web service interfaces must conform to the 'langtag' syntax rule defined in RFC 5646. However, only the primary language, script and region subtags are significant. All other subtags are discarded. Furthermore, the primary language, script and region subtags must be registered in the IANA Language Subtag Registry.

5.5.1.14 `LookupOptions`

Synopsis:

```
struct LookupOptions {
    DisplayMode*        displayMode;
    CatalogItemType[]   itemTypes;
    LanguageTag[]       languages;
    string[]            textPurposes;
    string[]            imagePurposes;
    string[]            resourceTypes;
    boolean             descriptors;
    boolean             subCategories;
    boolean             synonyms;
    boolean             resourceValues,
    boolean             resourceURLs
}
```

The lookup options are used as an optional parameter for the `lookupArticle` (§5.5.3.5), `getCatalogItem` (§5.5.3.6) and `listCatalogItems` (§5.5.3.7) operations. All fields of `LookupOptions` are optional too. For all fields, a default value is defined that is used if either no lookup options are passed to the operation, or if the field has no value.

`displayMode`

The display mode restricting the set of catalog nodes returned. Should be left unset by most clients.

Default Value: `Configuration`

`itemTypes`

A set of item types used to restrict the set of catalog nodes returned. The afore mentioned operations won't return a catalog item whose type is not contained in this set.

If the set of item types contains `CatalogView`, then the `listCatalogItems` operations lists the views of an catalog if called with a path consisting of a single element, and the second element in catalog paths represent OAS catalog views. In case of XCF catalogs, there is only one view, always named `DEFAULT`.

Default Value: `['Folder', 'Article', 'Information']`

As a special case, if the only item type specified is 'CatalogView', then the default value plus 'CatalogView' is used instead.

`languages`

A prioritized list of languages to use during text, image and resource lookup.

For each language in the list, the Online Configurator looks if there is at least one matching entry. If so, the entry is returned (or, in case of images, possibly multiple entries are returned). Otherwise, the process is repeated with the next language. As a last resort, the process is repeated with the undetermined language.

Default Value: the effective set of catalog languages

`textPurposes`

The text purposes of the texts to return. A text purpose must be a non-empty sequence of ASCII letter, digit or underscore, not starting with a digit. For predefined text purposes, see the description of `CatalogText`.

For each specified text purpose, the Online Configurator returns at most one `CatalogText` per catalog node.

`imagePurposes`

The image purposes of the images to return. A image purpose must be a non-empty sequence of ASCII letter, digit or underscore, not starting with a digit. For predefined image purposes, see the description of `CatalogImage`.

For each specified image purpose, the Online Configurator may return multiple `CatalogImage`'s per catalog node.

`resourceTypes`

The types of the resources to return. If the resource type does not contain a colon (':'), it is assumed to be an XCF resource type. Otherwise, it must start with `XCF`: or `OAS`:. The rules for the actual resource name are as follows:

`XCF`: A sequence of at least two ASCII upper case letters or digits, starting with a letter `OAS`: A non-empty sequence of ASCII letters, digits or underscore, not starting with a digit

`descriptors`

If set to true, this option enables operations to return additional information. See descriptions of complex types `CatalogItem (§5.5.1.3)`and `ItemCategory (§5.5.1.11)` for more information.

Default: false

`subCategories`

If set to true, this option enables operations to return additional information. See descriptions of complex types `CatalogItem (§5.5.1.3)`,`ItemCategory (§5.5.1.11)`and `ItemDescriptor (§5.5.1.12)` for more information.

Except for the `getItemDescriptors` operation, the default values of this option is `false`. For the `getItemDescriptors` operation, the default values is `true`.

`synonyms`

If set to true, this option enables operations to return additional information. See descriptions of complex types `CatalogItem (§5.5.1.3)`, `ItemCategory (§5.5.1.11)` and `ItemDescriptor (§5.5.1.12)` for more information.

Except for the `getItemDescriptors` operation, the default values of this option is `false`. For the `getItemDescriptors` operation, the default values is `true`.

5.5.1.15 `MaskedCatalog`

Synopsis:

```
struct MaskedCatalog (string moduleKey, Version? version)
```

5.5.1.16 `MetaPlanningCatalogItem`

Synopsis:

```
object MetaPlanningCatalogItem : CatalogItem {
        string    metaPlanningWorkflow;
        string    metaPlanningClass;
        string    metaPlanningArgument;
}
```

5.5.1.17 `MethodCallCatalogItem`

Synopsis:

```
object MethodCallCatalogItem : CatalogItem {
        MethodCallType        methodCallType;
        string                methodCallContext;
        string                methodCall;
}
```

5.5.1.18 MethodCallType

Synopsis:

```
enum MethodCallType {
    Instance,
    Class
}
```

The type of a method call. See the OAS specification for more information.

`Instance`
: The method is called on an OFML instance (object), e.g. an instance which represents an article.

`Class`
: A (static) class method is called.

5.5.1.19 PackageCategory

Synopsis:

```
Restriction: string

enum PackageCategory {
                       Unknown,
                       Furniture,
                       Building
                     }
```

5.5.1.20 PackageDependency

Synopsis:

```
struct PackageDependency (PackageId packageId, Version version)
```

5.5.1.21 PackageInfo

Synopsis:

```
struct PackageInfo (
             string                  packageId,
             string?                 ManufacturerId,
             string?                 supplierId,
             string[]?               SeriesIds,
             string?                 distributionRegion
             Version?                releaseVersion,
             date?                   ReleaseDate,
             dateTime?               releaseTimestamp;
             string?                 releaseState,
             string[]?               languages,
             PackageType?            type,
             PackageCategory?        category,
             PackageDependency[]?    dependencies,
             PackageId[]?            catalogs,
             string[]?               features,
             SeriesType?             seriesType,
             string?                 priceProfileRegionId,
             string?                 specialArticleScheme,
             string?                 globalTradeItemNumber,
             MaskedCatalog[]?        maskedCatalogs,
             DisplayText[]?          releaseText,
             DisplayText[]?          manufacturerName,
             DisplayText[]?         programName,
             DisplayText[]?         distributorName,
             DisplayText[]?         copyright,
             DisplayText[]?         description,
                   )
```

5.5.1.22 `PackageType`

Synopsis:

```
Restriction: string

enum PackageType {
                  Unknown,
                  Foundation,
                  ProductWithCatalog,
                  ProductWithoutCatalog,
                  CatalogOnly,
                  Extension,
                  ACMaterial,
                  User,
                  AttributeSelection
                  }
```

5.5.1.23 `ScoredCatalogItem`

Synopsis:

```
struct ScoredCatalogItem {CatalogIte item, float score}
```

Elements of this type are used by the return value of the search operation to return scored catalog items, where a higher value of `score` signals a better match with the query.

5.5.1.24 `SearchFlag`

Synopsis:

```
enum SearchFlag {
    ViewItems,
    CatalogItems,
    WildcardsInFilter,
    FolderText
}
```

The searchFlags are used by SearchParameterSet. The following flag values are available:

`ViewItems`

If only `ViewItems` has been specified, the search operation only considers structure items in selected views.

`CatalogItems`

If only `CatalogItems` has been specified, the searchoperation only considers catalog items. The set of selected views has no effect.

`WildcardsInFilter:`

The flag `WildcardsInFilter` enables the use of wildcard queries in filter expression (see description of filter expression syntax below).

`FolderText`

Allow operation `searchArticle` to return items if the search expression is found in the name of an ancestor item (folder). I.e., if there is a folder whose name matches the search expression then all items contained within the folder (children, grandchildren, …) will be returned too.

Since the weight of direct matches is 1.0 and the weight of folder matches in only 0.25, items whose own name matches the search expression will usually precede items whose name does not match the search expression but are contained within a folder with a matching name.

5.5.1.25 `SeriesType`

Synopsis:

```
Restriction: string

enum SeriesType {
                  Undefined,
                  GOMeta
                }
```

5.5.1.26 `BasicSearchParameterSet`

Synopsis:

```
struct BasicSearchParameterSet {
    int                     firstHitPosition;
    int                     numberOfHits;
    string[]                catalogIds;
    string[]                views;
    searchFlag[]            flags;
}
```

`firstHitPosition`

Defines the offset of first returned hit. Can be used in combination with `numberOfHits` to implement paged result representation.

`numberOfHits`

Defines the maximum number of hits (i.e. catalog items) returned by the search operation.

`catalogIds`

Identifys the catalogs to search in. The current implementation allows only a single catalog ID.

`views`

These elements are used to determine the set of catalog views (OAS views) to search for catalog items (actually structure items in OAS lingo). The set of selected catalog views is determined as follows:

- If no view has been specified, only the default view is selected.
- If at least one of the specified view names is '*', then all views are selected.
- Otherwise, the set of selected view names is the possibly empty intersection of specified view names and the names of available views.

`flags`

The following flag values are available:

`ViewItems, CatalogItems:`

Depending on the combination of these flags, and the selected views, the search behaves as follows:

If neither `ViewItems` nor `CatalogItems` has been specified, the search operation searches considers structure items in selected views and catalog items not referenced by structure items from selected views.

If only `ViewItems` has been specified, the search operation only considers structure items in selected views.

If only `CatalogItems` has been specified, the searchoperation only considers catalog items. The set of selected views has no effect.

If both `ViewItems` and `CatalogItems` has been specified, the search operation considers view items in selected views and all catalog items. Note that this will most likely result in duplicates, as the result set will usually contain both structure items and the referenced catalog items.

`WildcardsInFilter:`

The flag `WildcardsInFilter` enables the use of wildcard queries in filter expression (see description of filter expression syntax below).

5.5.1.27 `SearchParameterSet`

Synopsis:

```
struct SearchParameterSet : BasicSearchParameterSet{
      string                    query;
      filter                    filter;
}
```

`query`

This element contains the query string entered by the user. If a filter (§5.5.1.31) is specified, the query string may be empty. See below for more information about the query syntax (§5.5.1.30).

`filter`

This element may contain a filter expression to restrict the search to a subset of the catalog items.

The syntax of filter expressions is described in §5.5.1.31.

If both a query and a filter are specified, the filter does not contribute to the computed scores.

5.5.1.28 `SearchArticleParameterSet`

Synopsis:

```
struct SearchArticleParameterSet : BasicSearchParameterSet{
      string                    *brandId;
      string                    *manufacturerId;
      string                    *seriesId;
      string                    *basArticleNumber;
      VarCodeType               *varCodeType;
      string                    *variantCode;
}
```

The complex type `SearchArticleParameterSet` is derived from `BasicSearchParameterSet`.

All fields are optional attributes. The default value (used if the field is not present) is an empty string, except for `varCodeType`, where it is `None`.

5.5.1.29 `Index Construction`

EAIWS constructs one index database for each catalog (the set of catalog packages referenced by a data or catalog profile).

The current implementation constructs the index the first time a search is performed on a catalog, and keeps the index database in memory.

During indexing, one index item (document in Lucene lingo) is created for each structure item or catalog item of the catalog database (the internal catalog database uses a format very similar to the OAS 2.0 format).

However, no index item is created if

- A catalog item has no text ID. A catalog item has a text ID if the text table contains at least one text record for the catalog item.
- A structure item has no text ID and does not reference a catalog item with a text ID.

Each index item has multiple fields. A field consists of field name and text. Depending on the field type (implied by the field name) the text is treated as a single token or split into multiple tokens.

Since the index database uses an inverted index, the tokens are not stored in the index item. Instead, and somewhat simplified, the index stores, for each token encountered, a list of index item IDs of all index items whose fields contain the token and, in case of tokenized text, the position of the token in the text.

The following list describes the field names currently used. Field names starting with an underscore are internal field names not available in filter expressions. They are described here for a better understanding of how queries work.

`ban`

The base article number. Each index item that represents an article contains one instance of this field.

`series`

The commercial series ID. Each index item that represents an article contains one instance of this field unless the series ID stored in the catalog database is empty.

`catpkg`

The catalog package ID. Each index item contains one instance of this field. The catalog package ID has the format '::<manu>::<prog>::<version>/<region>'.

`Artpkg`

The article package ID. Each index item that represents an article contains one instance of this field. The article package ID is the ID of the OFML package that contains the article data. The format of the article package D is '::<manu>::<prog>::<version>/<region>'.

`Descriptor`

An OAS descriptor ID. Each index item for a catalog item, and each index item for a structure item referencing a catalog item, has one instance of this field for each descriptor assigned to the catalog item.

`category`

An OAS category ID. Each index item for a catalog item, and each index item for a structure item referencing a catalog item, has one instance of this field for each category assigned to the catalog item, and for each direct and indirect base category of this category.

`_text@<language>`

A text from the text table. For each text purpose found for a particular structure or catalog item, and each language tag from the set of language tags found anywhere in language fields of the catalog database, the corresponding text is determined and, if not empty, an instance of this field, with the language encoded in the field name and the tokenized text used as the field text, is added to the index item.

`_keyword@<language>`

A synonym from the synonym table. For each descriptor assigned to the catalog item (referenced by the structure item), and for each language tag from the set of language tags found anywhere in language fields of the catalog database, all synonyms for the descriptor matching the language tag are determined, and for each such synonym an instance of this field, with the language encoded in the field name and the tokenized synonym used as the field text, is added to the index item.

Tokenization of text and keyword fields is language dependent. Right now, in most cases the same algorithms are used as provided by the language specific analyzers of Lucene, with one exception in case of German languages.

In general, tokenization first splits the input text using Word Break rules from the Unicode Text Segmentation algorithm. The individual tokens are then converted to lower case. Words that carry no meaning (like 'a' and 'the' in English) are removed then.

Finally, a stemming algorithm is applied to reduce words to their word stem. The resulting tokens are then added to the index. For some languages there are additional steps that transform the tokens.

For German languages a compound word token filter has been added that attempts to split compound words into their parts (like 'Drehstuhl' into 'Dreh' und 'Stuhl', or possibly just 'Stuhl'). This is necessary to make sure that a search for 'Stuhl' also finds 'Drehstuhl'.

5.5.1.30 `Query Syntax`

Note: The following text is a slightly modified copy of the documentation of the 'SimpleQueryParser' class of Lucene.

The current implementation uses the 'SimpleQueryParser' of Lucene to parse the query. The main idea behind this parser is that a person should be able to type whatever they want to represent a query, and the parser will do its best to interpret what to search for no matter how poorly composed the request may be. Tokens are considered to be any of a term, phrase, or subquery for the operations described below. White space including ' ', '\n', '\r' and '\t' and the operators '(', ')', '+', '|' and '"' may be used to delimit tokens.
Any errors in query syntax will be ignored and the parser will attempt to decipher what it can; however, this may mean odd or unexpected results.

Query Operators

'+' specifies AND operation: token1+token2

'|' specifies OR operation: token1|token2

'-' negates a single token: -token0

'"' creates phrases of terms: "term1 term2 ..."

'*' at the end of terms specifies prefix query: term*

'~N' at the end of terms specifies fuzzy query: term~1

'~N' at the end of phrases specifies near query:

"term1 term2"~5

'(' and ')' specifies precedence: token1 + (token2 | token3)

The default operator is OR if no other operator is specified. For example, the following will OR token1 and token2 together:

token1 token2

Normal operator precedence will be simple order from right to left. For example, the following will evaluate token1 OR token2 first, then AND with token3:

token1 | token2 + token3

Escaping

An individual term may contain any possible character with certain characters requiring escaping using a '\'. The characters '+', '|', '"', '(', ')', '"' and '\' will need to be escaped in terms and phrases.

The '-' operator is a special case. On individual terms (not phrases) the first character of a term that is '-' must be escaped; however, any '-' characters beyond the first character do not need to be escaped. For example:

-term1 -- Specifies NOT operation against term1

\-term1 -- Searches for the term -term1.

term-1 -- Searches for the term term-1.

term\-1 -- Searches for the term term-1.

The '*' operator is a special case. On individual terms (not phrases) the last character of a term that is '*' must be escaped; however, any '*' characters before the last character do not need to be escaped:

term1* -- Searches for the prefix term1

term1* -- Searches for the term term1*

term*1 -- Searches for the term term*1

term*1 -- Searches for the term term*1

Searched Fields and their Weights

The query constructed from the query expression searches in the 'ban', '_text@<language>' and '_keyword@<language>' fields, where <language> is the first language from the list of languages specified by the lookup options (or the effective catalog languages if the lookup options to not specify any languages)

that is supported by the catalog database.

The fields are weighted differently so a match found in the 'ban' field is rated higher than a match found in the '_keyword@...' field, which in turn is rated higher than a match found in the '_text@...' field. The weights currently assigned to the fields are as follows:

'ban'	-- 5
'_keyword@...'	-- 2
'_text@...'	-- 1

5.5.1.31 `Filter Expression Syntax`

Character sequences within '<' and '>' are non-terminals. Character sequences within single quotes are regular expressions representing terminals/tokens. The regular expression syntax is that of POSIX extended regular expressions with the following additional character classes:

[:White_Space:] Unicode code points with property White_Space

[:ID_Start:]   Unicode code points with derived core property
ID_Start. For the US-ASCII subset, this are all
upper and lower case letters.

[:ID_Continue:] Unicode code points with derived core property
ID_Continue. For the US-ASCII subset, this are
all upper and lower case letters, the decimal
digits, and the underscore.

White space between tokens is optional and can be omitted unless it is necessary to delimit tokens. This is the case if

- the first token ends and the second token starts with a character belonging to the [:ID_Continue:] character class, or
- to terminate <non-white-space-character-seq> (except at the end of the filter expression).

Furthermore, there must be no white space between <field-name> and <term-text> (in other words, there must be no white space before and after the delimiting colon).

```
<filter-expression> ::= <expression>

<expression>          ::= <or-expression>
```

```
<or-expression>       ::= <and-expression>
                        | <or-expression> 'OR' <and-expression>
<and-expression>      ::= <unary-expression>
                        | <and-expression> 'AND' <unary-expression>
<unary-expression>    ::= <primary-expression>
                        | 'NOT' <unary-expression>
<primary-expression> ::= <term>
                        | '\(' <expression> '\)'
<term>                ::= <field-name> ':' <term-text>
<field-name>          ::= '[[:ID_Start:][:ID_Continue:]*]'
```

(informal: A Unicode identifier (see UAX#31, definition D1.), starting with an [:ID_Start:] character followed by zero or more [:ID_Continue:] characters.)

```
<term-text>           ::= <string-literal>
                        | <non-white-space-character-seq>
<string-literal>      ::= '"[^"]*(""[^"]*)*"'
                        | '\'[^\']*(\'\'[^\']*)*\''
```

(informal: A sequence of characters within quote characters, where the sequence of characters does not contain an odd number of consecutive quote characters. The quote character is either a double quote or a single quote.)

```
<non-white-space-character-seq> ::= '[^[:White_Space:]]+'
```

(informal: A non-empty sequence of non-white-space characters.)

'OR', 'AND' and 'NOT' represent the usual logical operations. The operands are boolean values, and the result is a boolean value.

'NOT' has the highest precedence, 'OR' the lowest. Expressions enclosed in parenthesis can be used to override the precedence rules. Thus, 'foo:a OR NOT bar:b AND baz:c' is the same as 'foo:a OR ((NOT bar:b) AND baz:c)'.

The result of the <term> sub-expression is a boolean value. For a particular index item the result of a <term> sub-expression is true if and only if the index item has a field with the specified name and its text matches the specified term text.

Filter expressions may use any field name allowed by the syntax above, no matter whether the field name is actually used by the index. See above (Index Construction) for a description of field names used by the index.

If the 'WildcardsInFilter' flag is not set, the term text matches if it is exactly identical (including case).

If the 'WildcardsInFilter' flag is set, the characters '*', '?' and '\' have special meaning. The backslash is used as a escape character. The escape character must be followed by one of these special characters and removes their special meaning. The '?' character matches any single character, and the '*' character matches any (possibly empty) sequence of characters.

Note that the use of wildcards may result in slow queries, as the search engine must iterate over many terms. In order to prevent extremely slow queries, the term text not start with '*'.

Synonym

Synopsis:

```
struct Synonym {
        string              name;
        string              language;
        boolean             isStandard;
}
```

`language`

`language` is a language tag representing the language of the synonym (e.g. 'de-DE') or an empty string in case of the undetermined language.

`isStandard`

`isStandard`, if true, identifies this synonym as the standard synonym that should be preferred over other synonyms of the same descriptor.

`name`

The element `name` contains the name of the synonym (i.e. the actual synonym).

5.5.1.32 `TopCatalogItems`

Synopsis:

```
struct TopCatalogItems {
        ScoredCatalogItem[]         scoredItems;
        int                         totalHits;
                      }
```

An element of this type is used as the return value of the search operation to return the total number of matching catalog items and (a subset of) the matching catalog items.

5.5.1.33 `VarCodeType`

Synopsis:

```
enum VarCodeType {
    None,
    Manufacturer,
    OFML
}
```

The type of the variant code stored in an article catalog item. See the OAS specification for more information.

`None`

The article should be created in its initial configuration.

`Manufacturer`

The variant code of this article is based on a manufacturer defined scheme.

`OFML`

`The variant code is based on a manufacturer neutral scheme. It corresponds to the predefined OCD code scheme KeyValueList.`

5.5.1.34 `SearchResourceParameterSet`

Synopsis:

```
SearchResourceParameterSet extends BasicSearchParameterSet (
                              string value
                              string catalogPackageId,
                              string language,
                              string resourceType
                              )
```

5.5.2 Faults

5.5.2.1 CatalogServiceFault

Synopsis:

```
    struct CatalogServiceFault {
        string* message;
    }
```

A `CatalogServiceFault` is returned by operations of the catalog service in case of errors the Online Configurator was prepared to detect and deal with (i.e. to recover cleanly).

5.5.3 Operations

5.5.3.1 getPackageInfo

Synopsis:

```
    PackageInfo[] getPackageInfo(
                                             SessionId sessionId,
                                             string[] manuIds,
                                             PackageId[] packageIds,
                              GetPackageInfoOptions? options
)
    throws CatalogServiceFault;
```

Arguments manuIds and packageIds must be possibly empty sequences of valid OFML manufacturer IDs and package IDs. Invalid IDs cause the operation to terminated with a CatalogServiceFault.

If both sets of IDs are empty the operation returns information about all registered packages. Otherwise it returns information about all packages whose OFML manufacturer ID matches one of the given OFML manufacturer IDs or whose package ID partially matches one of the given package IDs. Two package IDs match partially if the fields present in both package IDs are equal.

The options argument controls what data is returned for selected packages. All fields of GetPackageInfoOptions are optional. If no instance of GetPackageInfoOptions is passed to getPackageInfo the operation behaves as if an empty instance has been passed, i.e. an instance with all fields unspecified.

The following options are supported by operation getPackageInfo:

allData - Controls the default value of all other options except useManufacturerConfig. The default value of allData itself is false.

allLanguages - If true, all available translations are returned. If false, only one translation is returned. The returned translation depends on the current list of effective project languages. The lang field of returned instances of DisplayText is not present, i.e. there is no information which translation has been selected.

useManufacturerConfig - Controls whether or returned manufacturer name, distributor name and copyright may originate from the manufacturer configuration file for the package's manufacturer. The default value of this option is false.

If useManufacturerConfig is true and allLanguages is false, the operation tries to fetch the name/text from the manufacturer configuration if no non-empty text is found for the package.

If both useManufacturerConfig and allLanguages are true, the language-to-text mapping of the package is augmented with entries from the manufacturer configuration's language-to-text mapping (package entries take precedence).

All other options control whether corresponding fields (i.e. fields with the same name) are present in returned instances of PackageInfo.

With a few exceptions field names of PackageInfo are based on the names of DSR keys, with underline followed by a lower case letter replaced by the corresponding upper case letter. Exceptions are:

packageId - There is no DSR key package_id. Instead, the package ID is composed of the values of DSR keys manufacturer, program, version, and distribution_region. Consequently, there are no fields in PackageInfo that correspond to these DSR keys.

seriesIds - The corresponding DSR key is program_id.

dependencies - The corresponding DSR key is depend.

priceProfileRegionId - The corresponding DSR key is ppr_region_id.

globalTradeItemNumber - The corresponding DSR key is gtin_id.

Sequence fields are encoded as a single element whose name is equal to the field's name. The element contains one XML element for each sequence element. The nested XML element's name is the singular form of the field name, except for sequences of type DisplayText[], which use text.

5.5.3.2 `setLanguages`

Synopsis:

```
void setLanguages(
                   SessionId sessionId,
                   string[] languages,
                   SetLanguagesMode mode
                   )
         throws CatalogServiceFault;
```

This operation sets the language list of the catalog service (the catalog languages) to the specified, possibly empty list of ISO 639 alpha-2 language codes. It then augments[23] this list with the language of the current locale to build an effective list of catalog languages. The effective list of catalog languages is updated after each invocation of the session service's `setLocale` operation to reflect possible changes to the locale language.

Natural language texts are returned in the first language from the list of effective catalog languages supported by the catalog database. In case the catalog database does not support any of the listed languages then the Online Configurator returns the text in the first language found.

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. The languages will be changed for the whole session including all projects loaded in the session.

A `CatalogServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `languages` parameter contains an invalid language code[24].

23 The locale language is appended to the list if it is not already part of the list.
24 A language code is accepted as long as it consists of two lower case ASCII letters. It must not necessarily be defined by ISO 639.

5.5.3.3 `getLanguages`

Synopsis:

```
string[] getLanguages(SessionId sessionId) throws CatalogServiceFault;
```

This operation queries the current list of catalog languages as set by the `setLanguages` operation. If the `setLanguages` operation has not been invoked yet then the list of catalog languages is empty.

A `CatalogServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.

5.5.3.4 `lookupArticle`

The operation `lookupArticle` has been deprecated. Clients should use the new operation `searchArticle` instead.

Synopsis:

```
CatalogItem[] lookupArticle(
                     string sessionId,
                     string manuId,
                     string seriesId,
                     string baseArtNr,
                     string varCode,
                     LookupOptions options
```

5.5.3.5 `searchArticle`

Synopsis:

```
TopCatalogItems searchArticle(SessionId                  sessionId,
                              SearchArticleParameterSet  search,
                              LookupOptions              options)
        throws CatalogServiceFault;
```

The operation searches a set of catalogs for articles items matching the given manufacturer and series IDs and base article number.

The return value consists of the total number of hits found for the specified search parameters, and a sequence of scored catalog nodes, sorted in descending order according to the score.

If no catalog ID is specified and the brand and manufacturer Ids are empty, or the base article number is empty, the total number of hits is zero.

Otherwise, the operation determines the set of catalogs to search as follows: If `search.catalogIds` contains at least one catalog ID, these catalogs are used. Otherwise, all catalogs matching the brand and manufacturer IDs, if specified, are used.

The operation then iterates over all catalogs, looking for all catalog items matching the specified base article number. The list of matching items is then filtered as follows:

- If the catalog item contains a non-empty manufacturer ID (OAS), and the search parameters contain a different non-empty manufacturer ID, the item is ignored.
- If the search parameters contain a non-empty series ID and the catalog item contains a non-empty series ID (OAS), the item is ignored if both IDs are not equal.
- If the search parameters contain a non-empty series ID and the catalog item contains no (or an empty) series ID, the catalog item is ignored if the article package identified by the catalog item (the package containing the product data of the article) does not contain articles of the specified series (according to the DSR file).

- If the visibility flags of the catalog item are not compatible with the display mode specified by `options.displayMode`, the catalog item is ignored.
- Then, each remaining catalog item may be replaced, depending on the flags `ViewItems` and `CatalogItems` in `search.flags`, and the views specified in `search.views`, as follows:
- If `ViewItems` is set, or `CatalogItems` is not set:

  All view items referencing the catalog item are determined. (view items are (leaf) nodes of a catalog tree/view). There may be multiple view items, either because the catalog item is referenced from multiple locations in the same catalog tree/view, or because there are multiple catalog views (OAS). It is also possible that there are no view items at all if the catalog item is not referenced by any catalog view.

  Then, if `search.views` is not empty, the name of the view containing each view item is determined, and the view item is ignored if the view name is not found in `search.views`.

  If neither `ViewItems` nor `CatalogItems` is set:

  If the list of view items determined above is non-empty, the catalog item is replaced by the view items. Otherwise, the catalog item is retained.
- If only `ViewItems` is set:

  The catalog item is always replaced by the (possibly empty) list of view items.
- If only `CatItems` is set:

  No replacement takes place.
- If both `ViewItems` and `CatalogItems` are set:

  The possibly empty list of view items is inserted in front of the catalog item. The catalog item is always retained.

Then, for each view and catalog item remaining, a score is computed based on the comparison of the variant code of the item and the variant code given in `search.variantCode`. The exact algorithm used to compare both variant codes is unspecified and subject to change.

Once the scores have been computed, the item list is sorted according to the scores in descending order.

Finally, the operation returns the total number of items, and a sub-sequence of the sorted list of items, starting with the item whose index is given by `search.firstHitPosition` (default value is zero). The maximum number of items returned is given by `search.numberOfHits` (default value is ten).

5.5.3.6 `getCatalogItem`

Synopsis:

```
CatalogItem getCatalogItem(SessionId              sessionId,
                           string[]               path,
                           LookupOptions[]        options)
           throws CatalogServiceFault;
```

The `getCatalogItem` operation returns information about the catalog item represented by the `path` parameter. The path must be built from catalog node names, which are returned by catalog service operations as the value of the `name` field of the `CatalogItem` structure.

The `sessionId` parameter must identify an open session. It is used to reference a session configuration which determines the set of available manufacturer catalogs and the language used for human readable texts returned by operations of the catalog service.

The `path` references the desired catalog item. If a path consists of *n* catalog item names cin_1 to cin_n, then the path references the catalog item with name cin_n that is a child item of the item referenced by the path consisting of the catalog item names *cin*$_1$ to *cin*$_{n-1}$. A path consisting of a single catalog item name references the root item[25] of a manufacturer catalog.

The path may reference a catalog item of any type.

The `LookupOptions` argument is used to select the resources to be returned in the returned `CatalogItem` (§5.5.1.35.5.1.31). See §5.5.1.14 for more information.

A `CatalogServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `path` parameter is empty.
- The first element of the `path` parameter does not identify a manufacturer catalog registered for the session identified by the `sessionId` parameter.
- The path does not reference a catalog node.
- At least one node traversed by the path, or the final node referenced by the path, not visible according to the `displMode` and `insModes` parameters.
- There was an error accessing the catalog database.

5.5.3.7 `listCatalogItems`

Synopsis:

```
CatalogItem[] listCatalogItems(SessionId            sessionId,
                               string[]             path,
                               LookupOptions[]      options)
         throws CatalogServiceFault;
```

The `listCatalogItems` operation returns information about all child items of the catalog item represented by the `path` parameter. The path must be built from catalog node names, which are returned by catalog service operations as the value of the `name` field of the `CatalogItem` structure.

The `sessionId` parameter must identify an open session. It is used to reference a session configuration which determines the set of available manufacturer catalogs and the language used for human readable texts returned by operations of the catalog service.

The `path` references the catalog item whose child items will be returned. If a path consists of *n* catalog item names *cin*$_1$ to *cin*$_n$, then the path references the catalog item with name *cin*$_n$ that is a child item of the item referenced by the path consisting of the catalog item names *cin*$_1$ to *cin*$_{n-1}$. A path consisting of a single catalog item name references the root item[26] of a manufacturer catalog. An empty path does not actually reference a catalog item, but can be used to list all the catalog items representing manufacturer catalogs.

The path may reference a catalog item of any type.

The `LookupOptions` argument is used to select the resources to be returned in the `resources` field of returned instances of `CatalogItem` (§5.5.1.31). See §Fehler: Verweis nicht gefunden for more information.

The operation returns a sequence of catalog items, where each catalog item represents a visible child of the catalog item referenced by the path. If the referenced catalog item has no child items, then the sequence of returned catalog items is empty. This may be the case if

- the path references an empty folder,
- the path references a non-empty folder with no visible catalog items (all child items are hidden due to the `displMode` and `insModes` parameters),
- the path references an article or information item, which by definition does not have child items.

25 With the current implementation, one manufacturer catalog consists of one or more XCF catalogs. The root item of the manufacturer catalog is the parent of the root items of the XCF catalogs combined by this manufacturer catalog.

26 With the current implementation, one manufacturer catalog consists of one or more XCF catalogs. The root item of the manufacturer catalog is the parent of the root items of the XCF catalogs combined by this manufacturer catalog.

A `CatalogServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The first element of a non-empty `path` parameter does not identify a manufacturer catalog registered for the session identified by the `sessionId` parameter.
- The path does not reference a catalog node.
- At least one node traversed by the path, or the final node referenced by the path, not visible according to the `displMode` and `insModes` parameters.
- There was an error accessing the catalog database.

5.5.3.8 `searchCatalogItems`

Synopsis:

```
TopCatalogItems searchCatalogItems(SessionId            sessionId,
                                   SearchParameterSet   search,
                                   LookupOptions       *options)
throws CatalogServiceFault;
```

The operation combines the search parameters specified by the `search` argument with the languages, display mode and item types specified with the optional `options` argument to construct a query that is evaluated against the search index of the catalog identified by the catalog ID specified as part of the search parameter set.

The default set of languages is equal to the set of effective catalog languages. The default display mode is `Configuration`, and the default item types are `Folder`, `Article` and `Information`.

The result of the operation consists of the total number of catalog items matching the query, and a scored list of catalog items, with better matches coming first.

Note that the notion of catalog items used by the web service is not the same as the notion of catalog items used by OAS. Web service catalog items represent both OAS structure and catalog items. In case of XCF, there is no distinction between structure and catalog items.

It is only possible to search ofer multiple catalogs as long as the brands of all catalogs (usually equal to the OFML manufacturer name) belong to the same concern.

5.5.3.9 `setPreferredIconSize`

Synopsis:

```
void setPreferredIconSize(SessionId sessionId, int size)
        throws CatalogServiceFault;
```

The `setPreferredIconSize` operation can be used to tell the Online Configurator about the preferred icon size of the client. The Online Configurator interprets the `size` parameter as a hint. There is no guarantee that the icon URL returned as part of the `CatalogItem` structure references an image of the specified size.

The current implementation of the Online Configurator interprets the size parameter as follows: If the preferred icon size has not been explicitly set, or has been set to zero, then the base name of the image directory is `image`. Otherwise (if the preferred icon size has been set to a positive value), the name of the icon directory is the concatenation of `image`, the dot character (`.`), and the requested icon size, formatted as a decimal integer consisting of at least four decimal digits (the necessary number of zeros (`0`) is prepended if the preferred icon size is less than one thousand).

Furthermore, if `$data` is the data directory of the catalog package, *manu*, *prog*, *DR*, and *n* are the OFML manufacturer, program, distribution region and major version number of the catalog package, and `$image` is the base name of the image directory as selected above, then the Online Configurator uses `$data/`*manu*`/`*prog*`/`*DR*`/`*n*`/$image` as the directory containing the image files for catalog items of this catalog package.

If this image directory exists, then the Online Configurator returns an icon URL as part of the `CatalogItem` structure that references the icon file (whose base name is read from the catalog database) within the image directory. If the image directory does not exist[27], the Online Configurator returns an empty string instead.

***Note:** By convention, the maximum of width and height in pixels of each icon image stored in an image directory for a particular preferred icon size should be equal to the preferred icon size.*

A `CatalogServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `size` parameter is negative.

5.5.3.10 `getPreferredIconSize`

Synopsis:

```
int getPreferredIconSize(SessionId sessionId)
        throws CatalogServiceFault;
```

The `getPreferredIconSize` operation returns the preferred icon size assumed by the Online Configurator. The operation returns zero if the preferred icon size has not been previously set by an successful invocation of the `setPreferredIconSize` operation. Otherwise, the operation returns the value of the `size` argument from the last successful invocation of the `setPreferredIconSize` operation for the same session.

A `CatalogServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.

5.5.3.11 `getDescriptorIds`

Synopsis:

```
String[] getDescriptorIds(SessionId          sessionId,
                          string             catalogId,
                         DescriptorType      descrType)
        throws CatalogServiceFault;
```

The operation returns the IDs of descriptors of the given type found in the catalog database of the catalog specified by the given catalog ID. If `descrType` is `Category` then only the IDs of top categories are returned. For all other descriptor types (except `Undefined`, which results in a fault) the IDs of all descriptors of this type are returned.

5.5.3.12 `getItemDescriptors`

Synopsis:

```
ItemDescriptor[] getItemDescriptors(SessionId         sessionId,
                                    string            catalogId,
                                    string[]          descrIds,
                                    LookupOptions     *options)
           throws CatalogServiceFault;
```

The operation returns information about the descriptors identified by the descriptor IDs listed with the `descrIds` parameter. If one of these descriptor IDs does not identify a descriptor, the operation fails.

Most lookup options, if specified, are ignored. The only lookup options that are used are `languages`, `subCategories` and `synonyms`.

[27] Whether or not the icon file itself exists has no effect.

The `languages` option may be used to select the language of returned synonyms. Synonyms are returned if the `synonyms` option is `true`. Sub-categories of categories are returned if the `subCategories` option is true.

Deviating from other operations, the default value of the `subCategories` and `synonyms` options is true.

While the current implementation returns the descriptor information in the order of the descriptor IDs passed to the operation, this behavior should not be taken for granted.

Note that the actual type returned for category descriptors is `ItemCategory`, which is an extension of (derived from) `ItemDescriptor`.

5.5.3.13 `getCatalogPath`

Synopsis:

```
String[] getCatalogPath(SessionId        sessionId,
                        string           catalogId,
                        string           catNodeKey)
throws CatalogServiceFault;
```

Get the catalog path for a pair of catalog ID and node key.

The operation returns an empty path if the specified catalog does not contain a structure item with the specified node key. This may happen for node keys returned by the catalog search for non-folder items as such items may or may not be referenced from the catalog structure (tree).

The operation fails with a CatalogServiceFault if

- If the session ID is not a valid UUID, there is no session with the given UUID or the session is currently suspended and could not be resumed.
- The catalog ID is not a syntactically correct catalog ID or there is no catalog with the given ID.
- The catalog node key is not syntactically correct.
- There was an error accessing the catalog database.

5.5.3.14 `searchResource`

Synopsis:

```
TopCatalogItems searchResource(
                                  string sessionId,
                                  SearchResourceParameterSet search,
                                  LookupOptions options
                                  )
```

5.6 Basket Service

5.6.1 Type Definitions

5.6.1.1 `AddStateCode`

Synopsis:

```
struct AddStateCode extends string (
                                  string domain
                                      )
```

5.6.1.2 `ArticleDescription`

Synopsis:

```
struct ArticleClassification (
                                string        system,
                                string?       Qualifier,
                                string        classId,
                                DisplayText[]? description
                             )
```

5.6.1.3 `ArticleDescription`

Synopsis:

```
struct ArticleDescription (
                                DisplayText[] shortText,
                                DisplayText[] longText,
                                DisplayText[] featureText
                          )
```

5.6.1.4 `AttachmentMode`

Synopsis:

```
enum AttachmentMode {
                   None,
                   Item,
                   Composite,
                   TryComposite
                    }
```

`None`

The newly inserted article is not attached to an existing composite article, i.e. it is inserted as a stand-alone article. This is the traditional and default behavior.

`Composite`

Insert the new OFML article into a composite article. The fatherId passed to the insertOFMLArticle operation must identify a basket article item (i.e. an item with item type Article, Aggregate or PartialPlanning). The insert procedure determines a list of basket article items to consider as parents (in the composite article hierarchy) for the new article item, consisting of all items from the specified basket article item up to the root of the composite article hierarchy, both inclusive, in that order. It creates the new article as a sub-article of the first article item from the list whose item type is either Aggregate or PartialPlanning, and that accepts the new article as its sub-article (i.e. checkAdd() of the corresponding OFML article returned an insert position). The operation fails if no suitable parent article has been found.

`Item`

Like Composite, but only try the first element of the list of possible parents (if the item type of the first element, i.e. the item specified by fatherId, is Article, the insert operation will always fail).

`TryComposite`

Like Composite, but (try to) insert the new article as a stand-alone article if it can not be inserted as part of the composite article.

5.6.1.5 `BItemId`

Synopsis:

```
typedef UUID BItemId;
```

The basket service maintains a tree structure of basket items for each session. Each item, that is node, of the tree is identified by a basket item ID. Basket item IDs are UUIDs. The Online Configurator web services use strings to pass UUIDs between client and server (§5.2.1).

5.6.1.6 `BasketConfig`

Synopsis:

```
struct BasketConfig (
                        string[] viewIds,
                        string[] columnIds
                        string defaultView,
                        boolean columnsEditable,
                        string currency,
                        boolean currencyEditable,
                        date defaultPriceDate
)
```

`defaultView`

The ID of the default view. The default view is the view that is initially displayed by P-BK after the project has been loaded.

`columnsEditable`

A boolean value that indicates whether or not columns may be added to or removed from the current basket. For now the value of this attribute will always be true.

`currency`

The currency used by the basket. This is the same currency as returned by operation getCurrency.

`currencyEditable`

A boolean value that indicates whether or not the currency of the current basket can be changed[1].

`viewIds`

A list of IDs of views of the current basket. This list should always contain the ID of the standard view (5726009a-756d-11d6-9c21-00e029099a4b). The standard view uses display mode Planning and set-article mode Expand. It contains the following columns, in that order: position number, manufacturer, series, article number, description, quantity, single net price, total net price.

`columnIds`

A list of IDs of columns that have been added to the current basket. This list should always contain the following list of standard column IDs[2]:

69ec3fa0-795a-11d6-9c21-00e029099a4b	manufacturer
6d302258-795a-11d6-9c21-00e029099a4b	series
71803794-795a-11d6-9c21-00e029099a4b	article number
745802e4-795a-11d6-9c21-00e029099a4b	description
76eda34c-795a-11d6-9c21-00e029099a4b	quantity
7d21a60a-795a-11d6-9c21-00e029099a4b	single purchase price
7fe484fc-795a-11d6-9c21-00e029099a4b	total purchase price
82efa014-795a-11d6-9c21-00e029099a4b	single sales price
8541eb1a-795a-11d6-9c21-00e029099a4b	total sales price
8831eeec-795a-11d6-9c21-00e029099a4b	single net price
8b51fbe4-795a-11d6-9c21-00e029099a4b	total net price
8dcf0592-795a-11d6-9c21-00e029099a4b	single gross price
90895bc0-795a-11d6-9c21-00e029099a4b	total gross price
81d12edc-853a-11d6-9c21-00e029099a4b	position number
9c9ea8ea-20ce-11d7-9c21-00e029099a4b	catalog image
6a66160c-da62-11d8-b9d6-00e081513ada	generated image
73bd68f4-da62-11d8-b9d6-00e081513ada	article image

5.6.1.7 BasketItemType

Synopsis:

```
enum BasketItemType {
    Undefined,
    Folder,
    Article,
    Aggregate,
    PartialPlanning,
    UserArticle,
    SetArticle,
    OCDArticle,
    Text
}
```

The `BasketItemType` enumeration is used to identify the type of each basket item. Each basket item has one and only one type. The following enumeration values are defined:

`Undefined`
: This value is reserved for occasions where the item type is unknown or undefined. An basket item should never have this type.

`Folder`
: A basket item of type `Folder` can be used to group other items (the children).

`Article`
: A basket item of type `Article` represents an ordinary OFML article that cannot have sub-articles, but can itself be a sub-article of an aggregate or partial planning.

`Aggregate`
: A basket item of type `Aggregate` represents an OFML composite article that can have sub-articles and can itself be a sub-article of another aggregate or partial planning. An aggregate usually starts out as a single article without sub-articles, but allows sub-articles to be added though modification of the aggregate's properties.

`PartialPlanning`
: Technically, a basket item of type `PartialPlanning` represents an OFML composite article similar to an aggregate. However, a partial planning usually does not have an article number or a price and therefore should be treated by the application more like a folder than an article. A partial planning usually starts out with a single sub-article and allows the addition of further sub-articles through modification of the partial planning's properties.

`UserArticle`
: An user article is the most primitive kind of article position. It cannot have sub-articles, nor can it be a sub-article of an aggregate or partial planning. It is not backed by any kind of product data. Instead, all properties of an user article (manufacturer and series IDs, article numbers, article texts, price, …) must be set by the user (or another external agent) through the `setItemProperties` (§5.6.3.33) operation.

`SetArticle`
: A set article is a virtual article which encapsulates a set of real articles. Depending on the view, the articles contained within a set article may be hidden. Regarding its properties, a set article behaves more or less like an user article, except that the the default values of some properties (in particular the price) depend on the articles contained within the set article.

`OCDArticle`
: A basket item of type `OCDArticle` represents an article that only requires the OCD engine instead of the full OFML runtime environment. An OCD article cannot have sub-articles, nor can it be the sub-article of some form of composite article.

`Text`

A basket item of type `Text` can hold text information.

5.6.1.8 `BasketItem`

Synopsis:

```
struct BasketItem {
    BItemId             itemId;
    BasketItemType      itemType;
    string              label;
    BItemId[]           subItemIds;
    BItemId             mainArticleId;
    BItemId[]           subArticleIds;
    BItemId[]           setArticlePartIds;
    BItemId             setArticleId;
    Base64URL           geometryId;
    string              positionNumber;
    BItemId[]           basketItemIds;
    BasketItem[]        basketItems;
}
```

Instances of `BasketItem` are returned by the `getAllItems` operation (§5.6.3.13) to provide basic (mainly structural) information about the basket item. The structure has the following fields:

`itemId`

This field holds the UUID identifying the item.

`itemType`

This field holds the type of this item. See §5.6.1.7 for a description of possible item types.

`label`

Depending on the type of the item, this field either holds the label of a folder or the short text of an article. The label of the top folder is empty because the root folder itself is not meant to be displayed by the application.

`subItemIds`

This field is an possibly empty list containing the identifiers of all child items of this item. This is the same list as returned by the `getSubItemIds` operation when invoked for the item identifier stored in the `itemId` field.

`mainArticleId`

If this item is a sub-article in a hierarchy of composite articles, including the trivial case of a single composite article, this field holds the identifier of the topmost aggregate or partial planning. In all other cases this fields holds the NIL-UUID.

The field `mainArticleId` is not returned if the instance of `BasketItem` represents a view item.

`subArticleIds`

In case of a composite article (aggregate, partial planning), this field holds the possibly empty list of IDs of direct sub-article items.

No sub-article item IDs are returned if the instance of `BasketItem` represents a view item.

`setArticleId`

This field is present in return values of operations `getAllItems` and `pasteContainer` if option `setArticleIds` of these operations is `true`. If the basket item is part of a set-article, the attribute contains the ID of the set-article. Otherwise it contains the NIL-UUID.

No set-article item IDs are returned if the instance of `BasketItem` represents a view item.

`setArticlePartIds`

This field is used in return values of operations `getAllItems` and `pasteContainer` if option `setArticleIds` of these operations is `true` and the basket item is a set-article item. The field contains the basket item IDs of directly referenced parts of the set-article (sub-article items of composite articles are not listed, only the main article).

No set-article part IDs are returned if the instance of `BasketItem` represents a view item.

`GeometryId`

This field is used in return values of operations `getAllItems` and `pasteContainer` if option `geometryIds` of these operations is `true`, the corresponding basket item represents an OFML article item, and the geometry ID is stored in the basket item[28]. If present, the field contains the non-empty Base64URL-encoded geometry ID.

The geometry ID returned as part of type `BasketItem` is equal to the geometry ID returned by operation `getItemProperties`. It is supposed to be used to correlate basket items and geometry nodes in GFJ files.

No geometry IDs are returned if the instance of `BasketItem` represents a view item.

`positionNumber`

This field is present if and only if operation `getAllItems` has been invoked for view items and option `positionNumbers` is effectively `true`. The position number of the view's top folder is an empty string. The position number of all other items consists of a dot-separated list of unsigned decimal numbers.

`basketItemIds`

Basket item IDs are returned only if operation `getAllItems` has been invoked for view items and option `basketItems` is effectively `false`.

Each view item references one or more basket items[29]. This field is used to return the basket item IDs of all basket items referenced by the view item.

In most cases, a single view item references a single basket item. There are some cases, however, where a view item may reference multiple basket items:

- merged article items (display mode `Sorted` with merge mode other than `None`)

  The order of basket item IDs returned for merged article items is unspecified.
- sub-article items folded into their main article item (display mode `Sorted` with merge mode `Compact`)

  The list of basket item IDs consists of the item ID of the main article item followed by the item IDs of sub-article items. The order of sub-article item IDs is unspecified.
- parts of collapsed set-articles

  The list of basket item IDs consists of the item ID of the set-article item followed by the item IDs of set-article parts. The order of set-article part IDs is unspecified. IDs of basket folder parts are not returned if the view expands basket folder items.

`basketItems`

If operation `getAllItems` is invoked for view items and option `basketItems` is `true` then this field is used to return one instance of `BasketItem` for each basket item referenced by the view item. See description of field `basketItemIds` for information about the set of referenced basket items.

Options `setArticleIds` and `geometryIds` may be used to control the content of these nested instances of `BasketItem`.

Note that the composite article hierarchy defined by the main- and sub-article identifiers is not related in any

28 This should usually be the case unless the basket item has been read from a BSK/OBX stream produced by an application that does not support geometry IDs.

29 In theory, there may be view items that do not reference basket items, but so far no such view items have been implemented.

way to the visible item hierarchy defined by the top folder and sub-item identifiers. For one thing it is not uncommon for meta types to be configured in such a way that sub-articles of meta types appear as siblings of the meta type in the order hierarchy. For another thing a future version of the Online Configurator may allow more or less unrestricted reordering of the hierarchy of basket items.

5.6.1.9 `CSArithmeticOperationError`

Synopsis:

```
struct CSArithmeticOperationError extends CSValidationError (
                                          CSOperationKind operationKind
                                                               )
```

A CSArithmeticOperationError is reported in case of division by zero (DivisionByZero), overflow, i.e. an infinite result (Overflow), or an operation whose result is undefined for the particular parameter values, like addition of positive and negative infinity, or multiplication of zero with infinity (InvalidOperation). Possible values for operationKind are Add, Subtract, Multiply, Divide, Remainder, and UnitConversion.

5.6.1.10 `CSConformabilityError`

Synopsis:

```
struct CSConformabilityError extends CSValidationError (
                                          CSOperationKind operationKind,
                                                   string unit1,
                                                   string unit2
                                                             )
```

A CSConformabilityError is reported to quantities with different units are added or divided, or if an attempt to convert a quantity to a different unit failed because both units are not related. The error code is ConformabilityError. Field operationKind is Add, Divide, or UnitConversion. Fields unit1 and unit2 are the units involved. In case of operation UnitConversion, unit1 is the source unit and unit2 the target unit.

5.6.1.11 `CSCurrencyError`

Synopsis:

```
struct CSCurrencyError extends CSValidationError (
                                    CSOperationKind operationKind,
                                             string currency1,
                                             string currency2
                                                       )
```

A CSCurrencyError is reported if an operation was performed on monetary values with different currencies (CurrencyMismatch) or if a currency conversion failed (CurrencyConversionFailed or UnknownCurrency). Field operationKind is Add, Subtract, Divide, Remainder, or CurrencyConversion. Fields currency1 and currency2 are the currencies involved. In case of operation CurrencyConversion, currency1 is the source currency and currency2 the target currency (usually the document currency).

5.6.1.12 `CSErrorCode`

Synopsis:

```
Restriction: string

enum CSErrorCode {
                   InconsistentItemState,
                   UnknownItemDataAccessError,
                   UndefinedValue,
                   InvalidValue,
                   InfiniteValue,
                   NullValue,
                   PercentageExpected,
                   MonetaryValueExpected,
                   MonetaryBaseExpected,
                   QuantityBaseExpected,
                   UnknownValue,
                   UnknownCurrency,
                   CurrencyConversionFailed,
                   CurrencyMismatch,
                   ConformabilityError,
                   DivisionByZero,
                   Overflow,
                   InvalidOperation,
                   NoPreviousLine,
                   ReferencedLineNotFound,
                   NoInversePriceCalculation,
                   UndefinedInverseGroupConditionCalculation
                 }
```

5.6.1.13 `CSInvalidValueError`

Synopsis:

```
struct CSInvalidValueError extends CSValidationError (
                                     CSValueKind parameterKind
                                                          )
```

A CSInvalidValueError indicates an invalid parameter of an arithmetic or other operation. Field parameterKind describes the parameter value, whereas resultKind describes the value computed by the operation. Possible error codes are UndefinedValue (no value at all, i.e. a null reference), InvalidValue (like an invalid monetary value, comparable with NaN), InfiniteValue, NullValue (a monetary value or quantity with value zero but no currency or unit), PercentageExcpected (like a condition with calculation rule Percent but a monetary amount), MonetaryValueExpected, MonetaryBaseExcpected, QuantityBaseExpected, and UnknownValue (a general failure to determine a particular value, like the set-article quantity of an item that is supposedly part of a set-article when the corresponding set-article calculation is not found).

5.6.1.14 `CSItemDataAccessError`

Synopsis:

```
struct CSItemDataAccessError extends CSValidationError ( )
```

A CSItemDataAccessError indicates a failure to access data of the underlying basket article item. Possible error codes are InconsistentItemState, indicating an inconsistency in the basket item structure, like a sub-article whose main article could not be found, and UnknownItemDataAccessError, indicating an unknown error. Field resultKind describes the value being accessed (like ItemQuantity or SalesUnitSize, among others).

5.6.1.15 `CSLineNotFoundError`

Synopsis:

```
struct CSLineNotFoundError extends CSValidationError ( )
```

A CSLineNotFoundError is reported if a calculation line that is supposed to be present is not found. This error should not happen and indicates a bug or possibly a problem with the pricing procedure.

5.6.1.16 `CSOperationKind`

Synopsis:

```
Restriction: string
```

```
enum CSOperationKind {
                        Add,
                        Subtract,
                        Multiply,
                        Divide,
                        Remainder,
                        CurrencyConversion,
                        UnitConversion
                    }
```

5.6.1.17 CSUndefinedOperationError

Synopsis:

```
struct CSUndefinedOperationError extends CSValidationError (
                                             CalculationRule calculationRule
                                                                   )
```

A CSUndefinedOperationError indicates an inability to perform an inverse computation (like during margin adjustment) and is not reported by normal validation of price calculations.

5.6.1.18 CSValidationError

Synopsis:

```
struct CSValidationError (
                   string message
              CSValueKind resultKind,
              CSErrorCode errorCode
                  )
```

5.6.1.19 CSValueKind

Synopsis:

```
Restriction: string

enum CSValueKind {
                  Other,
                  ItemQuantity,
                  SalesUnitSize,
                  SalesUnitGrossWeight,
                  SalesUnitNetWeight,
                  SalesUnitVolume,
                  InactiveItemFlag,
                  SetArticleIdentifier,
                  CondRecordAmount,
                  Incoming,
                  Reference,
                  Outgoing,
                  CondBase,
                  ScaleBase,
                  CondAmount,
                  DefaultQtyRel,
                  QuantityRelation,
                  CondBaseQtyRelRatio,
                  CondValue,
                  SubtotalValue,
                  SetArticleQty,
                  GroupCondBase,
                  AltGroupCondBase,
                  GroupScaleBase,
                  GroupAmount,
                  CondBaseRatio,
                  GroupValue,
                  GroupValueResidue,
                  AggrIncoming,
                  AggrReference,
                  AggrOutgoing,
                  AggrCondBase,
                  AggrCondValue,
                  AggrSubtotalValue,
                  AggrSetArticleQtys
                 }
```

Values of this type are used in CSValidationError to identify the value whose computation triggered the error, and in CSInvalidValueError to identify the origin of the offending value.

`Other`

Placeholder for other internally used value kinds that should not appear in validation errors reported for price calculation sheets

`ItemQuantity to SetArticleIdentifier`

Values fetched from article items

`CondRecordAmount`

The amount of a condition record

`Incoming`

The incoming value of an item calculation line, i.e. the outgoing value of the parent line

`Reference`

The reference value of an item calculation line according to the reference level or reference range

`Outgoing`

The outgoing value of an item calculation line; equal to the incoming value in case of subtotal and text lines, and equal to the sum of incoming value and condition value in case of condition lines

`CondBase`

The condition base used by item conditions with calculation rule Quantity, GrossWeight, NetWeight, or Volume; computed as the product of the article's quantity and either the sales unit size, sales unit gross/net weight, or sales unit volume; used by item conditions to compute the condition value as the product of condition amount and the quotient of condition base and quantity relation (CV = CA * (CB / QR)); accumulated by condition groups to compute the base value of the group

`ScaleBase`

Like CondBase, but used during computation of GroupScaleBase for item-specific intermediate results

`CondAmount`

The condition amount

`DefaultQtyRel`

The default quantity relation of an article item; equal to the article's sales unit size, sales unit gross/net weight, or sales unit volume; used if the condition has no condition record and no manually specified quantity relation

`QuantityRelation`

The quantity relation used by item conditions with calculation rule Quantity, GrossWeight, NetWeight, or Volume

`CondBaseQtyRelRatio`

The quotient of condition base and quantity relation

`CondValue`

The value of item conditions

`SubtotalValue`

The value of item subtotals

`SetArticleQty`

The quantity reported by set-article calculations

`GroupCondBase`

Base value of condition group, the sum of base values of all active conditions in the group

`AltGroupCondBase`

Alternative base value of condition groups used for inactive conditions (conditions overridden by a subsequent price or conditions of inactive item calculations (alternative positions)) with calculation rule FixedAmount; temporarily computed as the sum of group condition base and condition base value; used instead of group condition base as to compute condition base ratio (see below)

`GroupScaleBase`

The scale base of a condition group; Depending on the scale base type, this is either the accumulated item condition base (scale base type Value) or the accumulated item scale base (scale base types Quantity, GrossWeight, NetWeight, or Volume).

`GroupAmount`

Amount of a condition group; usually obtained from condition record, but may be manually overridden

`GroupValue`

The value of a condition group; computed from group amount and group base value

`CondBaseRatio`

The condition base ratio, i.e. the quotient of item condition base and group condition base; used to compute the initial (before rounding difference compensation) item condition value as the product of group value and condition base ratio

`GroupValueResidue`

The difference of group value and accumulated initial item condition values; may be distributed in some way among item condition values if rounding difference compensation is enabled

`AggrIncoming`

The incoming value of an aggregate calculation line, i.e. the outgoing value of the parent line

`AggrReference`

The reference value of an aggregate calculation line according to the reference level or reference range; may be used by aggregate subtotals in calculation of the value reported as the subtotal's amount

`AggrOutgoing`

The outgoing value of an aggregate calculation line; equal to the incoming value in case of subtotal and text lines, and equal to the sum of incoming value and condition value in case of condition lines

`AggrCondBase`

The aggregated condition base, i.e. the sum of item condition bases; used by group and set-article calculations to determine whether the value of the aggregate condition is editable (for the value to be editable, the aggregated condition base must have a finite non-zero value as it is used as divisor when the new aggregate condition value is distributed among item conditions)

`AggrCondValue`

The value of aggregate conditions

`AggrSubtotalValue`

The value of aggregate subtotals

`AggrSetArticleQtys`

Mapping from item calculation ID to set-article quantity for items that are part of a set-article; used by conditions of group and set-article calculations to determine whether their condition value is editable

5.6.1.20 ItemProperties

Synopsis:

```
struct ItemProperties {
    BItemId*                          itemId;
    BasketItemType*                   itemType;
    boolean*                          visible;
    boolean*                          expanded;
    string*                           label;
    BItemId*                          setArticleId;
    TMRow[]                           tmRows,
    string                            tmDescription,
    string                            positionNumber
    FolderProperties*                 folder;
    ArticleProperties*                article;
    SetArticleProperties              setArticle,
    ComposableGeometryProperties      composableGeometry;
    ComposableGeometryProperties[]    composableGeometries;
    TextItemProperties                textItem;
}
```

The `ItemProperties` structure is used as the return value of the `getItemProperties` (§5.6.3.32) operation and as an argument type of the `setItemProperties` (§5.6.3.33) operation. Each field of this structure (except the fields referencing another structure) corresponds to a basket item property.

`itemId`
: This field holds the UUID identifying the item.

`itemType`
: This field holds the type of this item. See §5.6.1.7 for a description of possible item types.

`visible`
: This field indicates whether or not the item should be displayed in the article list. This field is not returned for item properties of view items.

`expanded`
: If `false`, the article list should show children of this item as sub-positions. If `true`, children of this item should be displayed in place of this item (if `visible` is `false`), or as siblings following this item (if `visible` is `true`). This field is not returned for item properties of view items.

`label`
: This field contains the label for this item (the short text of article items or the name of folder items).

`setArticleId`
: If this item is part of a set article (§5.6.1.7), this field contains the item ID of the set article. Otherwise, it contains the null UUID (not a null reference).

`positionNumber`
: This field is present if and only if operation `getItemProperties` has been invoked for view items. The position number of the view's top folder is an empty string. The position number of all other items consists of a dot-separated list of unsigned decimal numbers.

`folder`
: When used with the `getItemProperties` operation, this field contains a reference to an instance of the `FolderProperties` (§5.6.1.21) structure if the corresponding basket item is a folder, and a null reference otherwise. When used with the `setItemProperties` operation, the value of this field is ignored if the item is not a folder item. If the item is a folder item, the field should contain a reference to an instance of `FolderProperties`.

`article`

When used with the `getItemProperties` operation, this field contains a reference to an instance of the `ArticleProperties` (§5.6.1.27) structure if the corresponding basket item is an article, and a null reference otherwise. When used with the `setItemProperties` operation, the value of this field is ignored if the item is not an article item. If the item is an article item, the field should contain a reference to an instance of `ArticleProperties`.

`composableGeometry`

With the properties for composable geometry items, it is possible to compose individual item geometries into a composite geometry. To do so, the items must be added as parts to a set-article, the property `enabled` (§5.6.1.91) must be set to true, and the position and or rotation may be set. The order of these operations does not matter.

The ID of the set-article can then be used with the operations `getGeneratedImage` `(§5.6.3.40)` and `getExportedGeometry` (§5.6.3.43) to generate an image for or export the geometry of the whole set-article.

Right now, the only set-article parts that can be used in this way are basket main article items.

Note that the use of set-article items with the aforementioned operations implies some restrictions compared with ordinary article items:

- The Multi Content Picture export (or whatever MCP) means) is not supported.
- The GFX geometry export does not allow export of OBX data.
- The effective value of the 3D export option `no2D` should be `true` as the OFML object composed for the set-article does not contain 2D geometries. If 2D geometries are to be exported, the export uses the GX hidden liner to produce 2D symbols, which may take some time.

  In particular, this means that `no2D=true` should be used for the DWG export. For all other exports allowing export of 2D geometries, the default value of this option is `true` anyway.

An empty string passed as item property `composableGeometry.geometry` of operation `setItemProperties` resets the geometry assigned to user article items.

`composableGeometries`

This field is used instead of field `composableGeometry` if operation `getItemProperties` is invoked for view items instead of basket items[30].

`TextItem`

When used with the `getItemProperties` operation, this field contains a reference to an instance of the `TextItemProperties` (§5.6.1.30) structure if the corresponding basket item is a text item, and a null reference otherwise. When used with the `setItemProperties` operation, the value of this field is ignored if the item is not a text item. If the item is a text item, the field should contain a reference to an instance of `TextItemProperties.`

While it is not an error to call the `setItemProperties` operation with an `ItemProperties` argument whose `folder` or `article` field (whichever corresponds the the item type) contains a null reference, such a call would be pointless as all the basket item properties corresponding to the other fields of the `ItemProperties` structure are read-only.

`tmRows`

possible empty list of item text rows, always empty if option `tmGetText` (§ 5.6.3.76) is false or missing, otherwise a list of all text rows, including those rows that do not have a text defined. If no entry is found for a language from the prioritized list of effective product data languages the fallback is the text stored for the undetermined language.

30 Field `composableGeometry` cannot be used in this case because view items may reference multiple basket items. Therefore, it may be necessary to return more than one instance of `ComposableGeometryProperties` for a single view item.

tmDescription

The text-manager generated article description, see option tmDescrMode for more information. If no entry is found for a language from the prioritized list of effective product data languages the fallback is the text stored for the undetermined language.

5.6.1.21 FolderProperties

Synopsis:

```
struct FolderProperties {
    string*  name,
    DisplayText[] displayName,
    DisplayText[]?  longDisplayText
    boolean* showSubTotal,
}
```

The FolderProperties structure is referenced by the ItemProperties (§5.6.1.20) structure. The fields of this structure correspond to basket item properties specific to folder items. The fields are as follows:

name

This field contains the name of the folder. When returned by the getItemProperties (§5.6.3.32) operation, the value of this field is equal to the value of the label field of the ItemProperties structure.

showSubTotal

This field indicates whether or not the article list should display a sub-total of the value of all positions contained within this folder.

5.6.1.22 GetArticleDataOptions

Synopsis:

```
struct GetArticleDataOptions (
                              boolean noProperties,
                              boolean fetchCatalogImage,
                              boolean fetchCatalogIcon,
                              string viewId
                              boolean separateCurrencies
                              )
```

noProperties

The noProperties option, if set to true, tells the operation not to fetch the articles properties. This may be useful as fetching the properties may be slow, and they should not be necessary to display the list of articles contained in the basket. Furthermore, if an article has been loaded from a project file (§5.4.3.10), the operation may not be able to fetch the properties if the correct version of the article's product data is not available and will fail if the noProperties option is not set to true.

fetchCatalogImage

With the option fetchCatalogImage set to true, the operation getArticleData returns an URL for the article's catalog image, or an empty string if no catalog image has been found. The default value of this options is true.

viewId

If this field is present its value must be empty, be the NIL UUID, or the ID of a current basket view.

If the field contains the ID of a current basket view then itemId parameter must contain the ID of an item of that basket view and this item must reference a basket item. The operation will then behave as if invoked for that basket item.

If the field is missing or contains an empty string or the NIL UUID then the operation will behave as before.

5.6.1.23 `GetArticleFeaturesOptions`

Synopsis:

```
struct GetArticleFeaturesOptions (
                                  boolean description,
                                  boolean noInternal
                                )
```

5.6.1.24 `GetMultiArticleFeaturesOptions`

Synopsis:

```
struct GetMultiArticleFeaturesOptions : ItemSelectionOptions
        {
        boolean? description;                              // attribute
        boolean? noInternal;                               // attribute
        }
```

Note: Fields `description` and `noInternal` of type `GetMultiArticleFeaturesOptions` are encoded as attributes, even though the same fields of type `GetArticleFeaturesOptions` (§5.6.1.23) are encoded as elements.

5.6.1.25 `GetImagesOptions`

Synopsis:

```
GetImagesOptions extends ItemSelectionOptions (
                                       boolean useCache,
                                       boolean generate,
                                       boolean itemIds,
                                       boolean attributes
                                             )
```

`useCache`

If true, and the basket does not yet have a default article image representing the current configuration, a global cache look-up will be done and, if a matching image is found there, it will be used.

The default value of this option is false.

`generate`

If true, and the basket does not yet have a default article image representing the current configuration, a global cache look-up will be performed1 and, if no matching image is found there, an attempt will be made to generate/render the image.

The default value of this option is false.

`itemIds`

If true, returned ImageInfo instances will contain the item ID of the corresponding basket or view item.

The default value of this option is false.

`attributes`

If true, the attributes used to generate the image, or specified when the image was imported, are returned as part of ImageInfo.

For historical reasons, the names of these attributes differ from the names of the options passed to operation getGeneratedImage. There are attributes that do not have a corresponding option, and options that are not stored as an attribute. Nevertheless, some of the attributes may be useful to the client.

The default value of this option is false.

5.6.1.26 `GetManufacturerInfoOptions`

Synopsis:

```
struct GetManufacturerInfoOptions extends ItemSelectionOptions (
            boolean allData,
            boolean allLanguages,
            boolean manufacturerName,
            boolean seriesInfo,
            boolean seriesName,
            boolean manufacturerConfig,
            boolean distributorName,
            boolean copyright,
            boolean address
            )
```

5.6.1.27 `ArticleProperties`

Synopsis:

```
struct ArticleProperties {
    string*                   catalogId;

    string*                   manufacturerId;
    string*                   manufacturerName;
    string*                   seriesId;
    string*                   seriesName;

    string*                   manufacturer;
    string*                   program;
    string*                   distributionRegion;

    string*                   baseArticleNumber;
    string*                   finalArticleNumber;
    string*                   variantCode;
    string*                   ofmlVariantCode;
    boolean*                  useFinalArticleNumber;

    string*                   shortText;
    string*                   longText;
    string*                   featureText;
    ArticleDescriptionMode*   descriptionMode;
    ArticleDescription        productDescription,
    ArticleDescription        userDescription,
    string                    currency,
    string                    purchaseCurrency,
    string                    salesCurrency,
    decimal                   purchasePrice,
    decimal                   salesPrice,
    boolean                   priceDateSupported,
    boolean                   priceDateEditable,
    date                      priceDate,
    decimal                   quantity,
    boolean                   alternativePosition,
    string[]                    excludedCalculations,
    InactivePositionState[]      inactivePositionState
    boolean                     inconsistencyFlag,
    string[]                    inconsistencyReason,
    PriceInfo                   priceInfo,
    PackagingInfo               packagingInfo,
    boolean                     pseudoArticle,
    boolean?                           nonOfferArticle,
    boolean                     nonOrderArticle,
    ArticleClassification[]      articleClassifications,
    OFMLUpdateState             ofmlUpdateState,
    string                      geometryId,
    ArticleProperties[]         subArticles,
    AddStateCode[]              addStateCodes,

}
```

The `ArticleProperties` structure is referenced by the `ItemProperties` (§5.6.1.20) structure. The fields of this structure correspond to basket item properties specific to article items. The fields are as follows:

`catalogId`

If the article is an OFML or OCD article, then this field contains the identifier of the catalog containing the product data of the article if the catalog is represented by a catalog profile (as opposed to an old-style manufacturer profile). See §5.5.1.31 for more information about catalog identifiers.

`manufacturerId`

This field contains the commercial identifier of the article's manufacturer. When used with the `setItemProperties` (§5.6.3.33) operation to set the properties of an user article or set article, this field should contain either a null reference, an empty string or a valid manufacturer ID (see below).

`manufacturerName`

This field contains the name of the manufacturer. The manufacturer name depends on the manufacturer ID.

`seriesId`

This field contains the identifier for the article's series. When used with the `setItemProperties` (§5.6.3.33) operation to set the properties of an user article or set article, this field should contain either a null reference, an empty string or a valid series ID (see below).

`seriesName`

This field contains the name of the article's series. The series name depends on the manufacturer ID and series ID.

`Manufacturer, program, distributionRegion`

The elements contain the OFML manufacturer, OFML program and OFML distribution region of the package containing the OFML article data last used to insert or configure the OFML article.

Fields `manufacturer` and `program` should always be present in return value of operation `getItemProperties` for items of type `Article`, `Aggregate` and `PartialPlanning`. Field `distributionRegion` is usually present for items of these types unless the OFML article item has been read from a BSK/OBX stream and the corresponding `<pdInfo>` element did not have attribute `region`.

`baseArticleNumber`

This field contains the base article number.

`finalArticleNumber`

This field contains the final article number.

`variantCode`

This field contains the variant code.

`ofmlVariantCode`

This field contains the OFML variant code of the OFML article. Other than the ordinary variant code, the OFML variant code always uses a key-value encoding, allowing it to be reliably parsed. This element is returned for OFML articles only.

`useFinalArticleNumber`

If true, the client should display the final article number. If false, the client should display the base article number. This element is returned for OFML articles only.

`shortText`

This field contains the short description of the article. While not enforced by the Online Configurator, the short description should consist of a single line of text only. The short description describes the article and is supposed not to depend on the current configuration of the article.[31]

`longText`

This field contains the long description of the article, possibly consisting of multiple lines of text. The long description may describes the article in more detail than the short description, but, like the short description, should not depend on the current configuration of the article.

`featureText`

The feature description describes the configuration-dependent features of the article.

`DescriptionMode`

The field will always be present in instances of `ArticleProperties` returned from the server to the client. It is optional because the client does not need to set this field when it passes an instance of `ArticleProperties` to the server. In fact, setting this field has no effect[32].

`currency`

If the article position has a price, then this field contains the ISO 4217 currency code of the price (pseudo-currency codes are not supported).

`PurchasePrice`

This field contains the purchase price as read from the product database, possibly adjusted by an OFML price profile. If no purchase price could be determined, the field is not available. The field will not be provided if the currencies of purchase price and sales price are different and the value of the purchase price is non-zero.

`SalesPrice`

This field contains the sales price as read from the product database, possibly adjusted by an OFML price profile. If no sales price could be determined, the field is not available.

`priceInfo`

This field is used to return price information if option `priceInfo` in type `GetItemPropertiesOptions` is `true`, return of price information is enabled[33] and valid price information could be determined. No valid price information can be determined if the sales unit size and/or quantity of the calculation do not have finite values or, in merge mode `Compact`, the article properties are returned for a sub-article item and the quantity of the main article item is zero[34].

`quantity`

This field contains the position's quantity. The exact meaning depends on the article's order unit. The Online Configurator does not limit the quantity to a certain range or step size.

31 This does not mean that the (short) description of an article position does not change if a property of the position is changed. For instance, in the case of meta types, a property change may select another article, and thus result in another (short) description.

32 Operation `setItemProperties` ignores an article description mode specified by the client.

33 application feature `egr.eai.ws.basket.ReturnPriceInfo` is available

34 To determine the quantity of the sub-article item it is necessary to divide the total quantity of the sub-article item, as returned by the corresponding item property, by the quantity of the main article item, thus resulting in division of zero by zero if the main article item's quantity is zero. While this problem could be circumvented with a new property that contains the sub-article item's quantity relative to the main article item, this would not help much as the net value returned by the calculation would be zero too, and thus there would be no (easy) way to compute a net price.

`alternativePosition`

This field indicates whether or not the position is an alternative position. Alternative positions are supposed to be ignored when the total value of an order (or sub-totals for folders) are computed.

The `alternativePosition` element of complex type `articleProperties` has been deprecated, but is still supported for backward compatibility. With the new implementation, the value is `true` if the excluded calculation set is either the universal set or contains the names of all calculations (pricing procedures) used by the basket.

`excludedCalculations`

This field indicates the names of all price calculations which consider the item an alternative position (the name of a price calculation is equal to the name of the pricing procedure used by this calculation). As a special case, this set may be the universal set, containing all possible price calculation names. The universal set is represented by a set containing only the empty string.

Note that the `excludedCalculations` elements are used by the `getItemProperties` operation only. The `setItemProperties` operation ignores these elements, if present.

`inconsistencyFlag`

The value of this field indicates whether or not the Online Configurator detected an inconsistency in the article's current configuration.[35] If this field is `true`, an inconsistency has been detected, and the article's price and/or the position's value are probably invalid.

`inconsistencyReason`

If the value of field `inconsistencyFlag` is `true`, then this field should contain a description of the inconsistency's reason or reasons.

`packagingInfo`

When used with the `getArticleProperties` operation, this field contains a reference to an instance of the `PackingInfo` (§5.6.1.28) structure. When used with the `setArticleProperties` operation, the value of this field is ignored if the item is not an `UserArticle` (§5.6.1.7). If the item is an `UserArticle`, the field should contain a reference to an instance of `PackingInfo`.

`ofmlUpdateState`

If the `getItemProperties` (§5.6.3.32) operation is used to get the properties of an OFML article item, this element will be present and contains the update state of the article.

The update state returned here is always with respect to the full set of registered OFML catalogs. Furthermore, since computation of the update state may be a somewhat expensive operation, the update state is returned if and only if it is already known. If not, the returned update state is `Unknown`.

`geometryId`

The element occurs in result of operation 'getItemProperties' if and only if the necessary set of geometry checksums is stored in the corresponding basket article item (i.e. no attempt is made to get them from the OFML article item, even if the OFML article item has already been instantiated). This should always be the case unless the article item has been loaded from old BSK/OBX data, or from BSK/OBX data written by another application.

In case of main article items the geometry ID consists of the geometry checksum for the whole (composite) article. For sub-article items it consists of the geometry checksum for the whole sub-article plus a checksum computed for the transformation of the sub-article relative to the main article.

`subArticles`

This field is used to return data about sub-articles if operation `getItemProperties` has been called for a view item of type `PartialPlanning` or `Aggregate`, the view uses display mode `Sorted` and merge mode `Compact`, and the item has sub-articles.

[35] The fields `inconsistencyFlag` and `inconsistencyReason` may also be used to report other problems preventing the Online Configurator from the computation of a valid price or value for this position.

`inactivePositionState`

This field is a possibly empty array of instances of InactivePositionState, one for each pricing procedure that treats the position as an inactive position, or with a single element with an empty ppName field (see above).

Valid manufacturer and series IDs consist of a non-empty sequence of graphical ASCII characters except quotation mark (`"`), asterisk (`*`), slash (`/`), colon (`:`), less-than sign (`<`), greater-than sign (`>`), question mark (`?`), backslash (`\`) and vertical line (`|`)[36].

The purchase price and sales price are the prices of one order unit. They need to be multiplied by the quantity to obtain the position's value. They are not set to zero if the position is an alternative position.

Fields `pseudoArticle` and `nonOrderArticle` are returned for all article items, even though they are, as of now, only meaningful for OFML article items.

Field `articleClassifications` is returned for all article items except set-article items, and only if option `articleClassifications` in `GetItemPropertiesOptions` is set to true.

All three new fields are ignored by operation `setItemProperties`.

5.6.1.28 `PackagingInfo`

Synopsis:

```
struct PackingInfo {
    Quantity*     width;
    Quantity*     height;
    Quantity*     depth;
    Quantity*     volume;
    Quantity*     tareWeight;
    Quantity*     netWeight;
    int*          itemsPerPackUnit;
    int*          packUnitsPerArticle;
}
```

The `PackingInfo` structure is referenced by the `ArticleProperties` (§5.6.1.27) structure. This element is used to return OFML article packaging information to the client, and, in case of user articles, to allow the client to set the packaging information.The fields are as follows:

`width`

This field contains the width of the packaging unit of the article.

`height`

This field contains the height of the packaging unit of the article.

`depth`

This field contains the depth of the packaging unit of the article.

`volume`

This field contains the volume of the packaging unit of the article.

`tareWeight`

This field contains the weight of the packaging unit of the article.

36 While there seems to exist no official definition of valid manufacturer and series IDs, the Online Configurator restricts them to a set of characters that are not known to cause problems in certain contexts. For instance, Microsoft Windows prohibits the use of certain characters in file names. These characters should not be used in manufacturer IDs either, as manufacturer IDs are used as the base name of manufacturer registration files (`<data>/registry/<manufacturer_id>.cfg`).

`netWeight`

This field contains the weight of the individual article.

`itemsPerPackUnit`

This field contains the number of articles per packaging unit.

`packUnitsPerArticle`

This field contains the number of packaging units which are used for the article.

A NULL quantity is used if an article has no length, volume or weight (i.e. no physical existence). In case of an unknown length, volume or weight the corresponding element is not returned.

To reset a packaging info quantity (of an user article), the client can invoke the `setItemProperties` operation with the corresponding field set to an invalid quantity (complex type `Quantity` with the `unit` attribute set to an empty string and no `value` attribute).

5.6.1.29 PartCompositionFailure

Synopsis:

```
PartCompositionFailure (
                string cause
                string itemId
 )
```

Instances of `PartCompositionFailure` will be used in `BasketServiceFault` if thrown by operations getGeneratedImage and getExportedGeometry if these operations were invoked to get an image or geometry for a set-article item and the geometry compositor failed to obtain all set-article part geometries.

The field `itemId` holds the basket item ID of the set-article part whose geometry could not be obtained, and the array `cause` contains the chain of exceptions that led to the failure. It contains at least one element. Each element represents one exception and consists of the exception type, optionally followed by a colon, a space, and the exception message.

5.6.1.30 `TextItemProperties`

Synopsis:

```
struct TextItemProperties {
    string        text;
    DisplayText[] displayText;
    DisplayText[]?  longDisplayText
}
```

The `TextItemProperties` structure is referenced by the `ItemProperties` (§5.6.1.20) structure. The fields of this structure correspond to basket item properties specific to text items.

`text`

This field contains the text information.

5.6.1.31 `Value`

Synopsis:

```
struct Value {
    decimal* value;
}
```

Value is an abstract type used as base for Money (§5.6.1.34), Percentage (§5.6.1.35) and Quantity (§5.6.1.32). The value represented by these complex types is either valid or invalid. An invalid value is indicated by a missing `value` attribute. Unless explicitly specified otherwise, the web service does not return elements of above types which represent invalid values. Instead, the element is omitted.

5.6.1.32 `Quantity`

Synopsis:

```
struct Quantity : Value {
    string        unit;
}
```

The `Quantity` structure represents either a valid quantity, a null quantity, or an invalid quantity. In case of a valid quantity, the `value` attribute inherited from the `Value` type is present, and the `unit` attribute contains a valid UNECE unit of measure. In case of a null quantity, the `value` attribute contains the value zero, and the `unit` attribute contains an empty string. Null quantities may be used if it is known that the quantity is zero but no information about a particular unit of measure is available. An invalid quantity is represented by a missing `value` attribute and an `unit` attribute containing an empty string.

`unit`

This attribute contains the unit of the value inherited from the `Value` type (§5.6.1.31). The unit attribute must be empty if the value attribute is missing. If the value attribute exists, the unit attribute must represent a valid UNECE unit of measure, unless the value is zero, in which case the unit attribute may be empty.

5.6.1.33 `ManufacturerInfo`

Synopsis:

```
struct ManufacturerInfo (
             string concernId,
             string supplierId,
             string priceProfileRegionId,
             DisplayText[] manufacturerName,
             SeriesInfo[] seriesInfos,
             DisplayText[] distributorName,
             DisplayText[] copyright,
             DisplayText[] addressName,
             DisplayText[] addressStreet,
             DisplayText[] addressCity,
             DisplayText[] addressState,
             DisplayText[] addressPOBox,
             DisplayText[] addressZIP,
             DisplayText[] addressCountry,
             DisplayText[] addressEMail,
             DisplayText[] addressPhone,
             DisplayText[] addressFAX,
             DisplayText[] addressWWW,
             string manufacturerId,
             string? ExternalCatalogURL,
             DisplayText[]? externalCatalogName
                         )
```

`externalCatalogURL`

This field is present if the effective value of option `externalCatalog` is `true` and the manufacturer registration contains a HTTP/HTTPS URL for the external catalog of this manufacturer.

`externalCatalogName`

This field is present if and only if field `externalCatalogURL` is present. It contains a possibly empty list of `text` elements of type DisplayText with attribute lang specifying the text's language (missing in case of undetermined language) and the actual text as the text element's content.

Depending on the effective value of option `allLanguages`, field `externalCatalogName` returns all the external catalog names as specified in the manufacturer registration file (`allLanguages` is `true`) or a single (possibly empty) external catalog name selected based on the effective list of project languages (`allLanguages` is `false`). In the latter case, the `text` element does not contain the `lang` attribute, even if the name originates from a language-specific section of the manufacturer registration file.

5.6.1.34 `Money`

Synopsis:

```
struct Money : Value {
    string        currency;
}
```

The `Money` structure represents either a valid monetary quantity, a null quantity, or an invalid quantity. In case of a valid quantity, the `value` attribute inherited from the `Value` type is present, and the `currency` attribute contains a valid ISO currency code. In case of a null quantity, the `value` attribute contains the value zero, and the `currency` attribute contains an empty string. Null quantities may be used if it is known that the quantity is zero but no information about a particular currency is available. An invalid quantity is represented by a missing `value` attribute and a `currency` attribute containing an empty string.

`currency`

This field contains the currency of the value inherited from the `Value` type (§5.6.1.31). The currency attribute must be empty if the value attribute is missing. If the value attribute exists, the currency attribute must represent a valid ISO currency code (but no pseudo-currency[37]), unless the value is zero, in which case the currency attribute may be empty.

5.6.1.35 `Percentage`

Synopsis:

```
struct Percentage : Value {}
```

The `Percentage` structure represents either a valid or invalid percentage. The percentage is valid if the inherited value attribute is present, and invalid otherwise.

5.6.1.36 `PropertyClass`

Synopsis:

```
struct PropertyClass {
    string name;
    string description;
}
```

The `PropertyClass` structure contains the symbolic name and descriptive text of a property class. Instances of this structure are used as part of the `ArticleData` structure (§5.6.1.42) to return information about the property classes of an article.

`name`

the symbolic name of the property class; the values stored in this field correspond the the values stored in the `propClass` field of the `Property` (§5.6.1.40) structure.

Properties that do not belong to a particular property class are assigned to the pseudo-class named `OI_NONE_PROPCLASS`.

37 Pseudo currencies are all currencies with a currency code starting with the letter `'X'` except `XAF`, `XCD`, `XOF` and `XPF`.

`description`

This is the descriptive text of the property class. The description depends on the list of effective product data languages (see operation `setLanguages`, section 5.6.3.4). If no description is available, the returned description is either empty or equal to the name of the property class. In particular, for property class `OI_NONE_PROPCLASS` the client should be prepared to supply its own description.

5.6.1.37 `PropertyType`

Synopsis:

```
enum PropertyType {
    Character,
    Numeric,
    Length,
    Boolean
}
```

The `PropertyType` enumeration is used to specify the fundamental type of an article property. The following enumeration values are defined:

`Character`

The value of a character property consists of a possible empty sequence of arbitrary characters. Properties of type `Character` are also called string properties.

`Numeric`

The value of a numeric property is a fixed point decimal number. The complete property definition (§5.6.1.40) contains both the total number of digits and the precision (the number of digits right of the decimal point).

`Length`

A property of this type is a numeric property with the additional semantics that the value represents a length in meter. Properties of this type are used to allow the user interface to convert the property values into another unit before they are displayed to the user.

`Boolean`

Previously, boolean properties have been treated as properties of type `Numeric` with values `one` and `zero` for `true` and `false` when returned by operations `getArticleData`, `getChoiceList` and `getAllChoiceLists`, or `zero` and `non-zero` for `false` and `true` when set with operation `setPropertyValue`.

An optional boolean field `enableBooleanPropType` has been added to the `options` argument passed to these operations. Its default value is `false`, resulting in no change of behavior.

5.6.1.38 `PropertyValue`

Synopsis:

```
struct PropertyValue {
    string        value;
    string        text;
    URL           smallIcon;
    URL           largeIcon;
    Money?        Surcharge;
    string?       Image;
    boolean?      Selectable;
}
```

Instances of the `PropertyValue` structure are returned by the Online Configurator to completely describe a property value. The structure has the following fields:

`value`

This is the internal property value.

For properties of type `Character` this field contains either an internal symbolic character sequence or human readable text[38].

For properties of type `Numeric` and `Length` this field contains the numeric value of the property formatted as follows: The integral part of the value is formatted as an optional minus sign followed by one or more decimal digits without any insignificant leading zeros. If the property is declared with a non-zero number of decimal digits (digits right of the decimal point), the formatted integral part is followed by a decimal point (`.`) and the number of declared decimal digits. XXX

`text`

This is the human readable property value.

For properties of type `Character` this is either the same as `value`, or, if `value` is a symbolic character sequence, a human readable text representing the symbolic value. The language used for the human readable text depends on the configuration of the current session and the languages supported by the product data.

For properties of type `Numeric` and `Length` this is the numeric value of the property formatted according to the rules of the locale configured for the current session.

`smallIcon`

If the property has a choice list, and a small icon is available for this property value, then this field contains an URL referencing the image for the small icon. Otherwise it contains an empty string.

`image`

The field, if present, contains the URL of a large material image (stored in directory .../mat/l).

`largeIcon`

If the property has a choice list, and a large icon is available for this property value, then this field contains an URL referencing the image for the large icon. Otherwise it contains an empty string.

`Surcharge`

This field is used by operations getChoiceList and getAllChoiceLists to return the property value's surcharge if

- return of price information is enabled

- OFML actually reports a surcharge for the property value

- The effective value of option discardSurcharge is false.

If present, the surcharge will always be a finite monetary quantity, i.e. have a finite, possibly zero, value and a valid currency.

`selectable`

Depending on whether an instance of `PropertyValue` represents an actual (current) property value or a choice list value, field `selectable` is missing (property value) or present (choice list value). In the latter case, a value of `false` indicates that the choice list value should be displayed, but must not be available for selection as the new property value.

If the value of a property is undefined then the `value` field of the `PropertyValue` structure has one of the following special values:

`@UNDEFINED`

the property is neither visible, nor of type `Character`, nor does it have a choice list

`@VOID`

used if the property is an optional property

[38] Symbolic character sequences are usually used with visible properties that have choice lists and do not allow additional values.

`@UNSPECIFIED`

used if the property is an restrictable, but not an optional property

`@NOVALUE`

used if the property is neither an optional nor a restrictable property

The content of the `text` field of undefined property values is undefined and may change in further versions of the Online Configurator. The `smallIcon` and `largeIcon` fields of undefined property values contain an empty string.

5.6.1.39 `Interval`

Synopsis:

```
struct Interval {
    decimal minValue;
    decimal maxValue;
}
```

The complete property definition (§5.6.1.40) of a numeric or length property may use one or more instances of the `Interval` structure to limit the valid range of values for the property.

`minValue`

the minimum value of the interval

`maxValue`

the maximum value of the interval

Both the minimum and maximum values of the interval are included in the interval and adhere to the precision and the total number of decimal digits as specified by the associated property definition.

5.6.1.40 `Property`

Synopsis:

```
struct Property {
    string          propClass;
    string          propName;
    string          propText;
    PropertyType    type;
    int             width;
    int             digits;
    int             decDigits;
    boolean         visible;
    boolean         editable;
    boolean         addValues;
    boolean         choiceList;
    PropertyValue   value;
    Interval[]      intervals;
    boolean          multiLine
}
```

The `Property` structure contains all available information about a property, except for the choice list if there is one. It contains the following fields:

`propClass`

The property class is a symbolic name used to classify properties. Property class and property name uniquely identify the property with respect to the article. The property class names stored in this field correspond to the property class name stored in the `name` field of the `PropertyClass` structure (§5.6.1.36).

`propName`

The property name is the symbolic name of the property. The property name is unique relative to the property class, but there is no guarantee that no two properties of an article have the same property name.

`propText`

The property text is a short human readable description of the property, suitable for use in a tabular property editor. The language of the text depends on the configuration of the current session and the languages available in the product data.

`type`

This is the fundamental type of the property. See 5.6.1.37.

The value of this field will never change for a particular property.

`width`

This is the maximum number of characters for properties of type `Character`. For other property types the value of this field is undefined and should not be used.

The value of this field will never change for a particular property.

`digits`

This is the maximum number of significant decimal digits for properties of type `Numeric` and `Length`. For other property types the value of this field is undefined and should not be used.

The value of this field will never change for a particular property.

`decDigits`

For properties of type `Number` and `Length`, this is the number of decimal digits right of the decimal point (the precision). For other property types the value of this field is undefined and should not be used.

The value of this field will never change for a particular property.

`visible`

This field indicates whether the property should be displayed by a property editor[39]. The value of this field may change depending on the current configuration.

`editable`

This field indicates whether the property may be changed by the client[40]. It should be ignored for invisible properties. The value of this field may change depending on the current configuration.

The field `editable` is always `true` if field `visible` is `false`. The OFML API does not allow to differentiate between editable and read-only invisible properties, and EAIWS does not prevent the client from setting the value of an invisible property, so the value `true` is more appropriate.

`addValues`

This field indicates whether the property may assume values (and may be set by the client to a value) other then the values found in the choice list of the property. For numeric and length properties, the restrictions imposed by possible intervals are not affected by the value of this field.

`choiceList`

This field indicates whether a property has a choice list. Unless the field `addValues` is `true` a choice list restricts the allowed property values to the elements of the choice list. The value of this field may change depending on the current configuration.

`value`

This field contains information about the current value of the property. See 5.6.1.38.

39 The `visible` field is not used to mark properties that have been explicitly hidden by the relation knowledge as invisible, as such properties are not returned by the `getArticleData` operation. Instead, it is used to support properties that may affect the graphical representation of the article but are not to be displayed by a property editor.

40 The property may be marked editable even if the value of the property cannot really be changed, as may be the case if the current value is the only element of the properties choice list and the property does not allow additional values.

`intervals`

This field contains a sequence of intervals. For numeric and length properties this sequence may contain one or more elements defining the allowed set of values for this property. For properties of other types this sequence is always empty.

The allowed set of values of a property, and thus the intervals, may change depending on the current configuration.

5.6.1.41 `PriceComponent`

Synopsis:

```
struct PriceComponent {
    string  name;
    decimal value;
}
```

The `PriceComponent` structure contains information about a single price component. The following fields are defined:

`name`

The `name` field contains a short description the price component. The language of the text depends on the configuration of the current session and the languages available in the product data.

`value`

The `value` field contains the value of the price component.

5.6.1.42 `ArticleData`

Synopsis:

```
struct ArticleData {
    string                     manufacturerId;
    string                     seriesId;
    string                     baseArticleNumber;
    string                     variantCode;

    string                     shortText;
    string                     longText;
    string                     featureText;
    ArticleDescriptionMode     descriptionMode;

    PropertyClass[]            propertyClasses;
    Property[]                 properties;

    URL                        catalogImage;
    URL                        catalogIcon;

    string                     currency;
    decimal*                   pdPurchasePrice;
    decimal*                   pdSalesPrice;

    PriceComponent[]           pdPurchasePriceComponents;
    PriceComponent[]           pdSalesPriceComponents;

    string                     purchaseCurrency,
    string                     salesCurrency,
    boolean                    priceDateSupported,
    boolean                    priceDateEditable,
    date                        priceDate
}
```

The `ArticleData` structure contains most product information available for a particular article. The following fields are defined:

`manufacturerId`
: This field contains the commercial manufacturer ID of the article.

`seriesId`
: This field contains the commercial series ID of the article.

`baseArticleNumber`
: This field contains the base article number of the article.

`variantCode`
: This field contains the variant code of the article. Naturally, the variant code depends on the current configuration of the article.

  For ordinary OCD articles, manufacturer ID, series ID, base article number and variant code are sufficient to identify an article and its current configuration.

`shortText`
: This field contains the short article text – usually a single line describing the article. The language of the text depends on the configuration of the current session and the languages available in the product data.

`longText`
: This field contains the long article text – often consisting of more than a single line. The language of the text depends on the configuration of the current session and the languages available in the product data.

`featureText`
: This field contains a description of the article features / properties. It depends on the current configuration of the article. The language of the text depends on the configuration of the current session and the languages available in the product data.

`DescriptionMode`
: The field will always be present in instances of `ArticleProperties` returned from the server to the client. It is optional because the client does not need to set this field when it passes an instance of `ArticleProperties` to the server. In fact, setting this field has no effect[41].

`propertyClasses`
: This field contains a possibly empty sequence of all property classes (§5.6.1.36) of the article. It may contain property classes not used by any property stored in the `properties` field.

  The primary display order of properties in a property editor is defined by the order of property classes as stored in this field. Within a single property class, the secondary display order is defined by the order of properties as stored in the `properties` field.

  If a property editor groups properties below a property class, it should not display property classes with no associated visible properties, as the list of property classes may include classes that are used by internal properties only.

  Note that the mapping from symbolic property class name to property class description may differ from position to position, and may change even for a given position after a change to the position's configuration. Therefore, it is not recommended to cache the mapping from property class names to property class description.

`properties`
: This field contains a possibly empty sequence of the properties (§5.6.1.40) of the article. The sequence contains all user visible properties (configurable and read only properties to be displayed by

41 Operation `setItemProperties` ignores an article description mode specified by the client.

a property editor) as well as purely graphical properties (properties affecting only the graphical representation of the article, like geometry and material).

Naturally, the content of the sequence depends on the current configuration. This does not only apply to the content of the `Property` structures, but also to the set of properties as identified by their class and name. However, the order of properties is guaranteed not to change[42].

`catalogImage`

If, during insertion of the article into the basket structure, information about the catalog containing the article has been provided[43], and if the Online Configurator has been able to find an image for this article in the catalog, then this field contains an URL referencing this image. Otherwise, this field contains an empty string.

Although, in theory, the value of this field depends on the current configuration, catalogs usually do not contain entries for configuration-dependent images and thus the value of this field usually does not change for a particular article.

`catalogIcon`

This element is present if and only if option `fetchCatalogIcon` is `true`.

`currency`

This field contains the ISO 4217 currency code (but no pseudo-currency code) of the article's purchase and sales prices as reported by the values of the `pdPurchasePrice`, `pdSalesPrice`, `pdPurchasePriceComponents` and `pdSalesPriceComponents` fields.

If neither a purchase nor a sales price could be determined for (the current configuration of) the article, this field contains an empty string.

The actual currency depends on the configuration of the current session and the currencies available in the product data.

`pdPurchasePrice`

This field contains the purchase price for the current configuration of the article as determined by the product data. If no purchase price could be determined, the field is not available. The field will not be provided if the currencies of purchase price and sales price are different and the value of the purchase price is non-zero.

`pdSalesPrice`

This field contains the sales price for the current configuration of the article as determined by the product data. If no sales price could be determined, the field is not available.

`pdPurchasePriceComponents`

This field contains the individual components of the article's purchase price. If not empty, the sum of the listed price components is equal to the value of the `pdPurchasePrice` field.

Even with a non-zero purchase price the list of price components may be empty if, for whatever reason, the Online Configurator was not able to produce a proper list of purchase price components.

`pdSalesPriceComponents`

This field contains the individual components of the article's sales price. If not empty, the sum of the listed price components is equal to the value of the `pdSalesPrice` field.

Even with a non-zero sales price the list of price components may be empty if, for whatever reason, the Online Configurator was not able to produce a proper list of purchase price components.

42 If a particular configuration of an article contains properties A and B, with property A preceding property B, then in any other configuration of the article containing both the properties A and B, property A precedes property B.

43 The `catalogId` and/or `catalogPackageId` fields of the `InsertInfo` structure have been set to valid values.

5.6.1.43 `ArticleFeature`

Synopsis:

```
struct ArticleFeature {
    string  symbolicName;
    string  displayName;
    string  symbolicValue;
    string  displayValue;
    boolean visible;
}
```

The `ArticleFeature` structure is used as the element type of the return value of the `getArticleFeatures` operation (§5.6.3.35). It has the following fields:

`symbolicName`
: This field contains the symbolic name of the feature. The symbolic name is never an empty string. If the article is based on OCD product data, the symbolic feature name is equal to an OCD property name.

`displayName`
: This field contains the feature name as displayed to the user. If the feature is visible, the display name of the feature is never an empty string.

`symbolicValue`
: This field contains the symbolic value of the feature. If the article is based on OCD product data, the symbolic value is equal to the OCD property value of the OCD property identified by the symbolic name. The symbolic value may be an empty string, even if the display value is not an empty string.

`displayValue`
: This field contains the feature value as displayed to the user. The display value may be an empty string, even if the feature is visible, or the symbolic value is not an empty string.

`visible`
: This field is set to `true` if the feature's display name and possibly display value are displayed to the user. If set to `false`, the feature is internal and not to be displayed to the user.

A visible `ArticleFeature`, when displayed to the user, always results in a single line. If the description of a feature consists of multiple lines (as may be the case when the `getArticleFeatures` operation is called with the `description` option set to `true`), the sequence returned by this operation contains multiple subsequent instances of `ArticleFeature` with equal `symbolicName` and `symbolicValue`, each representing one line of the feature's description.

If the `description` option of the `getArticleFeatures` operation was set to `true`, the `displayName` field of returned `ArticleFeature` structures does not necessarily represent the features name. Instead, display name and display value should be treated as a pair of strings, at least one of them not empty, to be displayed in a single line.

Depending on the underlying product data, the display name as stored in the `ArticleFeature` structure may end with a colon (`:`), possibly followed by space. This should be taken into account when display name and display value are concatenated before they are displayed to the user.[44]

[44] When the Online Configurator concatenates the display name and value of a feature line to build the feature text returned by the `getArticleData` and `getItemProperties` operations, it behaves as follows: If the name is empty, the value is used (should not happen); otherwise, if the value is empty, the name is used; otherwise, if name ends with a colon (`:`), name and value are separated by space; otherwise, if the name ends with a colon followed by a single space character (U+0020), nothing is inserted between name and value; otherwise, name and value are separated by a colon followed by a single space.

5.6.1.44 ArticleFeatures

Synopsis:

```
struct ArticleFeatures{
      UUID itemId;                                       // attribute
      ArticleFeature[] features;                         // element
}
```

5.6.1.45 `ChoiceList`

Synopsis:

```
struct ChoiceList {
    string          propClass;
    string          propName;
    PropertyValue[] values;
}
```

Instances of the `ChoiceList` structure are used to associate the class and name of a property with its choice list (i.e. the sequence of `PropertyValue` instances). It contains the following fields:

`propClass`
: the property class

`propName`
: the property name

`values`
: a sequence of `PropertyValue` instances representing the current choice of values for the property[45]

5.6.1.46 `InsertInfo`

Synopsis:

```
struct InsertInfo {
    string  catalogId;
    string  catalogPackageId;

    string  articlePackageId;
    string  manufacturerId;
    string  seriesId;
    string  baseArticleNumber;
    string  variantCode;
    string  finalArticleNumber;
    string  ofmlVariantCode
}
```

The `InsertInfo` structure is used to collect arguments of the `insertOCDArticle` operation. It contains the following fields:

`catalogId`
: This field may be set to the identifier of the catalog containing the article as returned by the `catalogId` field of the `CatalogItem` structure (§5.5.1.31).

  This is an optional field. If the client is not able to, or does not want to specify a catalog ID, this field should be set to an empty string.

45 Unless the current value of the property is undefined it is always an element of the choice list.

`catalogPackageId`

This field may be set to the identifier of the OFML package containing the OFML catalog entry for this article as returned by the `catalogPackageId` field of the `CatalogItem` structure (§5.5.1.31).

This is an optional field. If the client is not able to, or does not want to specify the catalog package ID, this field should be set to an empty string.

`articlePackageId`

This field may be set to the OFML package ID of the package containing the product data of the article. If the article has been found in the catalog using the catalog service, it should be set to the value returned by the `articlePackageId` field of the `CatalogItem` structure (§5.5.1.31).

This is an optional field. If the client is not able to, or does not want to specify the article package ID, it should set this field to an empty string, and it **must** set the `manufacturerId` field to a valid value.

If this field is set to a non-empty string, then the identified OFML package must contain the product data for the article.

`manufacturerId`

This field may be set to the commercial manufacturer ID of the article. If the client is not able to, or does not want to specify the commercial manufacturer ID, this field should be set to an empty string.

If the `articlePackageId` field does not identify the OFML package containing the product data for the article, then the `manufacturerId` field must be set to a valid value. The Online Configurator searches for an OFML package that contains product data for the specified manufacturer, series (if specified), and base article number. If more than one such package exists, it is unspecified which package is actually used.

`seriesId`

This field may be set to the series ID of the article. If the client is not able to, or does not want to specify the series ID, this field should be set to an empty string.

`baseArticleNumber`

This field must be set to the base article number of the article. If the article has been found in the catalog using the catalog service, it should be set to the value returned by the `baseArticleNumber` field of the `CatalogItem` structure (§5.5.1.31).

`variantCode`

This field may be set to the variant code of the article, although it is not recommended to do so unless the product data of the article uses a key/value format for the variant code.

The variant code may be either partial or complete. If the product data of the article uses a key/value format for the variant code then the partial variant code may specify the value of any property. Otherwise a partial variant code may specify only an initial sequence of property values.

If the variant code is specified, it is used to determine the initial configuration of the article. If the client does not want to specify a variant code, it should set this field to an empty string.

`finalArticleNumber`

This element may be used instead of the variant code to specify the initial configuration of an article.

If both the variant code and final article number are specified (i.e. non-empty), the final article number is ignored.

If the base article number is not specified, and the final article number is specified and not ignored, the basket web service searches the OFML product data for an article matching the final article number. If no such article is found, the insert operation fails. If at least one article is found, the best matching article is used (i.e. the catalog ID, catalog package ID, article package ID and base article number of the best matching article are used as part of the insert info if not yet specified).

The details of the search algorithm, including selection of the best matching article, are implementation defined. However, the catalog ID, catalog package ID, article package ID, manufacturer ID and series ID, if specified as part of the insert info, are taken into account. The selected article is guaranteed to match these parameters, except for the series ID, which is only take as a hint due to limitations of the XCF catalog format and the fact that one OFML package can contain articles of multiple series.

Furthermore, search for a final article number requires that at least one of catalog ID, catalog package ID, article package ID or manufacturer ID is specified as part of the insert info.

Note that the mechanism must be enabled for for article packages supporting this feature by DSR feature `maySetFinalArticleSpec`.

`OfmlVarianCode`

If the element is specified[46], the element `basketArticleNumber` must be specified too, and the specified OFML variant code must be valid for the article to be inserted.

If the OFML variant code is specified together with the final article number and/or ordinary variant code, the OFML variant code is used to initialize the new article.

5.6.1.47 `ImageInfo`

Synopsis:

```
struct ImageInfo {
            string tag,
            string url,
            string[] attributes
            string itemId
}
```

Instances of `ImageInfo` are returned by the `getImages` (§5.6.3.42) operation to return information about images associated with a basket item. The fields of `ImageInfo` are as follows:

`tag`

The tag used to identify the image. This is either the tag specified with the `tag` option of the `getGeneratedImage` operation (5.6.3.40), or an empty string if the URL references the *PBK image* (see §5.6.3.42 for more information).

`url`

The value of this field is the URL that can be used to download the image. The URL is valid until the session that invoked the `getImages` operation returning this URL has been closed.

5.6.1.48 `ConfigDependentMediaInfo`

Synopsis:

```
struct ConfigDependentMediaInfo {
    int         weight;
    string      mediaType;
    string      format;
    string[]    modifiers;
    URL         url;
}
```

[46] An article number or variant code is specified if the element is present and has a non-empty value.

5.6.1.49 `ItemSelectionOptions`

Synopsis:

```
struct ItemSelectionOptions {
    boolean*    wholeComposite;
    boolean*    subItems;
    boolean*    parentItems;
    ViewId*     viewId;
}
```

Is used to augment the list of items selected by the `itemIds` parameter (if not empty).

`wholeComposite`

If the `wholeComposite` attribute is present and true, the operation iterates over all selected items, and for each basket sub-article item adds the main article item to the list of selected items if it is not already an element of this list. Then, the operation iterates again over the resulting list (including elements added by this step), and for each basket article item adds all sub-article items if they are not already part of the list.

`subItems`

If the `subItems` attribute is present and true, the operation iterates over the list of items selected thus far and adds all direct and indirect sub-items if they are not already an element of the list.

`parentItems`

If the `parentItems` attribute is present and true, the operation iterates over the list of items selected thus far and adds all direct and indirect parent items if they are not already an element of the list.

`viewId`

This field is ignored by most operations that take a parameter of type `ItemSelectionOptions`, or one of its sub-types. However, the operations `getAllItems`, `getItemProperties`, `getPriceCalculationSheet` and `getPriceCalculationSheets` use this field to select a basket view and interpret item IDs passed to the operation as view item IDs instead of basket item IDs.

Operations that use this field behave as before if the field is missing, has an empty value, or has the NIL-UUID as its value. Otherwise the field’s value must be the ID of an existing basket view.

5.6.1.50 `GetChoiceListOptions`

Synopsis:

```
struct GetChoiceListOptions (
                        boolean     convertSurcharge,
                        boolean     formatSurcharge,
                        boolean     discardSurcharge,
                        boolean?    HighResPropValueIcons,
                        boolean?    FetchPropValueImages,
                        boolean?    enableBooleanPropType
                               )
```

`convertSurcharge`

If true, convert surcharge from currency reported by OFML to currency used by basket. Default value is true.

`FormatSurcharge`

If true, append formatted surcharge to text reported for property values. Default value is true.

`DiscardSurcharge`

If true, don't report surcharge as surcharge field of complex type PropertyValue (if formatSurcharge is true, the formatted surcharge is still appended to the property value text). The default value is equal to the effective value of formatSurcharge.

The default values apply if no options are passed to aforementioned operations, or if they are passed but the corresponding option has not been set.

`highResPropValueIcons`

Controls whether fields smallIcon and largeIcon of type PropertyValue return URLs for the traditional low resolution material icons (if false) or may return URLs for high resolution material icons (if true).

If set to false (the default value), the behavior of EAIWS with regards do fields smallIcon and largeIcon does not change.

If set to true, and directories .../mat/ and .../mat/m exist, smallIcon, if non-empty, references a file in .../mat/s, and largeIcon, if non-empty, references a file in .../mat/m.

If set to true and at least one of .../mat/s and .../mat/m does not exist, both smallIcon and largeIcon, if non-empty, reference a file in .../mat.

`fetchPropValueImages`

Controls whether the new field image of type PropertyValue is absent (if false, the default value) or may reference a material image in directory .../mat/l (if true).

Field image is absent if option `fetchPropValueImages` is set to true but no corresponding image file is found. This differs from fields smallIcon and largeIcon, which are always present, but contain an empty string if the corresponding icon is not found.

`enableBooleanPropType`

The optional boolean field `enableBooleanPropType` enables boolean properties. Its default value is `false`, resulting in no change of behavior. Previously, boolean properties have been treated as properties of type `Numeric` with values one and zero for `true` and `false` when returned by operations `getArticleData`, `getChoiceList` and `getAllChoiceLists`, or zero and non-zero for `false` and `true` when set with operation `setPropertyValue`.

If `enableBooleanPropType` is set to `true`, the behavior for boolean properties changes as follows:

- Operation `getChoiceList` and `getAllChoiceLists` returns `true` and `false` instead of 1. and 0. in field `value` of type `PropertyValue`.

If option `enableBooleanPropType` is effectively `false`, operation `getChoiceList` returns the value of boolean properties as `1` and `0` instead of `1.` and `0.` in field `value` of type `PropertyValue`.

5.6.1.51 `GetAllItemsOptions`

Synopsis:

```
struct GetAllItemsOptions : ItemSelectionOptions {
    boolean *setArticleIds;
    boolean *geometryIds;
    boolean *positionNumbers;
    boolean *basketItems;
}
```

Can be used as argument of operation `getAllItems (§5.6.3.13).`

The optional attributes `setArticleIds` and `geometryIds` of type `boolean` control whether or not returned instances of complex type `BasketItem` contain set-article IDs (and set-article part IDs) and geometry IDs (if available).
The default value of these options is `false.`

`positionNumbers`

If the value of this field is `true` and field `viewId` contains the ID of a basket view then the basket items returned by operation `getAllItems` will return the position number of the returned view item.

The default value of this field is `true`.

`basketItems`

If the value of this field is `true` and field `viewId` contains the ID of a basket view then the basket items returned by operation `getAllItems` do not use field `basketItemIds` to return IDs of referenced basket items, but use field `basketItems` to return complete basket items.

Options `setArticleIds` and `geometryIds` may be used to control the content of these nested instances of `BasketItem.`

5.6.1.52 `DeleteItemsOptions`

Synopsis:

```
struct DeleteItemsOptions: ItemSelectionOptions {
    boolean*    subArticles;
}
```

The attributes from `ItemSelectionOptions` are used to augment the set of selected items passed to operation `deleteItems`.

`subArticles`

The boolean attribute `subArticles` of `DeleteItemsOptions`, if specified as `true`, enables the deletion of individual sub-articles of composite articles (if supported by the underlying OFML data).

5.6.1.53 `MoveItemsDirection`

Synopsis:

```
enum MoveItemsDirection {
    Up,
    Down,
    Indent,
    Unindent
}
```

The `MoveItemsDirection` enumeration is used to identify whether the specified items are to be moved up or down within their current folder, or are to be indented or unindented. The following enumeration values are defined:

`Up`

Move item up.

`Down`

Move item down.

`Indent`

Indent item.

`Unindent`

Undented item.

5.6.1.54 `MoveItemsResult`

Synopsis:

```
enum MoveItemsResult {
    Done,
    NothingToDo,
    NoConsecutiveItemSequence,
    NoPredecessor,
    NoSuccessor,
    HasSuccessor,
    NoGrandparent,
    CannotChangePlanningItemIndentation,
    NewParentHasInappropriateType
}
```

The `MoveItemsResult` enumeration is used to identify the results of the moveItem (§5.6.3.23) operation. If there are multiple possible return values, the first one from the list below is selected. The following enumeration values are defined:

`Done`

Operation successful, item structure changed.

`NothingToDo`

The specified set of items is empty.

`NoConsecutiveItemSequence`

The specified set of items is not a consecutive sequence of children of the same father.

`NoPredecessor`

The specified item sequence can not be moved up or indented because the first item of the sequence is also the first child of the parent item.

`NoSuccessor`

The specified item sequence can not be moved down because the last item of the sequence is already the last child of the parent item.

`HasSuccessor`

The specified item sequence can not be unindented because the last item of the sequence is not the last child of the parent item.

`NoGrandparent`

The specified item sequence can not be unindented because their current parent has no parent.

`CannotChangePlanningItemIndentation`

Planning items can not be indented or unindented.

`NewParentHasInappropriateType`

At least one of the items to indent is a folder and the new parent is not a folder or at least one of the items to indent is an article and the new parent is neither an article nor a folder.

5.6.1.55 `RelocateItemsOptions`

Synopsis:

```
struct RelocateItemsOptions {
        boolean *doNotModify;
        boolean *flat;
}
```

Is used to specify in which way the items should be moved by the method `relocateItems` (§5.6.3.24).

In general, the specified items are moved together with all their descendants (i.e. whole sub-trees are moved).

`flat`

The option flat (`false` by default) controls the behavior if one of the items to move is a descendant of another item to move (the ancestor). If `true`, the sub-tree rooted at the descendant will be moved to the father. If `false`, the descendant will be moved as part of the sub-tree rooted at the ancestor.

`doNotModify`

If option doNotModify (`false` by default) is true, the final step of moving the items is suppressed. Other than that, the operation behaves exactly the same way as with this option set to `false`, including returning the same result and throwing the same exceptions, unless they occur during the actual move of the items (which should not happen).

5.6.1.56 `RelocateItemsResult`

Synopsis:

```
enum RelocateItemsResult {
        Done,
        NothingToDo,
        CannotReparentPlanningItem,
        NewParentHasInappropriateType,
        IllegalStructuralChange
}
```

The operation `relocateItems` (§5.6.3.24) may return the following enumeration values:

`Done`

The items have been moved as specified.

`NothingToDo`

No item has been moved because the list of items to move was empty, or the items where already in the specified position and order.

`IllegalStructuralChange`

No item has been moved because doing so would result in an invalid basket item structure, or is just impossible. This may happen if the father or one of its ancestors, or the item identified by `beforeId` (if non-NIL and a child of the new father), is also included in the list of items to move.

`NewParentHasInappropriateType`

No item has been moved because at least one of the items to be moved is not allowed as a child of the new father/parent item.

`CannotReparentPlanningItem`

No item has been moved because doing so would require reparenting of a planning item, which is generally not allowed.

5.6.1.57 `OAPRotateObjectAction`

Synopsis:

```
struct OAPRotateObjectAction : OAPAction {
    Rotation rotation;
}
```

5.6.1.58 `OAPTranslateObjectAction`

Synopsis:

```
struct OAPTranslateObjectAction : OAPAction {
    Vector3 offset;
}
```

5.6.1.59 `OBKVersionInfo`

Synopsis:

```
final struct OBKVersionInfo{
                                   string  vendorKey;
                                   string  appKey;
                                   string  appVersion;
                                   string  bskXmlVersin;
                                   string? bskVersion;
                             }
```

The values of all fields in `OBKVersionInfo` are equal to the values of corresponding attributes in element <`versionInfo`> of BSK streams. While the values are expected to be non-empty, this isn't actually guaranteed (except for `bskVersion`, which is always non-empty if present).

All fields should be present in OBK files written by offline applications. In OBK files written by EAIWS, all fields except `bskVersion` should be present.

Field `obkVersionInfo` is used to return information about the version information contained in the currently loaded OBK file. The field is present if and only if at least one of the version information fields is non-empty.

5.6.1.60 `OperationMode`

Synopsis:

```
struct OperationMode {
    boolean*    subPositions;
    boolean*    mainArticles;
    boolean*    breakUpSetArticles;
    boolean     migrate
}
```

The flags in the operation mode parameter `opMode` control the behavior of some basket service operations. The documentation of these operations should contain more information.

In any case, all attributes of this type are optional. If missing, they default to false.

5.6.1.61 `CopyOptions`

Synopsis:

```
struct CopyOptions {
    string*     encoding;
    string*     suffix;
    boolean*    overwrite;
    boolean*    cut;
```

```
        boolean     omitPriceData
    }
```

Options to be used in copy (§5.6.3.51) operation.

`encoding`

This attribute specifies the encoding to be used for the OBX stream. The default value of this option is `UTF-8`.

`suffix`

This attribute specifies the suffix to be used in case the `uri` parameter is empty. If the `suffix` is not empty and does not start with a period character ('.'), a period is prepended to the suffix. The default value of this option is `.obx`.

`overwrite`

This attribute specifies whether the operation should fail if the file specified by the `uri` parameter already exists (overwrite=false) or overwrite the file. The default value of this option is `false`.

`cut`

This attribute

The `cut` option specifies whether or not the copied items are to be deleted once the OBX stream has been successfully written. The default value of this option is `false`.

`omitPriceData`

Possible values are true and false, with false being the default value.

5.6.1.62 `PricingProcedure`

Synopsis:

```
    struct PricingProcedure {
        string                      name;
        string                      displayName;
        PricingProcedureLine[]      lines;
        HalfWayRoundingMode         halfWayRoundingMode;
    }
```

Describes the pricing procedure (calculation scheme) and gives information about the lines used in the pricing procedure.

5.6.1.63 `PricingProcedureLine`

Synopsis:

```
    struct PricingProcedureLine {
        int                 level;
        int                 counter;
        LineType            lineType;
        string              conditionType;
        string              decription;
        int*                from;
        int*                to;
        LineInsertMode      insertMode;
        int                 maxOccurs;
        PrintControl        printControl;
        string[]            tags;
    }
```

Describes the pricing procedure line.

`Tags` Assign a particular semantic to a line. A tag may be assigned to multiple lines, and a line may be assigned multiple tags. The `PricingProcedureLine` contains zero or more `tags` elements of type `string`, each of them containing one tag.

`level` and `counter` describe the position of the calculation line in the calculation sheet. Whereas `counter` describes the position inside a `level`.

`from` and `to` describe the calculation line levels a condition line uses as input value. If no `from` value is set the condition line always uses the previos condition line value as input except it is of `LineType calculationBreak`.

`maxOccurs` desribes how often a manual condition can be added to the calculation.

5.6.1.64 `PricingProcedureDescription`

Synopsis:

```
struct PricingProcedureDescription {
    string     name;
    string     displayName;
}
```

Returned by operation `listPricingProcedures (§5.6.3.54)`.

5.6.1.65 `CalculationLine`

Synopsis:

```
struct CalculationLine {
    InactiveFlag[]  inactive;
    int             level;
    int             counter;
    string*         conditionType;
    string          description;
    Value*          amount;
    Quantity*       qtyRelation;
    Money*          value;
    CSValidationError groupError,
    CSValidationError error
    boolean         amountEditable;
    boolean        amountEdited,
    boolean         qtyRelationEditable;
    boolean        conditionRemovable,
    boolean        ValueEditable
}
```

Provides details of a calculation line.

The element inactive has zero or more occurrences. If there is at least one occurrence, the condition is inactive. The individual elements describe one or more reasons for the inactivity of the condition.

`level` and `counter` describe the position of the calculation line in the calculation sheet. Whereas `counter` describes the position inside a `level`.

If `amountEditable` is true a value can be set to this calculation line via setConditionAmount (§5.6.3.61).

If `qtyRelationEditable` is true the quantity relation can be changed via setQuantityRelation (§5.6.3.63).

The operation `getPriceCalculationSheet (§5.6.3.57)` may return calculation lines with elements `amount`, `qtyRelation` and `value` that do not have a non-empty `value` attribute, indicating an invalid amount, quantity relation or value.

ConditionRemovable: The value of this attribute is true if the line is a condition line that can be removed.

The information about addable and removable conditions is valid at the time the calculation sheet was returned by the server. Changes to the configuration of the article or to the calculation itself may invalidate this information.

Field `groupError` is present if and only if the line represents a group condition (this also includes header conditions) and there was an error validating the condition group.

Field `error` is present if there was an error validating the calculation line and the error is not suppressed.

An `error` is suppressed if:

- It can be ignored because the line represents an item condition with no valid data (as is the case, for example, after a manual condition has been added and before a condition amount has been set). Even though such lines have an invalid condition value, this doesn't matter as their outgoing value is always equal to their incoming value, and their condition value is ignored by reference ranges and aggregate conditions.

- The value is clearly a follow-up error. As of now the only errors considered follow-up errors are those of type CSInvalidValueError, and only if

    - The incoming value of a line is invalid because the outgoing value of the parent line is invalid,

    - The reference value of a line is invalid because the outgoing value, condition value, or subtotal value of a referenced line is invalid, or

    - The value of an item or aggregate subtotal is invalid because the incoming value is invalid, unless it is an item subtotal with no parent line. (This matters only for subtotals that compute their value as the difference of incoming value and base value, as is the case for subtotals that represent a margin, as all other subtotals simply use the base value as their value, and an invalid base value is covered by above two conditions.)

An user interface may choose to ignore errors (field error) of item calculations, or treat them as less severe, if they are of type CSInvalidValueError with result kind GroupCondBase and parameter kind CondBase, as such errors are probably follow-up errors of another error in the same item calculation or of an error in another item calculation with a condition in the same condition group.

5.6.1.66 `CalculationSheet`

Synopsis:

```
struct CalculationSheet {
    Quantity*            quantity;
    string               currency;
    Money*               netValue;
    Money*               tax;
    CalculationLine[]    lines;
    string[]?           addableConditions
}
```

The calculation sheets represents the calculation of the item passed to the function `getPriceCalculationSheet (§5.6.3.57)` respectivly the header calculation if no item id is passed.

If present, the attribute `tax` contains the sum of the values of all lines tagged with `TAX`. The `quantity` element is missing if the item ID passed to the operation `getPriceCalculationSheet (§5.6.3.57)` represents a folder item.

AddableConditions: Each element of this sequence contains the name of a condition type that can be added to this calculation sheet.

5.6.1.67 `ItemCalculationSheet`

Synopsis:

```
struct ItemCalculationSheet {
    string           id;
    CalculationSheet sheet;
}
```

5.6.1.68 `HalfWayRoundingMode`

Synopsis:

```
enum HalfWayRoundingMode {
    Even,
    Up
}
```

Rounding mode of the calculation.

`Even`
: The system will round toward the nearest neighbor. In case of equidistant neighbors the system will round towards the even neighbor. This rounding mode is also called Banker's Rounding.

`Up`
: With this rounding mode the system will round towards the nearest neighbor unless both neighbors are equidistant, in which case the system will round up (away from zero).

5.6.1.69 `LineInsertMode`

Synopsis:

```
enum LineInsertMode {
    Always,
    Manual,
    Auto
}
```

Manually inserted condition lines may be manually inserted by the user one line at a time until the maximum number of lines for the condition type has been reached. For each such condition line in an item calculation there is a corresponding line with the same level and counter in corresponding aggregate calculations (for item groups, set-articles, and the document).

Automatically inserted condition lines are inserted automatically by the system whenever the access method of the condition type returns at least one condition record. If the condition type supports multiple results, and the access method returns multiple condition records, multiple lines are inserted into the item calculation. In aggregate calculations, however, there is only one condition line representing the accumulated condition lines in item calculations for the same condition type.

5.6.1.70 `LineType`

Synopsis:

```
enum LineType {
    Condition,
    Subtotal,
    Text,
    CalculationBreak
}
```

Possible pricing procedure line types:

`Condition`

Condition line. Can show prices coming from product data, modify a price or represent a tax.

`Subtotal`

Sum of all previous condition lines till last subtotal or calculation break.

`Text`

A text line displays an optional description, but no values. The outgoing value of a text line is equal to its incoming value.

`CalculationBreak`

A calculation break line is like a text line, but prevents propagation of the ‘inactive due to subsequent price’ flag.

5.6.1.71 `PrintControl`

Synopsis:

```
enum PrintControl {
    Never,
    PositionAlways,
    PositionNonZero,
    PositionChanged,
    PositionChangedNonZero,
    HeaderAlways,
    HeaderNonZero,
    HeaderChanged,
    HeaderChangedNonZero
}
```

Defines when to print a condition.

`Never`

Condition should never be shown on print-out.

`PositionAlways`

Always show condition on print-out of article.

`PositionNonZero`

Print article condition if not zero.

`PositionChanged`

Print article condition if changed by user.

`PositionChangedNonZero`

Print article condition if changed by user and not zero.

`HeaderAlways`

Always show header condition on print-out of article.

`HeaderNonZero`

Print header condition if not zero.

`HeaderChanged`

Print header condition if changed by user.

`HeaderChangedNonZero`

Print header condition if changed by user and not zero.

5.6.1.72 CalculationRule

Synopsis:

```
enum CalculationRule {
    Invalid,
    Percent,
    FixedAmount,
    Quantity,
    GrossWeight,
    NetWeight,
    Volume,
    Rounding
}
```

5.6.1.73 ConditionType

Synopsis:

```
struct ConditionType {
    string               name;
    string               description;
    string*              accessMethod;
    ConditionClass       conditionClass;
    ConditionSign        conditionSign;
    CalculationRule      calculationRule;
    RoundingRule         roundingRule;
    boolean              groupCondition;
    boolean              headerCondition;
    ConditionEditMode    editMode;
    boolean              amountEditable;
    boolean              quantityRelationEditable;
    boolean              valueEditable;
    boolean              calculationRuleEditable;
    boolean              conditionDeletable;
    boolean              deferredCurrencyConversionEnabled;
}
```

Describes the details of a `ConditionType`.

Currently available `accessMethod`:

`BPR00`
: (Base) sales price coming from OFML data

`BPR01`
: (Base) purchase price coming from OFML data

`VAPR00`
: sales variant prices coming from OFML data

`VAPR01`
: purchase variant prices coming from OFML data

`PR00`
: sales price coming from OFML data (BPR00 + VARP00)

`PR01`
: purchase price coming from OFML data (BPR01 + VARP01)

`TAX00_<type>`
: tax value from OFML data. If tax type is not supported by currently selected tax scheme it gives an

invalid value.

`TAX01_<type>`

tax value from OFML data. If tax type is not supported by currently selected tax scheme it gives no value.

`RND00`

rounding interval in document currency

If `amountEditable` is `true` the amount or percentage value can be changed.

If `quantityRelationEditable` is `true` the base quantity of an amount can be edited.

If `deferredCurrencyConversionEnabled` is `true` the currency conversion is done after multiplication with quantity.

If `valueEditable` is `true` the value can be changed.

5.6.1.74 `ConditionClass`

Synopsis:

```
enum ConditionClass {
    Price,
    PriceModifier,
    Tax
}
```

Defines the class a condition belongs to.

`Price`

Price coming from the product data.

`PriceModifier`

Condition which modifies a price. E.g. a discount or surcharge condition.

`Tax`

,
Tax condition, e.g. VAT.

5.6.1.75 `ConditionSign`

Synopsis:

```
enum ConditionSign {
    NoRestriction,
    Negative,
    Positive
}
```

Defines the sign of a condition.

5.6.1.76 `ConditionEditMode`

Synopsis:

```
enum ConditionEditMode {
    Never,
    IfCondRecFound,
    IfCondRecNotFound,
    ManualWithDefault,
    Manual
}
```

Defines the edit mode of a condition.

`Never`
: Condition cannot be edited.

`IfCondRecFound`
: Condition can be edited when found.

`IfCondRecNotFound`
: Condition can be edited when not found.

`ManualWithDefault`
: Condition can be edited and has a default value.

`Manual`
: , Condition can be edited.

5.6.1.77 `RoundingRule`

Synopsis:

```
enum RoundingRule {
    NoRounding,
    RoundToNearest,
    RoundUp,
    RoundDown
}
```

Defines the rounding rule of a condition.

`NoRounding`
: No rounding.

`RoundToNearest`
: The system will round toward the nearest neighbour. In case of equidistant neighbours the system will round towards the even neighbour. This rounding mode is also called Banker’s Rounding.

`RoundUp`
: With this rounding mode the system will round towards the nearest neighbour unless both neighbours are equidistant, in which case the system will round up (away from zero).

`RoundDown`
: ,
This rounding mode rounds towards the nearest neighbour unless both neighbours are equidistant. In that case the system will round down (towards zero).

5.6.1.78 `TaxSchemeDescription`

Synopsis:

```
struct TaxSchemeDescription {
    string  identifier;
    string  country;
    string  region;
    string  variantName;
}
```

Short information about available tax schemes. Result of operation `listTaxSchemes` (§5.6.3.66).

5.6.1.79 `TaxScheme`

Synopsis:

```
struct TaxScheme {
    string          identifier;
    string          country;
    string          region;
    string          variantName;
    string          currency;
    TaxType[]       taxTypes;
    OrderedTax[]    orderedTaxes;
}
```

Detailed information about tax scheme. Result of operation `getTaxScheme` (§5.6.3.67).

`TaxTypResolveURIsOptionse`

Synopsis:

```
struct TaxType {
    string          identifier;
    string          rateUnit;
    string          abbreviation;
    string          name;
    TaxCategory[]   taxCategories;
}
```

The value of attribute `rateUnit` of complex type `taxType` is either `%` in case of relative taxes, or `<cy>/<uom>`, where `<cy>` is a currency code and `<uom>` is an UNECE unit of measure. With the current implementation of pricing procedures, both weight and volume units are meaningful.

5.6.1.80 `OrderedTax`

Synopsis:

```
struct OrderedTax {
    string  typeId;
    int     order;
}
```

5.6.1.81 TaxCategory

Synopsis:

```
struct TaxCategory {
    string      identifier;
    string      name;
    decimal*    rate;
    boolean     rateReadOnly
}
```

5.6.1.82 TaxInfo

Synopsis:

```
struct TaxInfo {
    string  taxType
    string  taxCategory;
}
```

Returned by the operation `getTaxInformation` (§5.6.3.70). Represents the known tax information for an article item. The tax information consists of a possibly empty list of pairs of tax type and tax category identifiers (like `VAT` and `reduced_rate`), specifying the tax category to be used by this article for the tax type.

5.6.1.83 OFMLUpdateState

Synopsis:

```
enum OFMLUpdateState {
    Unknown,
    UpToDate,
    Updatable,
    Migratable,
    Invalid,
    NoCatalog,
    MultipleCatalogs
}
```

The update state determined by EAIWS for a composite article item depends on the configuration of the article item, the set of registered OFML catalogs and the product data referenced by these catalogs, and a possibly specified subset of OFML catalogs.

The following update states are defined:

`Unknown`
: The update state has not been determined yet. See operation `getItemProperties (§5.6.3.32)` for more information.

`UpToDate`
: The data of the basket article item has been copied by this session from an OFML article item. Thus it is known that the current configuration of the basket article item is supported by at least one of the registered OFML catalogs, and that EAIWS should always be able to re-instantiate an OFML article with exactly the same configuration.

`Updatable`
: Catalog selection resulted in exactly one OFML catalog, and it has been determined that the product data referenced by this catalog allows the re-instantiation of an OFML article such that the configuration of the OFML article is exactly the same as the configuration of the basket article item.

`Migratable`

Like `Updatable`, except that the variant code of the OFML article item may differ from the original variant code stored at the basket article item.

`Invalid`

Catalog selection resulted in exactly one OFML catalog, and it has been determined that the product data referenced by this catalog does not contain the required OFML article (there is no OFML article with the same base article number).

`NoCatalog`

Catalog selection did not find a matching OFML catalog.

`MultipleCatalogs`

Catalog selection found multiple matching OFML catalogs.

The process of 'catalog selection', as used above in the description of individual update states, determines zero or more OFML catalogs that contain the same OFML package as specified by the program ID stored for the basket article item (i.e. a package with the same OFML manufacturer and program IDs).

The 'updateBasketArticles' operation described below allows the specification of a set of catalog IDs. If this set is not empty, it is used to further restrict the set of catalogs determined above (catalogs whose catalog ID is not contained in the set of specified catalog IDs are removed).

If, at this point, the set of catalogs contains more than one catalog, a catalog profile ID is stored at the article, and the catalog profile ID is equal to the catalog ID of one of the catalogs selected so far, all catalogs other than the catalog identified by this catalog ID are moved from the set of selected catalogs.

The remaining set of catalogs is the result of catalog selection as used in update state descriptions.

(The process of catalog selection is still subject of consideration and may be modified in future versions of EAIWS.)

5.6.1.84 `UpdateBasketArticleResult`

Synopsis:

```
struct UpdateBasketArticleResult {
    BItemId itemId;
    OFMLUpdateState updateState;
}
```

This type has two (required) attributes: `itemId` and `updateState`. An element of this type is returned by the `updateBasketArticles` (§5.6.3.80) operation for each basket main article item the operation operated upon.

`itemId`

The itemId is the ID of the basket main article item.

`updateState`

The `updateState` is the update state of the (whole composite) article item after the operation `updateBasketArticles`. Thus, if the article item has been updated or migrated, the update state is `UpToDate`.

5.6.1.85 `UpdateBasketArticlesOptions`

Synopsis:

```
struct UpdateBasketArticleOptions : ItemSelectionOptions {
    boolean        update;
    boolean        migrate;
    boolean        recalculate;
```

```
        date         priceDate
    }
```

This type extends the type `ItemSelectionOptions`. Additional optional boolean attributes are `update` and `migrate`. These attributes can be used to control whether or not the operation should update and/or migrate articles, or just determine the update state.

If the attribute `recalculate` is set to `true`, the calculations of selected article items not updated or migrated are recalculated. The calculations of updated or migrated article items are always recalculated. Recalculation includes execution of access methods. Note that, actually, the `updateBasketArticles (§5.6.3.80)` operation just invalidates the calculations. Recalculation happens on demand the next time affected calculation data is requested.)

Selection of article items for recalculation, updated or migrate works as followign: if one or more articles of a composite article have been selected, the behavior depends on whether or not the main article is among the selected articles, and (if not), whether or not the `wholeComposite` flag is set. In any case, either all articles of a composite article are recalculated, migrated or updated, or none.

The default value of `update` is `true`, whereas the default value of `migrate` and recalculate is `false`.

5.6.1.86 `DisplayText(Basket)`

Synopsis:

```
    struct DisplayText extends string (string lang)
```

5.6.1.87 `ExchangeRate`

Synopsis:

```
    struct  ExchangeRate {
        string  currency;
        decimal       rate;
    }
```

The attribute `currency` contains the currency code. The attribute `rate` contains the exchange rate. If the attribute `rate` is missing, the exchange rate is unknown or invalid.

5.6.1.88 `Vector2`

Synopsis:

```
    struct  Vector2 {
        double  x;
        double  y;
    }
```

5.6.1.89 `Vector3`

Synopsis:

```
    struct  Vector3 {
        double  x;
        double  y;
        double  z;
    }
```

This type is used to represent a position (through a location vector) or a rotation axis.

In the coordinate system assumed by the basket web service interface, the x-axis points to the right, the y-axis points up, and the z-axis points to the front.

5.6.1.90 `Rotation`

Synopsis:

```
struct  Rotation {
    double  angle;
    Vector3       axis;
}
```

The axis parameter is a vector that indicates the direction of an axis of rotation, and the angle describes the magnitude of the rotation about the axis. The rotation occurs in the sense prescribed by the right-hand rule.

The angle is measured in radian. The axis, when returned from EAIWS, is a unit vector unless the angle is a multiple of 2pi. An axis passed to EAIWS does not need to be a unit vector, but |axis| must be non-null unless the angle is zero.

5.6.1.91 `ComposableGeometryProperties`

Synopsis:

```
struct  ComposableGeometryProperties {
    BItemId       itemId;
    boolean       enabled;
    Vector3       position;
    Vector3       scale;
    Rotation      rotation;
    string        geometry;
}
```

`itemId`

The ID of the basket item having this set of composable geometry properties. This field is used if and only if the composable geometry properties are part of item properties returned for a view item.

`enabled`

This optional property controls whether or not the composable geometry item, when used as a set-article part, will be considered or ignored during composition of the set-article geometry.

`position`

This optional property indicates where the geometry of the composable geometry item is to be positioned when the geometry of the set-article is composed. The default value of this property is x=0, y=0 and z=0.

`scale`

Operation `getItemProperties (§5.6.3.32)` returns this element for composable geometry items with a scalable geometry (only user article items right now). Its absence indicates that the geometry is not scalable.

Operation `setItemProperties (§5.6.3.33)` ensures that `scale.x, scale.y` and `scale.z` are finite (i.e. neither infinite nor NaN). Other than that the operation ignores this element unless the item is a composable geometry item[47]. Use with a basket main article item causes an error, even if the scaling factor is one in each direction. Thus user article items are the only items that let the client specify scaling factors for the geometry.

The scaling factors are applied (or are supposed to be applied) to the geometry before it is rotated and translated.

`rotation`

This optional property indicates the rotation of the geometry of the composable geometry item when the geometry of the set-article is composed. Rotation occurs around the origin of the geometrie's local coordinate system. The default value of this property indicates no rotation.

[47] Basket main article items and user article items are composable geometry items.

`geometry`

This optional property can be used with user article items to store the URI of an imported or generated geometry.

If the item is a user article item, the item property `enabled` has been set to true, the item property `geometry` contains a valid geometry URI, and the user article item is part of a set-article item, then the specified geometry becomes part of the composite geometry of the set-article item.

Supported geometry formats are EOX, GFX, DWG, DXF, OBJ, GEO and 3DS. Import of materials is supported for EOX, GFX, DWG and DXF only.

If the `getItemProperties` operation returns composable geometry properties, then the value of the `geometry` element will be either an empty string or an URI with scheme `imp` or `gen`.

If the `setItemProperties` operation encounters a non-empty value for the `geometries` element, the operation fails without modifying any properties if:

- this value does not adhere to the URI syntax
- is a relative URI
- has a scheme other than `imp` or `gen`
- is an URI not known by the current session (i.e. has not been imported imported or generated by the current session)
- if the item whose properties are to be set does not allow the explicit specification of a composable geometry part.

5.6.1.92 `CondGroupSelectionOptions`

Synopsis:

```
struct CondGroupSelectionOptions : ItemSelectionOptions {
    boolean? ExcludeInactiveItems;
    boolean? IncludeInactiveItems;
    boolean? groupPseudoArticleParts
}
```

Options `excludeInactiveItems` and `includeInactiveItems` are mutually exclusive. Operations fail with a `BasketServiceFault` if both are set to `true`. Other than that the values are ignored if the set of item IDs passed to the operation and the specified `ItemSelectionOptions` do not result in construction of a group calculation.

If both options are unspecified, the default values of `excludeInactiveItems` and `includeInactiveItems` are `true` and `false`, respectively. Otherwise, the default value of either option is `false`.

If `excludeInactiveItems` and `includeInactiveItems` are both `false`, group calculations ignore item calculations of inactive items (i.e. alternative positions) if and only if the group contains at least one active item.

If `excludeInactiveItems` is `true`, group calculations always ignore calculations of inactive items.

If `includeInactiveItems` is `true`, group calculations do not ignore calculations of inactive items, i.e. group condition values are computed as the sum of item conditions values of both active and inactive items.

The option `groupPseudoArticleParts` is used by all operations with a parameter of type `CondGroupSelectionOptions` or `GetPriceCalculationSheetOptions` to a) enforce the use of group calculations for all pseudo articles and b) add all sub-articles of pseudo-articles to group calculations referencing a pseudo article.

The default value of this option is `false` (traditional behavior).

If this option is `true`, the behavior of these operations is changed as follows:

- If the operation operates on or returns a single calculation, and the effective set of selected items consists of a single pseudo article, the operation enforces the use of a group calculation instead of an item calculation.
- Construction of group calculations referencing a pseudo article includes all (not just direct) sub-article items of the pseudo article item (applies to all pseudo articles, even those which are sub-articles themselves).
- Operation `getPriceCalculationSheets` (§5.6.3.57) returns one calculation for each element of the effective set of selected items. Depending on the item type the behavior is as follows:
  - non-pseudo article: construction of an item calculation
  - pseudo article: construction of a group calculation referencing all sub-articles (independent of their position in the basket or view structure)
  - folder: construction of a group calculation referencing all article items that are descendants of the folder item in the basket or view structure

5.6.1.93 `ColumnType`

Synopsis:

```
enum ColumnType {
    Undefined,
    Builtin,
    Text,
    Number,
    Boolean,
    Image,
    EClass
}
```

This enumeration type defines the type of user defined columns. It places certain restriction on the data that can be stored in fields of these columns.

`Undefined`

This type should never occur.

`Builtin`

Used for predefined columns that cannot be modified or deleted, have well defined column IDs, and are further identified by enumeration type `BasketItemAttrId`.

`Text`

Allows storage of arbitrary character sequences.

`Number`

Allows storage of character sequences that can be parsed as decimal floating point numbers. After removal of leading and trailing white space, values stored in columns of this type must be either empty strings or must match one of the following regular expressions (ignoring case):

[+-]?inf(inity)?

[+-]?nan

[+-]?[0-9]+(\.[0-9]*)?(e[+-]?[0-9]+)?

[+-]?\.[0-9]+(e[+-]?[0-9]+)?

If they match one of these regular expressions, they are converted to an IEEE 754-2008 Decimal64, the formatted as a string. The result of formatting matches one of the regular expressions listed above. Other than that, the result is implementation-defined.

Note that two numbers with identical values may end up with different string representations due to different quantum exponents or, if data is exchanged between different applications, due to different formatting rules.

Also note that P-BK and derived applications use Decimal128 Instead of Decimal64, and other applications may use still other internal representations. This is acceptable as long as they are prepared to deal with any character sequence matching one of the regular expressions listed above.

`Boolean`

Allows storage of either "0" or "1" (leading and trailing white space will be removed by operation `setItemFields (§5.6.3.90)`).

`Image`

Allows storage of an URI reference to an image. Once leading and trailing white space has been stripped, the remaining character sequence must be an empty string or parsable as an absolute URI with a scheme of either `imp`, `gen`, `file` or `http`. Furthermore, URIs with scheme `imp` or `gen` must be opaque (i.e. not hierarchical), whereas URIs with scheme `file` or `http` must be convertible to an URL.

No matter what URI scheme is used, there is no requirement that the identified resource actually exists. If it exists, it should be an image file, preferably `JPEG` or `PNG`.

Once an URI passed validation, it is converted to US-ASCII before stored in the field (RFC 2396 is limited to US-ASCII).

`EClass` (deprecated)

Had been used to store eCl@ss classifications

5.6.1.94 `BasketItemAttrId`

Synopsis:

```
enum BasketItemAttrId {
    Undefined,
    UserDefined,
    Manufacturer,
    Series,
    ArticleNumber,
    Description,
    Quantity,
    SinglePurchasePrice,
    TotalPurchasePrice,
    SingleSalesPrice,
    TotalSalesPrice,
    SingleNetPrice,
    TotalNetPrice,
    SingleGrossPrice,
    TotalGrossPrice,
    Position,
    CatalogImage,
    GeneratedImage,
    ArticleImage
}
```

This enumeration type defines an identifier for predefined columns. The value `Undefined` should never occur. The value `UserDefined` is used for user-defined columns. All other IDs identify a predefined column.

For each predefined column ID, each basket has exactly one column. Fields of predefined columns can be read, but not written. Predefined columns have a well known column ID.

`Manufacturer`

Column ID: 69ec3fa0-795a-11d6-9c21-00e029099a4b

This column contains the commercial manufacturer ID, if available, or an empty string.

`Series`

Column ID: 6d302258-795a-11d6-9c21-00e029099a4b

This column contains the series ID, if available, or an empty string.

`ArticleNumber`

Column ID: 71803794-795a-11d6-9c21-00e029099a4b

This column contains the base article number, if available, 0or an empty string.

`Description`

Column ID: 745802e4-795a-11d6-9c21-00e029099a4b

This column contains the short description of article items or the label of the item for all other item types. The short description of article items is determined based on the currently configured list of effective product data languages.

`Quantity`

Column ID: 76eda34c-795a-11d6-9c21-00e029099a4b

For items with a valid quantity (article items with a finite quantity) this column contains the quantity formatted as a decimal number with at least two and at most three fraction digits. Otherwise it contains an empty string.

Three fraction digits are used if the exact value of the quantity cannot be expressed using only two fraction digits.

Decimal separator, grouping separator and grouping size are determined based on the session's current locale.

`SinglePurchasePrice`

Column ID: 7d21a60a-795a-11d6-9c21-00e029099a4b

For article items, this column contains the purchase price. Otherwise, it contains an empty string.

See below for information about formatting of prices.

The purchase price of an OFML article item is usually the purchase price provided by the OFML product database. If, however, the article item was part of a project produced by some other application (like P-BK), an explicitly specified gross purchase price may be used instead.

`TotalPurchasePrice`

Column ID: 7fe484fc-795a-11d6-9c21-00e029099a4b

For article items, this column contain the product of purchase price and quantity.

`SingleSalesPrice`

Column ID: 82efa014-795a-11d6-9c21-00e029099a4b

For article items, this column contains the sales price. Otherwise, ti contains an empty string.

See below for information about formatting of prices.

The sales price of an OFML article is the sales price provided by the OFML product database.

`SingleNetPrice`

Column ID: 8831eeec-795a-11d6-9c21-00e029099a4b

`SingleGrossPrice`

Column ID: 8dcf0592-795a-11d6-9c21-00e029099a4b

The current implementation uses the value of the single sales price column as the value of these columns. This may be changed in future versions of EAIWS.

`TotalSalesPrice`

Column ID: 8541eb1a-795a-11d6-9c21-00e029099a4b

For article items, this column contains the product of sales price and quantity.

`TotalNetPrice`

Column ID: 8b51fbe4-795a-11d6-9c21-00e029099a4b

`TotalGrossPrice`

Column ID: 90895bc0-795a-11d6-9c21-00e029099a4b

The current implementation uses the value of the total sales price column as the value of these columns. This may be changed in future versions of EAIWS.

`Position`

Column ID: 81d12edc-853a-11d6-9c21-00e029099a4b

With the current implementation this column always contains an empty string. This is because position numbers for items are provided by the basket views, not the basket itself. Basket views, however, are not supported yet.

`CatalogImage`

Column ID: 9c9ea8ea-20ce-11d7-9c21-00e029099a4b

In case of OFML article items, this column contains an image URI for an image from an XCF or OAS catalog if such an image has been found for this article. Otherwise, it contains an empty string.

Unless resolved, the URI will usually be a file URL, although there is no guarantee this won't change in future versions of EAIWS.

`GeneratedImage`

Column ID: 6a66160c-da62-11d8-b9d6-00e081513ada

In case of OFML article items, this column contains an image URI for an image dynamically rendered for the current configuration of this article item unless rendering of images has been suppressed or disabled. Otherwise, it contains an empty string.

The images contained in this column are the sames images as returned with an empty tag by the `getImages (§5.6.3.42)` operation. However, other than the `getImages (§5.6.3.42)` operation, callers of operation `getItemFields` (§ 5.6.3.89) can control if generated images should be fetched from the global cache if not already referenced by the item, or, if not found in the global cache, should be generated. See also GetItemFieldsOptions (§ 5.6.1.98) for more information.

If an image needs to be generated, the rendering settings used are the same as those used by the `getGeneratedImage (§5.6.3.40)` operation when invoked with no options. Other than P-BK and derived applications, the current implementation makes no attempt to reuse the settings from the last image generated (for a different configuration) for this position.

`ArticleImage`

Column ID: 73bd68f4-da62-11d8-b9d6-00e081513ada

For OFML, OCD and user article items, this column contains the value of the `GeneratedImage` column, if not empty, and the value of the `CatalogImage` column otherwise. For all other basket items, it contains an empty string.

Formatting of prices:

Finite prices are formatted as a fixed point decimal number followed by a SPACE (U+0020) and the ISO 4217 currency code. As a special case, if the amount is zero, the SPACE and currency code may be missing.

The amount is formatted using either the default number of fraction digits of the currency, or one more fraction digit if the amount cannot be expressed exactly with the default number of fraction digits.

Decimal separator, grouping separator and grouping size are determined based on the session's current locale.

The formatting of infinite and invalid prices is highly locale-dependent and subject to change.

5.6.1.95 `ColumnId`

Synopsis:

```
typedef UUID ColumnId;
```

ColumnId uniquely identifys a basket column. Predefined columns always have the same ID. See `BasketItemAttrId` (§5.6.1.94) for the values of predefined column IDs.

5.6.1.96 `BasketColumn`

Synopsis:

```
struct  BasketColumn {
                        string name,
                        string title,
                        string defaultValue
                        string id,
                        ColumnType type,
                        BasketItemAttrId itemAttrId,
                        boolean removable,
                        boolean editable,
                        string defaultColumn,
                        int minWidth,
                        double weight,
                        boolean readOnly
                    }
```

This complex type describes a single basket column. It has the following attributes and elements:

`id` (attribute)

The UUID that identifies the column. Predefined columns always have the same ID. See BasketItemAttrId (§5.6.1.94) for the values of predefined column IDs.

`type` (attribute)

The type of user-defined columns, or `Builtin` for predefined columns.

`itemAttrId` (attribute)

Type of predefined columns, or `UserDefined` for user-defined columns.

`removable` (attribute)

True if the column can be removed with operation `removeBasketColumns (§5.6.3.86)`

`editable` (attribute)

True if the column configuration can be changed with operation `setBasketColumnProperties (§5.6.3.88)`

`defaultColumn` (attribute)

The ID of a column that provides the default value for this column

`minWidth` (attribute)

Exists for P-BK compatibility; In P-BK, this property contains the minimum width of the article table column in pixel. If used by other applications, the value should be interpreted relative to the font size used in the article table. The actual column width should be chosen so it allows display of about the same number of characters as an 8 point Arial font assuming 96 DPI and the given number of pixels.

`weight` (attribute)

Exists for P-BK compatibility; In P-BK, the value of this property determines the initial width of columns relative to each other.

`readOnly` (attribute)

True if the column is read-only (i.e. operations `setItemFields` (§5.6.3.90) and `resetItemFields (§5.6.3.91)` cannot be used with this column)

`name` (element)

The name of the column

`title` (element)

The title of the column to be displayed in the header of the article table

`defaultValue` (element)

Optional default value. Default values may contain variable references. See description of complex type `ItemField` (§5.6.1.97) for more information about variable references.

Except for the `id` attribute, all attributes and elements are optional. For more information, see documentation of operations `getBasketColumns` (§Fehler: Verweis nicht gefunden), `addBasketColumns` (§5.6.3.87), and `setBasketColumnProperties` (§5.6.3.88).

5.6.1.97 `ItemField`

Synopsis:

```
struct  ItemField {
        BItemId             itemId;
        ColumnId            columnId;
        string              data;
}
```

This complex type holds the date of a single field (the intersection of basket column and item). It has the following attributes and elements:

`itemId` (attribute)

the ID of the basket item

`columnId` (attribute)

the ID of the basket column

`data` (element)

the data stored in the field, or an empty string if no data has been stored. The descriptions of enumeration types `ColumnType (§5.6.1.93)` and `BasketItemAttrId` (§5.6.1.94) contain information about the format of data stored in user-defined and predefined columns.

All attributes and elements are required.

Variable References:

The default values of user defined columns, as well as the values set for fields of user defined columns of type `Text` and `EClass`, may contain zero or more variable references embedded in their value. Values set for fields of user defined columns of type `Number` may consist of a single variable reference.

Variable references start with a dollar sign (U+0024), immediately followed by an left parenthesis (U+0028). They end with the next right parenthesis (U+0029).

Most variable references can only be used for items of type `Article`, `Aggregate`, `PartialPlanning`, `UserArticle` and `OCDArticle.` The `Label` reference can be used for any item type.

`Label`

the label of the item, corresponding to the `label` element of complex type `ItemProperties` (§5.6.1.20)

`ManuName, ManufacturerName`

the manufacturer name

`ManuId, ManufacturerId`

the commercial manufacturer id

`SeriesName`

the series name

`SeriesId`

the (commercial) series ID

`ArtNr`

the default article number; Depending on information provided by OFML data, this is either the base or final article number. For user article items with a non-empty final article number it is the final article number, and the base article number otherwise.

`BaseArtNr`

the base article number

`FinalArtNr`

the final article number

`VarCode`

the variant code

`Text`

the default description of the article; Depending on information provided by OFML data, this is either the short description, the long description, or the concatenation of both, delimited by a single line feed (U+000A) character.

For user article items with a non-empty long description, the long description is used, otherwise the short description.

`ShortText`

the short description of the article

`LongText`

the long description of the article

`VarText, VariantText`

the feature description of the article; Whereas short and long description solely depend on the article (base article number), the feature description depends on the current configuration of the article.

`VarCodes, VariantCodes`

coded features of OFML articles; The value is a list of key/value pairs, where subsequent key/value pairs are separated by a single line feed (U+000A) character, and key and value are separated by an equals sign (U+003D). The key is always non-empty, the value may be empty.

The encoded features correspond the the features returned by OFML method getAllArticleFeatures(NULL).

`CatImage, CatalogImage`

an empty string or an URI for the catalog image. See `CatalogImage` value of enumeration `BasketItemAttrId (§5.6.1.94)` for more information.

`GenImage, GeneratedImage`

an empty string or an URI for the generated image. See `GeneratedImage` value of enumeration `BasketItemAttrId (§5.6.1.94)` for more information.

`ArtImage, ArticleImage`

an empty string or an URI for the article image. See `ArticleImage` value of enumeration `BasketItemAttrId (§5.6.1.94)` for more information.

`SinglePurchasePrice`
`SingleSalesPrice`
`SingleNetPrice`
`SingleGrossPrice`
`TotalPurchasePrice`
`TotalSalesPrice`
`TotalNetPrice`
`TotalGrossPrice`

the named price of the article; For more information, see documentation of corresponding value of enumeration `BasketItemAttrId (§5.6.1.94).`

`PackInfo:<suffix>`

packaging information

<suffix> consists of the name of a packaging property, optionally followed by a conversion specification enclosed in left and right square brackets (U+005B, U+005D).

The following table contains the supported packaging property names and the default unit used to report the value:

Width	MTR
Height	MTR
Depth	MTR
Volume	MTQ
TareWeight	GRM
NetWeight	GRM
ItemsPerPackUnit	C62
PackUnitsPerArticle	C62

The following name is supported for backward compatibility:

TaraWeight	GRM

The conversion specification, if present, must consist of a preferred unit code (UNECE unit of measure), optionally followed by the specification of a precision. The precision specification must consist of a colon (U+003A), an optional minus sign (U+002D), and a non-empty sequence of decimal digits (U+0030..U+0039). The precision is clamped to range [0, N], where N is the number of significant decimal digits supported by the used IEEE 754-2008 decimal format (the current implementation uses N=16).

The variable reference expands to an empty string if the position does not have information about the named packaging property, the packaging property name is unknown, or the conversion specification is invalid.

Otherwise, if the variable reference contains a conversion specification, an attempt is made to convert the property value to the specified unit. If conversion fails (as would be the case for conversion from MTR to GRM), the original value will be used.

The amount of the value will then be converted as a fixed point decimal number. If the variable reference specifies a precision, then exactly as many fraction digits will be produced. If the variable reference does not specify a precision, then there will be exactly as many faction digits as necessary to preserve the value. If no fraction digits are produced, there will be no decimal separator either.

Formatting is done according to the rules of the session's locale. The exact rules for formatting, including the use of grouping, is implementation-defined.

Finally, if the unit is not 'C62', then a single space character (U+0020) and the unit will be appended to the formatted value.

`Tax:<suffix>`

either a particular tax or the position total of all taxes;

<suffix> consists of the name of the pricing procedure and the name of the condition type, separated by colon (U+003A).

If the name of the pricing procedure consists of a single asterisk (U+002A) then it matches the name of an active pricing procedure if and only if there is exactly one active pricing procedure. A leading asterisk followed by a colon may be omitted.

If the condition type name consists of a single asterisk, then the total of all tax lines of the selected pricing procedure is used. If at least one line of the pricing procedure is tagged with `TAX` then all lines tagged with `TAX` are considered tax lines. Otherwise, all condition lines with condition class `TAX` are considered tax lines.

If the condition type name names a particular condition type, then the line for this condition type is used even if the condition class is not `TAX`.

Lines selected from the pricing procedure that do not exist in the price calculation of the position are ignored. If no lines have been selected, or all selected lines are ignored, the reported monetary amount will be null.

If <suffix> does not adhere to the syntax specified above, no pricing procedure can be selected, or the selected pricing procedure does not have the specified condition type, then an invalid monetary amount is reported

`TaxPerUnit:<suffix>`

tax per sales unit; This is like `Tax:<suffix>`, except that the determined value is divided by the position's quantity.

Note that, depending on the calculation rule of the taxes involved, this value may differ from the value computed for `Tax:<suffix>` for an otherwise identical position with a quantity of one. Also note that this value cannot be computed, and an invalid monetary amount will be reported, if the quantity of the position is zero.

`TaxRate:<suffix>`

the tax rate, consisting of condition amount and possibly quantity relation

<suffix> is interpreted as described for `Tax:<suffix>`, except that an asterisk must not be used for the condition type name.

Access to the variable first determines amount and quantity relation. Both default to an invalid value.

If a pricing procedure line has been selected, and the price calculation of the position contains this line, then the line's values are used for amount and quantity relation. Note that depending on the calculation rule, the quantity relation may be invalid (for example in case of `Percent` and `FixedAmount`).

If a pricing procedure line has been selected, but the line does not exist in the price calculation of the position, then the amount will be zero percent if the calculation rule of the condition type is `Percent`, and null otherwise.

Once amount and quantity relation have been determined, they are formatted as just the amount if the quantity relation is invalid, or as amount and quantity relation, in that order, separated by space, slash and space (U+0020, U+002F, U+0020).

Note that, while P-BK and derived applications also support `Tax:<suffix>`, `TaxPerUnit:<suffix>` and `TaxRate:<suffix>`, they interpret `<suffix>` in a different way. Also note that the `TaxCategory` variable supported by P-BK is not supported at all by EAIWS.

5.6.1.98 `GetItemFieldOptions`

Synopsis:

```
struct  GetItemFieldOptions : ItemSelectionOptions {
    boolean *useDefault;
    boolean *expandVariables;
    boolean *resolveURI;
    boolean *useImageCache;
    boolean *generateImages;
}
```

The `options` parameter may specify the following options to control field access and representation of the return value:

`useDefault`

if `true`, and a field has no value, than the default value of the field (if any) will be used instead

`expandVariables`

if `true`, variables found in the field (or the field's default value) are expanded

`resolveURI`

if true, and the field is of a predefined column with attribute ID `CatalogImage`, `ArticleImage` or `GeneratedImage`, or a user defined column with type `Image`, an attempt is made to interpret the field's value as an absolute URI and, if the URIs scheme is `file`, `gen` or `imp`, to produce an HTTP URL for this URI. If a HTTP URL could be produced, then the HTTP URL will be returned as the field's value. Otherwise, the original value of the field will be returned.

`useImageCache`

If specified and `true`, try to fetch the generated image from the global cache if the operation `getItemFields` has been called to return a generated image or article image and the current position does not already reference a generated image.

Default value: `false`

`generateImages`

If specified and `true`, try to fetch the generated image from the global cache if the operation `getItemFields` has been called to return a generated image or article image and the current position does not already reference a generated image. Generate the image if it has not been found in the cache.

Default value: `false`

Note: It is not possible to bypass the cache with useImageCache set to false and generateImages set to true. The latter one set to true always causes a cache lookup before the image is eventually generated.

5.6.1.99 `PasteOptions`

Synopsis:

```
struct  PasteOptions {
    boolean reuseItemIds;
    boolean returnOriginalItemIds;
    boolean ignoreComposableGeometries;
}
```

The complex type `PasteOptions` defines two optional attributes:

`reuseItemIds`

Default: `true`

If `false`, newly inserted items will have new unique item IDs. If `true`, newly inserted items will have the item ID found in the OBX stream unless there is already another item using this item ID (in which case a new unique item ID will be used).

`returnOriginalItemIds`

Default: `false`

If `false`, the sequence returned by the `paste` operation contains the item IDs of all newly inserted items. If `true`, the sequence contains twice as many entries. Entries at even indices (zero-based) contain IDs of newly inserted items, whereas entries at odd indices contain the corresponding item ID of the item as found in the OBX stream.

`IgnoreComposableGeometries`

Default: `false`

If set to true, operations `paste (§5.6.3.52)` and `pasteContainer` ignore `<composableGeometry/>` elements.

The default values are used if no options argument has been passed to `paste`, or if the attribute has not been specified.

5.6.1.100 `PasteContainerOptions`

Synopsis:

```
struct PasteContainerOptions : PasteOptions {
    boolean *setArticleIds;
    boolean *geometryIds;
}
```

Can be used as argument of operation `pasteContainer (§5.6.3.53)`.

The optional attributes `setArticleIds` and `geometryIds` of type `boolean` control whether or not returned instances of complex type `BasketItem` contain set-article IDs (and set-article part IDs) and geometry IDs (if available).
The default value of these options is `false`.

5.6.1.101 `ItemAppData`

Synopsis:

```
struct  ItemAppData {
    BItemId   id;
    string*[] data;
}
```

5.6.1.102 `OAPAction`

Synopsis:

```
struct OAPAction {
    string                 id;
    OAPObjectDefinition[] objects;
}
```

5.6.1.103 `OAPActionChoiceAction`

Synopsis:

```
struct OAPActionChoiceAction : OAPAction {
    OAPViewType          viewType;
    OAPTileSize          *tileSize;
    OAPActionListItem[] actions;
    string              *title;
}
```

The complex type `OAPActionChoiceAction` used to return information about Action-Choice actions.

`viewType`

```
enum OAPViewType {
    List,
    Tile
}
```

`tileSize`

The attribute `tileSize` will be present if and only if attribute `viewType` has value `Tile`.

```
enum OAPTileSize {
    Small,
    Medium,
    Large
}
```

`actions`

```
struct OAPActionListItem {
    OAPAction action;
    string    *text;
    string    *imageFile;
    string[]  actionIds;
}
```

Attribute `imageFile`, if present, contains an URL for the image file.

The field `actionIds` contains the list of action IDs if multiple action IDs per action list item are enabled (see description of OAP client capability MultiActionChoice § 5.6.1.106).

`title`

This field contains the action choice dialog title. It is present if and only if the dialog has a nonempty title.

5.6.1.104 `OAPArticleSpecMode`

Synopsis:

```
enum OAPArticleSpecMode {
      Self,
      Explicit
}
```

5.6.1.105 `OAPPropChangeAction`

Synopsis:

```
struct OAPPropChangeAction : OAPAction {
    OAPPropChangeType type;
    string            property;
    string            value;
}
```

The information returned by `OAPPropChangeAction` is not supposed to be used by the client to set the property value or change the property state. Instead, clients should use `oapProcessActions` to execute property change actions[48].

```
enum OAPPropChangeType {
    Value,
    Visibility,
    Editability
}
```

5.6.1.106 `OAPClientCapability`

Synopsis:

```
enum OAPClientCapability {
    MultiActionChoice,
    PlanningMode
}
```

A set of values of this enumeration is used to specify OAP-related capabilities of the client. There is one value for each capability. The client calls operation `oapSetClientCapabilities (§ 5.6.3.101)` to inform the server about the capabilities it supports.

No new capability will be defined for new server-side OAP features if they only results in new data being returned to the client that can safely be ignored[49].

The current version of EAIWS defines the following capabilities:

`MultiActionChoice`

Staring with OAP draft 16, OAP action list items (henceforth called action choices) contain a list of action IDs instead of a single action ID.

If the `MultiActionChoice` capability has not been set then no `OAPActionListItem` is returned for an action choice that contains multiple action IDs, or for an action choice with a single action ID if the referenced action is inactive. The action-field of `OAPActionListItem` always contains the complete action data.

If capability `MultiActionChoice` has been set then an `OAPActionListItem` is returned for each visible action choice[50]. Field `action` of `OAPActionListItem` is missing. Instead, field `actionIds` is used to return one or more action IDs (which may reference inactive actions[51]). Once an action choice has been selected the corresponding list of action IDs can be processed with operation `oapProcessActions`.

`PlanningMode`

Starting with OAP draft 16, interactor definitions contain a flag that indicates whether or not the actions attached to the interactor require the user interface to support planning mode. Supporting plan-

[48] The client has no information about the property class, and finding out the property class would cause undue overhead on the server side. Furthermore, the returned value is an OAP expression that must be evaluated to determine the new property value, which cannot be done by the client, at least not in the general case. And finally, the client does not have access to an API to modify the property state.

[49] Clients are expected to ignore data returned with unknown XML attributes and elements.

[50] as determined by the new (optional) condition of the action choice

[51] Action choices with no condition, or with a satisfied condition, are expected to reference at least one active action

ning mode means that the user interface is able to simultaneously display, and allow interaction with, multiple top-level objects.

If OAP client capability `PlanningMode` has not been set then the server does not return interactors that require the user interface to support planning mode according to the aforementioned flag. If capability `PlanningMode` has been set then all active interactors are returned.

5.6.1.107 `OAPMediaSource`

Synopsis:

```
enum OAPMediaSource {
        PIM,
        YouTube
 }
```

5.6.1.108 `OAPAttachAreasPlacement`

```
struct OAPAttachAreasPlacement : OAPPlacement {
    OAPObjectDefinition referenceObject;          // element
    string[]            refObjectAttachAreaIds;   // element
    string              newObjectAttachAreaId;    // element }
```

`referenceObject`

definition of reference object as found in OAP data

`refObjectAttachAreaIds`

list of IDs of attach areas of the reference object as found in OAP data

`newObjectAttachAreaId`

attach area ID of new object as found in OAP data

5.6.1.109 `ViewDisplayMode`

Synopsis:

```
enum ViewDisplayMode {
    Undefined,
    Sorted,
    Planning
}
```

The view display mode is the primary mode defining the structure of basket views. The following enumeration values are defined:

`Undefined`

Reserved for internal use.

`Sorted`

This display mode mostly ignores the structure of the basket item tree. Instead it assumes a basket item tree that looks as follows:

- The top folder continues to be the top folder.
- Basket folder items that are not part of a set-article are children of their nearest basket folder item ancestor[52] that is not part of a set-article either.
- Main article items that are not part of a set-article and set-article items are children of their nearest basket folder ancestor that is not part of a set-article either.

[52] ancestor: parent and ancestors of parent

- Basket folder items and main article items that are part of a set-article are children of either the set-article item or a basket folder item that is part of the same set-article, whichever is their nearest ancestor in the original basket item tree. If there is no such ancestor, they are children of the set-article item.
- Sub-article items are children of their main article item.

The actual tree of view items presented to the client may be further modified depending on other view configuration options.

Siblings are sorted according to a configurable sort order. The default sort order is as follows:

1. All folders, sorted by name.
2. All partial plannings and set-articles, sorted by name.
3. All (other) articles, sorted by manufacturer, series, article specification (usually base article number, but may be final article number depending on product data), and description.

Right now there is no web service operation to query or configure the sort order.

`Planning`

The tree of view items reflects more or less the tree of basket items. The only exception are set-article parts that may be hidden depending on view configuration option setArticleMode and the collapsed state of the set-article item.

5.6.1.110 `MergeMode`

Synopsis:

```
enum MergeMode {
    None,
    Articles,
    Composites,
    CompositesStrict,
    SubArticles,
    Compact,
    ArticlesCompact
}
```

The merge mode is meaningful for views that use display mode `Sorted`. It controls whether or not, and how, identical child items of the same parent are merged into a single item:

`None`

Items are never merged.

`Articles`

All equal basket article items are merged[53]. Basket article items are considered equal if they have the same configuration (manufacturer and series ID, base article number, variant code, OFML variant code, fields, additional texts, set-article IDs[54]) and calculation data.

There is usually no need for two article items to have the same quantity to be merged unless the items have a calculation with an item condition (i.e. not a group or header condition) with calculation rule `FixedAmount`.

The current implementation allows merging of main article items of composites with different sets of sub-article items.

`SubArticles`

Only basket sub-article items are merged. Otherwise the criteria for merging are similar to merge mode `Articles`.

Sub-article items of different composites may be merged if they happen to have the same view item

53 The current implementation does not merge main article items with sub-article items.

54 Both items are either part of the same set-article or are not part of a set-article.

as their parent, as may happen if options `expandPartialPlannings` and/or `expandAggregates` are `true`.

`Compact`

Merge mode `Compact` behaves more or less like merge mode `SubArticles`. The difference to merge mode `SubArticles` is as follows:

- Expansion of partial plannings and aggregates is always disabled.
- Article view items that represent sub-article items (merged or not) do not get a position number assigned. Their IDs are never returned by the basket web service. Instead, their article data is returned as part of the item properties of the main article item they belong to.
- Operation `getAllItems` returns, in addition to the basket item ID of the referenced main article item, the basket item IDs of all sub-article items of this main article item.
- Operations `getPriceCalculationSheet` and `getPriceCalculationSheets` automatically return group calculations if they are asked to return the calculation for an article view item that represents a partial planning or aggregate with at least one sub-article item.
- The default value of option `hideSubArticles` of operation `getGeneratedImage` defaults to `false` if the operation is called for a main article item.

5.6.1.111 `SetArticleMode`

Synopsis:

```
enum SetArticleMode {
    Expand,
    Collapse,
    Dynamic
}
```

The view configuration option `setArticleMode` controls the view's handling of set-article parts. It is honored in all view display modes.

`Expand`

Set-article parts are always visible.

`Collapse`

Set-article parts are never visible.

`Dynamic`

Set-article parts are hidden if and only if the set-article item is collapsed.[55]

5.6.1.112 `PriceInfo`

Synopsis:

```
struct PriceInfo {
    boolean   inactive;
    decimal   quantity;
    Quantity  unitSize;
    Money    *netPrice;
    Money    *netValue;
    Money    *tax;
    PriceInfoElement[] extraPriceInfos
}
```

`inactive`

The value of this field is usually `false`. A value of `true` indicates that this price information should

55 The 'collapsed' state of set-articles can be queried with operation `getItemProperties` (`property setArticle.collapsed`). It can be modified with operations `expandSetArticles` and `collapseSetArticles`.

be ignored when the total price of the document is computed (be it because the position is an alternative position, part of a set-article, or part of sub-article data in case of merge mode `Compact`).

`quantity`

The quantity (in terms of unit size) of this item. Note that in case of sub-article items and merge mode Compact, and for set-article parts, this is the quantity of this item relative to a single sales unit of the main article or set-article.

`unitSize`

The sales unit size of this item. For set-article items and user article items the unit size is always one piece (1 C62). The usual unit size for other types of article items is one piece too, but other unit sizes may occur.

`netPrice`

The net price is the net value divided by the quantity as reported by the price information and thus the net price of a single sales unit.

`netValue`

The net value corresponds to the value of the calculation line tagged with `NET_VALUE`. If the price information is queried for a view item referencing multiple basket items, as may happen in case of a merge mode other than `None`, the net value is the sum of the net values of all item calculations, including item calculations of sub-article items in case of merge mode `Compact`.

In case of article data returned for sub-articles in merge mode `Compact`, the net value is further adjusted to correspond to the net value of this item relative to a single sales unit of the main article.

`tax`

This element contains the total of all calculation lines tagged with `TAX`. If the price information is queried for a view item referencing multiple basket items, as may happen in case of a merge mode other than `None`, the returned tax is the sum of the taxes of all item calculations. In case of sub-article items and merge mode `Compact` the returned tax is further adjusted to correspond to the total tax of this item relative to a single sales unit of the main article.

Fields `netPrice`, `netValue` and `tax` may be missing if no lines tagged with `NET_PRICE` and/or `TAX` have been found in the calculation or values returned by the calculation are invalid.

5.6.1.113 `GetPriceCalculationSheetOptions`

Synopsis:

```
struct GetPriceCalculationSheetOptions : CondGroupSelectionOptions {
            boolean sumUpComposite,
            boolean itemCondAmountPerUnit,
            boolean headerCondAmountPerUnit
}
```

`sumUpComposite`

If all selected items are part of one or more composite articles, all main and sub-article items of these composite articles are selected, all composite articles are considered equal[56], and the unit of the sales unit size of the main article items is C62 (pieces), then a group calculation produced for all selected articles will report a quantity equal to the total of the quantities of all main article items and a sales unit size matching the sales unit size of the main article items.

Use of this option is more or less a precondition for use of options `itemCondAmountPerUnit` and `headerCondAmountPerUnit`.

The default value of this option is `false`.

56 same configuration and calculation of main articles and corresponding sub-articles, but not necessarily same structure of sub-articles

`itemCondAmountPerUnit`

`headerCondAmountPerUnit`

The usual algorithm to combine item calculation conditions into group conditions is to sum up the item calculation condition values to get the group calculation condition value and display amount and quantity relation of the group calculation condition if and only if all combined item calculation conditions have the same value.

If one or both of these options are used, this algorithm is replaced for the corresponding condition type by another one that behaves as follows:

- Calculation rule, amount and quantity relation are invalid if the group calculation does not have a finite non-zero quantity.
- Otherwise, the amount is computed as the condition value divided by the quantity. If the amount is finite and not null then the calculation rule is set to `Quantity`, the amount to the properly rounded computed amount, and the quantity relation to the sales unit size of the group calculation. Otherwise invalid values are used.

Furthermore, if at least one of these options is `true`, all operations that work on calculations enforce the use of a group calculation, even if only a single (set-)article item is selected.

The default value of these options is `false`.

`ExcludeInactiveItems and includeInactiveItems`

Both options are mutually exclusive. The operation fails with a BasketServiceFault if both are set to true. Other than that the values are ignored if no group calculation sheet is returned.

If both options are unspecified, the default values of excludeInactiveItems and includeInactiveItems are true and false, respectively. Otherwise the default value of both options is false.

If excludeInactiveItems and includeInactiveItems are both false, group calculations ignore item calculations of inactive items (i.e. alternative positions) if and only if the group contains at least one active item.

If excludeInactiveItems is true, group calculations always ignore calculations of inactive items. This is the default behavior if neither option is specified, and results in more or less the traditional behavior. The only difference is that calculations of inactive items no longer affect the set of condition lines added to the group calculation, so the group calculation returned for a group consisting entirely of inactive items looks more or less the same as a group calculation returned for an empty group, an empty folder, or a document calculation returned for an empty document (it contains no condition lines except header conditions that are present in the document calculation).

If includeInactiveItems is true, group calculations do not ignore calculations of inactive items, i.e. group condition values are computed as the sum of item conditions values of both active and inactive items.

5.6.1.114 `ArticleDescriptionMode`

Synopsis:

```
enum ArticleDisplayMode {
    Short,
    Long,
    Both
}
```

The article description mode is used to indicate which article text (short, long or both) should be displayed in the order listing.

5.6.1.115 Axis

Synopsis:

```
enum Axis {
      X,
      Y,
      Z
}
```

5.6.1.116 ARSRenderingSetup

Synopsis:

```
struct ARSRenderingSetup (
      ARSCamera camera,
      ARSLighting lighting
)
```

5.6.1.117 BasketViewConfig

Synopsis:

```
struct BasketViewConfig (
                         string name,
                         string[] visibleColumns,
                         ColumnWidth[] columnWidths,
                         BasketViewSortConfig sortConfig,
                         DescrType[] articleDescrMode
                         string viewId,
                         boolean removable,
                         boolean editable,
                         ViewDisplayMode displayMode,
                         boolean expandGroups,
                         boolean expandPlanningFolders,
                         boolean expandBasketFolders,
                         boolean expandPartialPlannings,
                         boolean expandAggregates,
                         boolean? textItemsVisible,
                         boolean nonOfferArticlesVisible,
                         boolean? nonOrderArticlesVisible,
                         MergeMode mergeMode,
                         boolean autoColumnWidth,
                         boolean hiddenDiscounts,
                         SetArticleMode setArticleMode
                        )
```

```
struct ColumnWidth {
    ColumnId columnId;
    int      width;
}
```

The parameter `viewIds` must be a possibly empty list of valid view IDs. If empty, the set of all current basket views is used instead. Return value is a list of instances of type `BasketViewConfig`, one for each specified view ID.

All fields of type `BasketViewConfig` except `viewId` are optional because the type is also used by operation `changeBasketViewConfig` to change a subset of basket view configuration options.

The following fields are defined:

`viewId`

The view ID. This field is most useful if parameter `viewIds` is empty, but in general clients should not assume that view configurations are returned in the same order as specified by parameter `viewIds`.

`removable`

A Boolean value that indicates whether or not the view can be removed. The current implementation allows all views except the standard view to be removed.

`editable`

A Boolean value that indicates whether or not the configuration of this view can be changed. The current implementation allows the configuration of all views except the standard view to be changed.

`displayMode`

The display mode used by this view.

`expandGroups`

A Boolean value that controls expansion of planning groups. This option exists for compatibility with P-BK and has, at least for now, no meaning for EAIWS.

`expandPlanningFolders`

A Boolean value that controls expansion of planning folders. This option exists for compatibility with P-BK and has, at least for now, no meaning for EAIWS.

`expandBasketFolders`

A Boolean value that controls expansion of basket folders in display mode `Sorted`. If a view expands basket folders then all basket folder are replaced by their content (the basket folders themselves become invisible).

Basket folders are expanded if and only if the display mode is `Sorted` and the value of this option is `true`.

`expandPartialPlannings`

A Boolean value that controls expansion or partial plannings (basket main article items with item type `PartialPlanning`). Expanded partial plannings are replaced by their content. The partial plannings themselves become invisible.

Basket main article items with item type `PartialPlanning` are expanded if and only if the display mode is `Sorted`, the merge mode is *not* `Compact`, and the value of this option is `true`.

Basket sub-article items with item type `PartialPlanning` are expanded if and only if the display mode is `Sorted` and the merge mode is *not* `Compact`.

`expandAggregates`

A Boolean value that controls expansion of aggregates (basket main article items with item type `Aggregate`). Children of expanded aggregates become children of the aggregates parent[57]. The aggregate itself remains visible.

Basket main article items with item type `Aggregate` are expanded if and only if the display mode is `Sorted`, the merge mode is *not* `Compact`, and the value of this option is `true`.

Basket sub-article items with item type `Aggregate` are expanded if and only if the display mode is `Sorted` and the merge mode is *not* `Compact`.

`nonOrderArticlesVisible`

The field is used to query and set the corresponding flag of basket views. Its default value when

57 i.e. they become siblings of the aggregate

used as template argument of operation `addBasketView` is true.

`mergeMode`

The merge mode to be used in case of display mode `Sorted`. The merge mode reported for views with display mode `Planning` is always `None`, even if another merge mode has been explicitly configured with operation `changeBasketViewConfig`.

`textItemsVisible`

This attribute is used by views with display mode Sorted to control whether or not text items are visible. It can be set for views with display mode Planning, even though its value is ignored by these views.

Switching display modes does not affect the value of this flag.

The default value of this flag is false.

`autoColumnWidth`

A Boolean option that exists for compatibility with P-BK and controls whether or not P-BK automatically adjust the width of columns to match their content or uses fixed column widths.

`hiddenDiscounts`

A Boolean option that exists for compatibility with P-BK and controls whether or not discounts (and extra charges) should be hidden in this view[58].

`setArticleMode`

The set-article mode controls whether or not parts of set-articles should be visible within the view. See description of type `SetArticleMode` for details.

`name`

The name of the view.

`visibleColumns`

A list of IDs of columns that should be visible in this view.

`columnWidths`

A possibly empty list of explicitly configured column widths. See option `autoColumnWidth` for more information.

Note that the values returned for the various expand modes may be `true` even if the corresponding item types are not expanded due to the selected display and merge modes.

5.6.1.118 `DescrType`

Synopsis:

```
enum DescrType {
      Default,
      Short,
      Long,
      Features,
      Extra,
      PreferUserDescr
}
```

58 If so, single and total sales prices, if displayed, will have to be faked.

Default - the default article text according to OFML (either the short text, the long text, or both). The default text is inserted in the returned description in place of either the short text row or the long text row, whichever comes first. If 'Default' is specified, 'Short' and 'Long' are ignored.

Short - the article short text

Long - the article long text

Features - the feature/variant text

Extra - application-specific text

5.6.1.119 `Dimension`

```
enum Dimension {
       X,
       Y,
       Z,
       PX,
       PY,
       PZ,
       NX,
       NY,
       NZ
 }
```

5.6.1.120 `GetArticleDataOptions`

Synopsis:

```
struct GetArticleDataOptions (
      boolean noProperties,
      boolean fetchCatalogImage,
      boolean fetchCatalogIcon,
      string viewId
      boolean separateCurrencies,
      boolean? highResPropValueIcons,
      boolean? FetchPropValueImages,
      boolean? enableBooleanPropType
)
```

`highResPropValueIcons`

Controls whether fields smallIcon and largeIcon of type PropertyValue return URLs for the traditional low resolution material icons (if false) or may return URLs for high resolution material icons (if true).

If set to false (the default value), the behavior of EAIWS with regards do fields smallIcon and largeIcon does not change.

If set to true, and directories .../mat/ and .../mat/m exist, smallIcon, if non-empty, references a file in .../mat/s, and largeIcon, if non-empty, references a file in .../mat/m.

If set to true and at least one of .../mat/s and .../mat/m does not exist, both smallIcon and largeIcon, if non-empty, reference a file in .../mat.

`fetchPropValueImages`

Controls whether the new field image of type PropertyValue is absent (if false, the default value) or may reference a material image in directory .../mat/l (if true).

Field image is absent if option `fetchPropValueImages` is set to true but no corresponding image file is found. This differs from fields smallIcon and largeIcon, which are always present, but contain an empty string if the corresponding icon is not found.

The optional boolean field `enableBooleanPropType` enables boolean properties. Its default value is `false`, resulting in no change of behavior. Previously, boolean properties have been treated as properties of type `Numeric` with values one and zero for `true` and `false` when returned by operations `getArticleData`, `getChoiceList` and `getAllChoiceLists`, or zero and non-zero for `false` and `true` when set with operation `setPropertyValue`.

If `enableBooleanPropType` is set to `true`, the behavior for boolean properties changes as follows:

- Operation `getArticleData` returns `Boolean` instead of `Number` in field `type` of type `Property`.
- Operation `getArticleData` returns `true` and `false` instead of 1. and 0. in field `value` of type `PropertyValue`.

If option `enableBooleanPropType` is effectively `false`, operation getArticleData returns the value of boolean properties as `1` and `0` instead of `1.` and `0.` in field `value` of type `PropertyValue`.

5.6.1.121 `InactiveFlag`

Synopsis:

```
enum InactiveFlag {
      NoData,
      SubsequentPrice
}
```

5.6.1.122 `InactivePositionState`

```
struct InactivePositionState (
      string ppName,
      boolean? optional
)
```

Field `ppName` specifies the name of the pricing procedure that treats the position as an inactive position, and field `optional` indicates whether the user interface should treat the position as an optional (true) or alternative (false) position. The default value of optional is false.

An empty string as `ppName` acts as a kind of wildcard, matching all pricing procedures, including those later added to the basket.

5.6.1.123 `MergeResult`

Synopsis:

```
struct MergeResult(
      string[] addedItems,
      string[] removedItems,
      string[] movedItems,
      string[] updatedItems
)
```

5.6.1.124 `OAPActionContext`

Synopsis:

```
struct OAPActionContext (
      string self,
      string interactor,
      float dpr
)
```

5.6.1.125 OAPActionListItem

Synopsis:

```
struct OAPActionListItem (
      OAPAction action,
      string[] actionIds,
      string text
      string imageFile
)
```

5.6.1.126 `OAPActionResult`

Synopsis:

```
struct OAPActionResult (
      OAPAction actionData,
      string[] objects,
      string referenceObject,
      Vector3 newObjectPosition,
      Rotation newObjectRotation,
      string[] addedItems,
      string[] removedItems,
      string[] movedItems,
      string[] updatedItems
      string id,
      OAPActionState state
)
```

5.6.1.127 `OAPActionState`

Synopsis:

```
enum OAPActionState {
      Disabled,
      Enabled,
      Success,
      Failure,
      NotResponsible
}
```

5.6.1.128 `OAPArticleData`

Synopsis:

```
struct OAPArticleData (
      OAPGeneralInfo generalInfo,
      string[] propertyChangeActions,
      OAPAttachArea[] activeAttachAreas,
      OAPAttachArea[] passiveAttachAreas,
      string[] interactorIds
)
```

5.6.1.129 `OAPGeneralInfo`

Synopsis:

```
struct OAPGeneralInfo {
    boolean         needsOFML;
    Vector3         boundsMin;
    Vector3         boundsMax;
    string?         preview;
}
```

5.6.1.130 `OAPAttachArea`

Synopsis:

```
struct OAPAttachArea {
    string          id;
    OAPGeometry     geometry;
    Vector3?        cursorPosition;
    string[]        linkedAreas;
}
```

5.6.1.131 `OAPGeometry`

Synopsis:

```
struct OAPGeometry {}
```

5.6.1.132 `OAPPointGeometry`

Synopsis:

```
struct OAPPointGeometry : OAPGeometry {
    Vector3         position;
}
```

5.6.1.133 `OAPPointListGeometry`

Synopsis:

```
struct OAPPointListGeometry : OAPGeometry {
    Vector3[]       points;
}
```

5.6.1.134 `OAPPolyLineGeometry`

Synopsis:

```
struct OAPPolyLineGeometry : OAPGeometry {
    Vector3[]       definitionPoints;
    OAPRasterType      rasterType;
    double?         rasterValue;
    Vector3[]       rasterPoints;
}
```

5.6.1.135 OAPRectangleGeometry

Synopsis:

```
struct OAPRectangleGeometry : OAPGeometry {
    Vector3[]       vertices;
    RasterType      rasterType;
    Vector2?        rasterValue;
    Vector3[]       rasterPoints;
}
```

5.6.1.136 RasterType

Synopsis:

```
enum RasterType { None, Fixed, List }
```

5.6.1.137 OAPDataDefinedPlacement

Synopsis:

```
struct OAPDataDefinedPlacement {
    OAPObjectDefinition?  referenceObject;
    string?  attachPoint;
}
```

5.6.1.138 OAPDeleteObjectAction

Synopsis:

```
struct OAPDeleteObjectAction : OAPAction {
}
```

5.6.1.139 OAPMethodCallAction

Synopsis:

```
struct OAPMethodCallAction : OAPAction {
    MethodCallType type;
    string         context;
    string         method;
    string         arguments;
}
```

5.6.1.140 OAPMethodCallType

Synopsis:

```
enum OAPMethodCallType {
      Instance,
      Class
}
```

5.6.1.141 OAPObjectCategory

Synopsis:

```
enum OAPObjectCategory {
      Self,
      ParentArticle,
      TopArticle,
      SubArticles,
      SubArticlesDeep,
      SubArticle,
      Article,
      Attached,
      AttachedRecursive,
      AttachedArea,
      AttachedAreaRecursive,
      AttachedArticle,
      MethodCall
}
```

5.6.1.142 OAPObjectDefinition

Synopsis:

```
struct OAPObjectDefinition {
    OAPObjectCategory     category;
    string[]              arguments;
}
```

5.6.1.143 OAPPlacement

Synopsis:

```
struct OAPPlacement {}
```

5.6.1.144 OAPCreateObjectAction

Synopsis:

```
struct OAPCreateObjectAction : OAPAction {
    OAPObjectDefinition?  parent;
    OAPArticleSpecMode    articleSpecMode;
    string?               packageId;
    string?               baseArticleNumber;
    string?               ofmlVariantCode;
    OAPPlacement          initialPlacement;
}
```

5.6.1.145 OAPDataDefinedPlacement

Synopsis:

```
struct OAPDataDefinedPlacement : OAPPlacement (
      OAPObjectDefinition referenceObject
      string attachPoint
)
```

5.6.1.146 OAPDeleteObjectAction

Synopsis:

```
struct OAPDeleteObjectAction : OAPAction ( )
```

5.6.1.147 `OAPDimChange`

Synopsis:

```
struct OAPDimChange (
      decimal[]? PropChoiceList,
      string[] symbolicPropValues,
      Axis? axis,
      Dimension dimension,
      string propClass,
      string propName,
      decimal propValue,
      decimal minPropValue,
      decimal maxPropValue,
      boolean fixedValue,
      int propDecDigits,
      decimal multiplier,
      decimal precision,
      boolean changeSeparately,
      boolean thirdDimension
)
```

`axis`

Field `axis` is optional. It is present if and only if the new field `dimension` has a bi-directional value (`X`, `Y` or `Z`).

`dimension`

specifies the axis and direction of an allowed dimension change.

`propClass`

Optional attribute, type string class of the property to use; This attribute is optional because a) determining the class of the property would cause some additional overhead and b) in case of OFML articles the property class is ignored by operation 'setPropertyValue' anyway.

So, if no property class is returned, the client should pass an empty string as property class to 'setPropertyValue'. If a property class is returned the client is expected to pass this roperty class to 'setPropertyValue'.

`PropName`

Required attribute, type string name of the property to use

`propValue`

Required attribute, type decimal current property value

`minPropValue`

Required attribute, type decimal minimum allowed property value

`maxPropValue`

Required attribute, type decimal maximum allowed property value

`propChoiceList`

Optional sequence of elements of type decimal optional choice list `fixedValue`

Required attribute, type boolean, if true, only choice list values are allowed as property values; if false, choice list values (if present) are just suggestions

`propDecDigits`

Required attribute, type int, allowed number of decimal digits in property values right of decimal point

`multiplier`

Required attribute, type decimal, conversion factor used to convert between meters and the unit used by property values (given in meters per prop value unit)

`precision`

Required attribute, type decima, precision of dimension change given as an absolute value in meters (for instance, 0.1 means that dimension should be changed in multiples of 10 centimeters)

`changeSeparetely`

corresponds to column \`Separate\` of OAP table \`DimChange\`. For OAP 1.1 data, the value of this field is false.

`symbolicPropValues`

The sequence of symbolic property values is either empty or contains the same number of elements as the sequence of numeric choice values (propChoiceList). If non-empty, the client must set the property specified by fields propClass and propName to the symbolic value at index N to effect a dim-change to the choice value at index N.

`thirdDimension`

corresponds to column \`ThirdDim\` of OAP table \`DimChange\`. For OAP 1.1 data, the value of this field is false.

5.6.1.148 `OAPDimChangeAction`

Synopsis:

```
struct OAPDimChangeAction : OAPAction (
      OAPDimChange[] dimChanges
)
```

5.6.1.149 `OAPDimChange2Action`

Synopsis:

```
     OAPDimChange2Action extends OAPAction (
            OAPDimChange[] dimChanges
)
```

5.6.1.150 `OAPPropEdit2Action`

Synopsis:

```
OAPPropEdit2Action extends OAPAction (
       string title,
       OAPPropEditProp[] properties,
       OAPPropEditClass[] propertyClasses
```

```
    )
```

5.6.1.151 `OAPPropEditAction`

Synopsis:

```
struct OAPPropEditAction : OAPAction {
    string                title;
    string[]              properties;
    string[]              propertyClasses;
    boolean               visibleOnly;
    boolean               editableOnly;
}
```

5.6.1.152 `OAPPropEditClass`

Synopsis:

```
OAPPropEditClass (
       string propClass,
       boolean visibleOnly,
       boolean editableOnly
 )
```

Field `propClass` of `OAPPropEditProp` is optional. If missing (currently always the case) or empty, clients should either ignore the property class (e.g. if comparing property class and name with instances of `Property` as returned by operation `getArticleData`) or specify an empty property class in situations where a property class must be specified (as required by operation `setPropertyValue`).

Fields `visibleOnly` and `editableOnly` are used to restrict the visibility of individual properties in OAP property editors. If `visible` and `editable` are the property-specific flags reported as fields of complex type `Property`, then the corresponding property must be visible if and only if the expression

(visible || !visibleOnly) && (editable || !editableOnly) is true. Or, perhaps more intuitive, they must be hidden if

(visibleOnly && !visible) || (editableOnly && !editable) is false.

5.6.1.153 `OAPPropEditProp`

Synopsis:

```
OAPPropEditProp (
       string propClass,
       string propName,
       boolean visibleOnly,
       boolean editableOnly
 )
```

Fields `visibleOnly` and `editableOnly` are used to restrict the visibility of individual properties in OAP property editors. If `visible` and `editable` are the property-specific flags reported as fields of complex type `Property`, then the corresponding property must be visible if and only if the expression

(visible || !visibleOnly) && (editable || !editableOnly) is true. Or, perhaps more intuitive, they must be hidden if

(visibleOnly && !visible) || (editableOnly && !editable) is false.

5.6.1.154 `OAPRasterType`

Synopsis:

```
enum OAPRasterType {
      None,
      Fixed,
      List
}
```

5.6.1.155 `OAPRasterType`

Synopsis:

```
enum OAPRasterType {
      None,
      Fixed,
      List
}
```

5.6.1.156 `OAPSelectObjectAction`

Synopsis:

```
struct OAPSelectObjectAction : OAPAction ( )
```

5.6.1.157 `OAPShowMediaAction`

Synopsis:

```
OAPShowMediaAction extends OAPAction (
       OAPMediaSource mediaSource,
       string mediaId
 )
```

The fields `mediaSource` and `mediaId` correspond to columns `Type` and `Media` of OAP table `ExtMedia`.

5.6.1.158 `OAPSymbolSize`

Synopsis:

```
enum OAPSymbolSize {
      Small,
      Medium,
      Large
}
```

5.6.1.159 `PriceInfoElement`

Synopsis:

```
struct PriceInfoElement (
      Value value
      string selector
)
```

5.6.1.160 `SeriesInfo`

Synopsis:

```
struct SeriesInfo (
                  DisplayText[] seriesName,
                        string seriesId
                        )
```

5.6.1.161 `SetArticleProperties`

Synopsis:

```
struct SetArticleProperties (
                    string[] partIds,
                    string[] allPartIds
                    boolean collapsed
                                  )
```

5.6.1.162 `SetLanguagesMode`

Synopsis:

```
enum SetLanguagesMode {
                          Default,
                          CurrentProject,
                          AllProjects
                          }
```

5.6.1.163 `SetPropertyValueOptions`

```
struct SetPropertyValueOptions (
      boolean computeVisibilityChangeFlags,
      boolean computeValueChangeFlags,
      boolean computeChoiceListChangeFlags,
      boolean? EnableBooleanPropType,
)
```

The fields control whether certain flags in the return value of operation `setPropertyValue` are computed. Not computing them may have a positive impact on performance.

The flags controlled by these fields, and their potential performance impact, are as follows:

`computeVisibilityChangeFlags`

The default value is `true`. If `false`, flags S and H are not computed. Setting this flag to `false` won't improve performance unless all flags are set to `false`, and even then the time saved in operation `setPropertyValue` will probably later be spent in operation `getArticleData` (which would otherwise reuse the information cached during `setPropertyValue`).

`ComputeValueChangeFlags`

The default value is `true`. If `false`, flags V, v, U, and A are not computed. As of now, the effects on performance should be the same as described for `computeVisibilityChangeFlags`.

`computeChoiceListChangeFlags`

The default value is `true`. If `false`, flags C and I are not computed. Setting this flag to `false` will always speed up operation `setPropertyValue`, and should not have a negative effect on operation `setArticleData`. However, client's that use operation `getAllChoiceLists` after each invocation of `setPropertyValue` should not see a difference in overall performance.

`enableBooleanPropType`

The optional boolean field enableBooleanPropType enables boolean properties. Its default value is false, resulting in no change of behavior. Previously, boolean properties have been treated as properties of type Numeric with values one and zero for true and false when returned by operations getArticleData, getChoiceList and getAllChoiceLists, or zero and non-zero for false and true when set with operation setPropertyValue.

If enableBooleanPropType is set to true, the behavior for boolean properties changes as follows:

- Operation setPropertyValue expects argument value to be true or false instead of a numeric value (usually but not necessarily 1 or 0). For consistency with the handling of numbers, leading and trailing whitespace are ignored.

If option enableBooleanPropType is effectively false, operation getChoiceList returns the value of boolean properties as 1 and 0 instead of 1. and 0. in field value of type PropertyValue.

5.6.1.164 TMColumnDef

Synopsis:

```
struct TMColumnDef (
       string displayName
       TMColumnId columnId
)
```

5.6.1.165 TMColumnId

Synopsis:

```
enum TMColumnId {
       Undefined,
       Identifier,
       Name,
       Language,
       Text,
       Visible
}
```

5.6.1.166 TMRow

Synopsis:

```
struct TMRow (
       TMText[] texts
       string textId,
       boolean readOnly,
       boolean visible
)
```

5.6.1.167 TMRowDef

Synopsis:

```
struct TMRowDef(
       string displayName
       TMTextType type,
       string textId,
       boolean readOnly,
       boolean defaultVisible
)
```

5.6.1.168 TMTable

Synopsis:

```
struct TMTable (
      TMColumnDef[] columns,
      TMRowDef[] rows
)
```

5.6.1.169 TMText

Synopsis:

```
struct TMText (
      string text
      string language
)
```

5.6.1.170 TMTextType

Synopsis:

```
enum TMTextType {
      Undefined,
      Short,
      Long,
      Features,
      Extra
}
```

5.6.1.171 BasketViewSortConfig

Synopsis:

```
struct BasketViewSortConfig (
                               SortGroup[] groups
                               CollatorStrength collatorStrength,
                               CollatorDecomposition collatorDecomposition)
```

5.6.1.172 ColumnSortOrder

Synopsis:

```
struct ColumnSortOrder (
                         string columnId,
                         boolean ascending,
                         boolean undefinedFirst)
```

5.6.1.173 SortGroup

Synopsis:

```
struct SortGroup (
                   SortGroupSelector[] selector,
                   ColumnSortOrder[] sortOrder
                   int position)
```

5.6.1.174 SortGroupSelector

Synopsis:

```
struct SortGroupSelector (
                        BasketItemType itemType,
                        string itemId)
```

5.6.1.175 CollatorDecomposition

Synopsis:

```
enum CollatorDecomposition {
                        None,
                        Canonical,
                        Full
                          }
```

5.6.1.176 CollatorStrength

Synopsis:

```
enum CollatorStrength {
                        Primary,
                        Secondary,
                        Tertiary,
                        Identical}
```

5.6.2 Faults

5.6.2.1 BasketServiceFault

Synopsis:

```
struct BasketServiceFault {
                  string* message;
                  PartCompositionFailure[]      partCompositionFailures;
}
```

A `BasketServiceFault` is returned by operations of the basket service in case of errors the Online Configurator was prepared to detect and deal with (i.e. to recover cleanly).

See description of complex type PartCompositionFailure and operations getGeneratedImage and getExportedGeometry for more information.

5.6.3 Operations

5.6.3.1 adjustCalculationLineAmount

Synopsis:

```
void adjustCalculationLineAmount(
            string sessionId,
            string[] itemIds,
            string ppName,
            int level,
            int counter,
            decimal amount,
            string currency,
            CondGroupSelectionOptions options
      )
```

The primary purpose of these operations is to adjust margin values and amounts, but they can also be used to set condition amounts and adjust condition values.

Parameters 'sessionId', 'itemIds', 'ppName' and 'options' are interpreted the same way as by 'setConditionAmount()'.

Parameters 'level' and 'counter' must specify a line of the item, aggregate (group/set-article) or document calculation specified by parameters 'itemIds' and 'ppName'. (Other than 'setConditionAmount()', these operations do not allow use of -1 as counter in cases where the specification of the level alone would be sufficient to uniquely identify a calculation line.)

Both 'value' and 'currency' must be specified. The currency must be either an ISO 4217 currency code, '%', or an empty string. An empty string is usually replaced by the document currency, unless the operation is adjustCalculationLineAmount() and the subtotals/conditions calculation rule is 'Percent' (in which case it is replaced by '%') or 'Invalid' (in which case the operation terminates abnormally).

The current implementation does not allow the use of currencies other than the document currency (or the empty string) unless 'adjustCalculationLineAmount()' is used for a condition line.

5.6.3.2 `adjustCalculationLineValue`

Synopsis:

```
    void adjustCalculationLineAmount(
                string sessionId,
                string[] itemIds,
                string ppName,
                int level,
                int counter,
                decimal value,
                string currency,
                CondGroupSelectionOptions options
        )
```

The primary purpose of these operations is to adjust margin values and amounts, but they can also be used to set condition amounts and adjust condition values.

Parameters 'sessionId', 'itemIds', 'ppName' and 'options' are interpreted the same way as by 'setConditionAmount()'.

Parameters 'level' and 'counter' must specify a line of the item, aggregate (group/set-article) or document calculation specified by parameters 'itemIds' and 'ppName'. (Other than 'setConditionAmount()', these operations do not allow use of -1 as counter in cases where the specification of the level alone would be sufficient to uniquely identify a calculation line.)

Both 'value' and 'currency' must be specified. The currency must be either an ISO 4217 currency code, '%', or an empty string. An empty string is usually replaced by the document currency, unless the operation is adjustCalculationLineAmount() and the subtotals/conditions calculation rule is 'Percent' (in which case it is replaced by '%') or 'Invalid' (in which case the operation terminates abnormally).

The current implementation does not allow the use of currencies other than the document currency (or the empty string) unless 'adjustCalculationLineAmount()' is used for a condition line.

5.6.3.3 `AdjustConditionValue`

Synopsis:

```
void adjustConditionValue(
        string sessionId,
        string[] itemIds,
        string ppName,
        string condType,
        int counter,
        decimal value,
        string currency,
        CondGroupSelectionOptions options
       )
```

The operation behaves more or less like setConditionAmount, but instead of just setting the amount to the given value, it tries to compute an amount that results in a value as close as possible to the given value.

The currency argument must be either an empty string (in which case the document currency is assumed) or equal to the document currency.

5.6.3.4 `setLanguages`

Synopsis:

```
void setLanguages(SessionId sessionId, string[] languages)
        throws BasketServiceFault;
```

This operation sets the language list of the basket service (the product data languages) to the specified, possibly empty list of ISO 639 alpha-2 language codes. It then augments[59] this list with the language of the current locale to build an effective list of product data languages. The effective list of product data languages is updated after each invocation of the session service's `setLocale` operation to reflect possible changes to the locale language.

Natural language texts are returned in the first language from the list of effective product data languages supported by the package containing the product database for the article in question. The set of supported languages is defined by the value of the DSR key `languages`. In case the package does not support any of the listed languages then the Online Configurator returns the text in the first supported language.

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. The languages will be changed for the whole session including all projects loaded in the session.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `languages` parameter contains an invalid language code[60].

5.6.3.5 `getLanguages`

Synopsis:

```
string[] getLanguages(SessionId sessionId) throws BasketServiceFault;
```

[59] The locale language is appended to the list if it is not already part of the list.
[60] A language code is accepted as long as it consists of two lower case ASCII letters. It must not necessarily be defined by ISO 639.

This operation queries the current list of product data languages as set by the `setLanguages` operation. If the `setLanguages` operation has not been invoked yet then the list of product data languages is empty.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.

5.6.3.6 `setConditionDescription`

Synopsis:

```
void setConditionDescription(
      string sessionId,
      string ppName,
      string condType,
      string lang,
      string description
)
```

5.6.3.7 `setCurrency`

Synopsis:

```
void setCurrency(SessionId sessionId, string currency)
        throws BasketServiceFault;
```

The `setCurrency` operation changes the current session's basket currency to the currency specified by the `currency` parameter, which must be a known[61] ISO 4217 currency code (consisting of three upper case letters, pseudo-currency codes are not supported).

Prices returned by basket operations use the current basket currency if it is supported by the product database. If the current basket currency is not supported by the product database the behavior is undefined[62].

Note that, although the default basket currency depends on the session's default locale, invocations of the session service's `setLocale` operation never change the current basket currency.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `currencies` parameter contains an invalid currency code, or the specified currency is not supported by the Online Configurator.
- The `currencies` parameter contains a currency code with no available or valid conversion rate in the currency conversion table of the current basket. If a new basket is created (as with the 'openSession' operation), this table is initialized with the currencies found in etc/currencies.cfg. If a basket is loaded using the 'loadSession' operation, the table is loaded from the OBK file.
- If it is passed the code of a pseudo currency.

5.6.3.8 `getCurrency`

Synopsis:

```
string getCurrency(SessionId sessionId) throws BasketServiceFault;
```

This operation queries the current basket currency as set by the `setCurrency` operation, or the session's

[61] The current implementation accepts all currencies known by the Java virtual machine. Future versions of the Online Configurator might further limit the set of accepted currencies.

[62] The current implementation returns prices in currency more or less randomly chosen from the set of supported currencies. A future versions of the Online Configurator will use a well defined algorithm to choose one of the supported currencies and convert the prices fetched from the product database into the current basket currency.

default currency if the setCurrency operation has not been invoked yet.

A BasketServiceFault is returned if any of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.

5.6.3.9 getTopFolderId

Synopsis:

```
BItemId getTopFolderId( SessionId sessionId,
                        ViewId    viewId)
        throws BasketServiceFault;
```

The getTopFolderId operation returns the basket item ID of the root node of the tree of basket items maintained by the basket service for the session identified by the sessionId parameter.

If the argument viewId is present and neither an empty string nor the NIL-UUID then it must be the ID of a current basket view, the itemId argument must be the ID of an item of this view, and the returned item IDs will be IDs of items of this view.
A BasketServiceFault is returned if any of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.

5.6.3.10 getFatherId

Synopsis:

```
BItemId getFatherId(SessionId sessionId,
                    BItemId   itemId,
                    ViewId    viewId)
        throws BasketServiceFault;
```

The getFatherId operation returns the basket item ID of the parent item of the basket item identified by the itemId parameter.

The itemId parameter must identify a basket item that is part of the basket item tree maintained by the basket service for the session identified by the sessionId parameter.

If the itemId parameter references the root node of the basket item tree then the operation returns the null-UUID (§5.2.1).

If the argument viewId is present and neither an empty string nor the NIL-UUID then it must be the ID of a current basket view, the itemId argument must be the ID of an item of this view, and the returned item IDs will be IDs of items of this view.
A BasketServiceFault is returned if any of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.
- The itemId parameter does not represent an UUID.
- The itemId parameter does not identify a basket item.

5.6.3.11 getManufacturerInfo

Synopsis:

```
ManufacturerInfo[] getManufacturerInfo(
                                     string sessionId,
                                     string[] itemIds,
                                     string[] manuIds,
                                     string[] seriesIds,
                                     GetManufacturerInfoOptions options,
                                     boolean? externalCatalog
                                       )
```

`externalCatalog`

> This option controls whether returned instances of `ManufacturerInfo` contain information about external catalogs in fields `externalCatalogURL` and `externalCatalogName`. The default value of this option is equal to the effective value of option allData.

Furthermore, the default value of option `manufacturerConfig` is now determined as follows: If the effective value of `allData` is `true`, the default value of `manufacturerConfig` is `true` too. Otherwise, the default value of `manufacturerConfig` is `true` if and only if at least one of the effective values of options `address`, `copyright`, `distributorName`, or `externalCatalog` is `true`.

5.6.3.12 `getSubItemIds`

Synopsis:

```
    BItemId[] getSubItemIds(SessionId sessionId,
                            BItemId   itemId,
                            ViewId    viewId)
           throws BasketServiceFault;
```

The `getSubItemIds` operation returns a sequence of basket item IDs for the child items of the basket item identified by the `itemId` parameter.

The `itemId` parameter must identify a basket item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `itemId` parameter may identify basket items of any type. If the item does not have child items, the operation returns an empty sequence.

If the argument `viewId` is present and neither an empty string nor the NIL-UUID then it must be the ID of a current basket view, the `itemId` argument must be the ID of an item of this view, and the returned item IDs will be IDs of items of this view.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify a basket item.

5.6.3.13 `getAllItems`

Synopsis:

```
    BasketItem[] getAllItems(SessionId              sessionId,
                             BItemId                itemIds,
                             GetAllItemsOptions     options)
           throws BasketServiceFault;
```

The `getAllItems` operation returns basic information about all items of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter and the items specified in the `itemIds` parameter. Return value is a list of `BasketItem` structures in the same order as produced by a pre-order traversal of the basket item hierarchy, starting with the (invisible) top folder. See section 5.6.1.8 for more information.

The final order of selected items is determined by a pre-order traversal of the basket item hierarchy.

The option `viewId` of `ItemSelectionOptions` is supported by this operation. If it contains the ID of a view then the behavior of the operation is changed as follows:

- Item IDs passed with parameter `itemIds` are interpreted as IDs of items within this view.
- The options `parentItems` and `subItems`, if set, affect the selection of additional view items.

Instances of `BasketItem` returned for view items do not contain fields `mainArticleId`, `subArticleIds`, `setArticleId`, `setArticlePartIds` and `geometryId`. Instances of `BasketItem` returned for referenced basket items (field `basketItems`) contain or may contain (depending on options `setArticleIds` and `geometryIds`) these fields.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- If the options parameter is present and duplicate item Ids occur.

5.6.3.14 InsertArticleOptions

Synopsis:

```
struct InsertArticleOptions (
                    boolean noComplexType,
                    AttachmentMode attachmentMode,
                    PlanningDirection planarPlanningDirection,
                    SpatialPlanningDirection spatialPlanningDirection,
                    boolean strictVarCode,
                    boolean migrate,
                    string attachPoint
                                )
```

The optional options parameter references an instance of the InsertArticleOptions structure with the following optional fields:

`noComplexType`

Basically, a single article can be inserted as an ordinary OFML article (item type Article) or as a complex article1 (item types Aggregate and PartialPlanning). The OFML data for a particular article may support insertion as an ordinary article or insertion as a complex article, or both. If both possibilities are supported, the noComplexType option, if set to true, causes the article to be inserted as an ordinary article. The default value is false (i.e., articles are inserted as complex articles whenever possible).

5.6.3.15 InsertOCDArticle (deprecated)

Synopsis:

```
BItemId insertOCDArticle(SessionId  sessionId,
                         BItemId    fatherId,
                         BItemId    beforeId,
                         InsertInfo insertInfo)
        throws BasketServiceFault;
```

With EAIWS 4.14 this operation is removed in the SOAP service! Please use insertOFMLArticle instead.

The `insertOCDArticle` operation inserts a new OCD article item into the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `fatherId` parameter, if not the null-UUID, must identify the basket item that will become the parent of the new OCD article item. If `fatherId` is the null-UUID (§5.2.1), the root item of the basket item tree will be used as the parent item.

The `beforeId` parameter, if not the null-UUID, must identify an existing basket item. If this item is a descendant of the basket item that has been selected as the parent of the newly inserted item, then the new item will be inserted in front of the parent's child which is either equal to the item identified by `beforeId`, or an ancestor of this item. Otherwise (if `beforeId` is a null-UUID, or if the basket item identified by `beforeId` is not a descendant of the parent item) the new item will be inserted as the last child of the parent item.

The `insertInfo` parameter specifies the OCD article to be inserted. For more information see §5.6.1.46.

If the operation was able to find the requested article in the product data it determines the initial configuration of the article, evaluating the variant code if specified, and, if successful so far, inserts the article into the basket item tree.

The inserted basket item is of type `OCDArticle` (§5.6.1.7).

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- Either the `fatherId` or `beforeId` parameter does not represent an UUID.
- Either the `fatherId` or `beforeId` parameter does not identify a basket item and is not the null-UUID.
- The basket item identified by the `fatherId` parameter does not permit child items.
- If an article package ID has been specified and it is either not a valid OFML package ID or it does not identify a package referenced by the configuration of the current session.
- If no article package ID has been specified and the Online Configurator was unable to find an OFML package matching the specified manufacturer and series (if specified) IDs.
- No base article number has been specified.
- The product data does not contain an article with the same manufacturer ID (if specified), series ID (if specified), and base article number.
- An invalid variant code has been specified, i.e. the variant code does not match the variant code scheme of the article.
- There was some error evaluating the relation knowledge of the article.
- There was some error accessing the product data.

5.6.3.16 `insertOFMLArticle`

Synopsis:

```
BItemId insertOFMLArticle(SessionId              sessionId,
                          BItemId                fatherId,
                          BItemId                beforeId,
                          InsertInfo             insertInfo,
                          InsertArticleOptions*  options)
        throws BasketServiceFault;
```

The `insertOFMLArticle` operation inserts a new OFML article item into the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `fatherId` parameter, if not the null-UUID, must identify the basket item that will become the parent of the new OFML article item. If `fatherId` is the null-UUID (§5.2.1), the root item of the basket item tree will be used as the parent item.

The `beforeId` parameter, if not the null-UUID, must identify an existing basket item. If this item is a descendant of the basket item that has been selected as the parent of the newly inserted item, then the new item will be inserted in front of the parent's child which is either equal to the item identified by `beforeId`, or an ancestor of this item. Otherwise (if `beforeId` is a null-UUID, or if the basket item identified by `beforeId` is not a descendant of the parent item) the new item will be inserted as the last child of the parent item.

The `insertInfo` parameter specifies the OCD article to be inserted. For more information see §5.6.1.46.

For attachment modes other than `None`, the item specified by the `fatherId` passed to the 'insertOFMLArticle' operation must be an basket article item that is part of a composite article.

(Note that a composite article may consist, in the most trivial case, of a single item. Also note that its usually not possible to insert an arbitrary article into a particular composite article. Either the composite article must be a special composite article that accepts any kind of article item (i.e. some kind of partial planning), or the composite article and the article to be inserted must have been designed to be used together.)

`planar`

```
enum PlanningDirection{
            Right_Front,
            Right_Back,
            Left_Front,
            Left_Back,
            Front_Right,
            Front_Left,
            Back_Right,
            Back_Left}
```

Default: `Right_Front`

`spatialPlanningDirection=Horizontal|Vertical`

Default: `Horizontal`

Specification of planning directions only affects insertion into a composite article. If the new article is inserted as a stand-alone article, the planning directions are ignored.

`strictVarCode=true|false`

Default: `false`

The `strictVarCode` flag affects OFML article instantiation as follows:

- If false, OFML method `setCreationMode(0)` is called before the article is instantiated. Once the article has been instantiated, its configuration is restored using OFML method `setXArticleSpec()`. Finally, OFML method `assignDefaultPropValues()` is called to apply custom configuration profiles (not implemented right now).
- If true, OFML method `setCreationMode(1)` is called before the article is instantiated, and its configuration is restored using OFML method `setupConfiguration()`. There is no call of OFML method `assignDefaultPropValues()`, so the variant code of the inserted article should conform to the specified variant code.

`migrate=true|false`

Default: `false`

If option `strictVarCode` is set to `true` then this option can be used to allow insertion of an article even if the configuration specified by the variant code is not supported by the product data in use, and thus cannot be restored exactly[63].

The operation first searches registered catalogs for the requested article. If the supplied insert information contains a catalog identifier, only this catalog is considered, and the catalog and article packages, if specified, must be part of the catalog. Otherwise, if either a catalog package and/or an article package has been specified, the operation considers the set of catalogs that contain the specified package or packages. Otherwise, the insert information must contain a commercial manufacturer identifier, and the operation considers all catalogs containing articles for this manufacturer.

If, at this point, no catalogs could be determined, the operation fails. Otherwise, if there is more than one eli-

[63] This condition manifests itself with a `BasketServiceFault`, with `egr.eai.basket.ArticleMigrationRequiredException` as its cause.

gible catalog, the catalogs are sorted according to their validity, release date, and the current date[64]. The sorted list of catalogs is searched for the first catalog containing at least one catalog entry matching the given series (if specified), article package (if specified) and base article number. Subsequent catalogs are not considered.

If multiple catalog entries are found, and a catalog package is specified, and at least one of the found catalog entries originates from the specified catalog package, then the set of catalog entries is reduced to the set of entries originating from the specified catalog package.

If, at this point, the remaining catalog entries reference OFML articles from different packages, the insert operation fails[65].

Finally, from the set of remaining catalog entries, the ‘best’ catalog entry is chosen to be inserted[66].

The inserted basket item is of type `Article`, `Aggregate`, or `PartialPlanning` (§5.6.1.7).

Note that the insertion of an OFML article may result in more than one new basket article item[67].

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- Either the `fatherId` or `beforeId` parameter does not represent an UUID.
- Either the `fatherId` or `beforeId` parameter does not identify a basket item and is not the null-UUID.
- The basket item identified by the `fatherId` parameter does not permit child items.
- If the catalog ID, if specified, does not adhere to the syntax of OFML catalog identifiers or does not identify a catalog referenced by the current session configuration.
- If the catalog package ID or article package ID, if specified, does not adhere to the syntax of OFML package identifiers, does not identify a package referenced by the current session configuration, or, if a catalog has been specified, the referenced package is not part of the catalog.
- No base article number has been specified.
- If the Online Configurator was not able to find a catalog entry matching the given insert information.
- There was some error during the actual process of inserting the OFML article. There are too many reasons for the insertion process to fail to describe here, but they usually originate from some product data error.

5.6.3.17 `insertUserArticle`

Synopsis:

```
void insertUserArticle(SessionId sessionId,
                       BItemId fatherId,
                       BItemId beforeId,
                       string shortText)
        throws BasketServiceFault;
```

The `insertUserArticle` operation inserts a new user article item (also known as manual article) into the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `fatherId` parameter, if not the null-UUID, must identify the basket item that will become the parent of the new OFML article item. If `fatherId` is the null-UUID (§5.2.1), the root item of the basket item tree will be used as the parent item.

The `beforeId` parameter, if not the null-UUID, must identify an existing basket item. If this item is a descendant of the basket item that has been selected as the parent of the newly inserted item, then the new

64 At present, the sort order is as follows: valid catalogs come first, followed by catalogs not yet valid, followed by catalogs no longer valid. Within each group, more recently released catalogs come first.
65 This can not happen if an article package has been specified.
66 The selection of the ‘best’ catalog entry is based on the comparison of the catalog entry’s variant code and the specified variant code, if any. The actual algorithm used for the comparison is implementation-defined.
67 The best example is the insertion of a meta type where the meta type and main child both require their own order position.

item will be inserted in front of the parent's child which is either equal to the item identified by `beforeId`, or an ancestor of this item. Otherwise (if `beforeId` is a null-UUID, or if the basket item identified by `beforeId` is not a descendant of the parent item) the new item will be inserted as the last child of the parent item.

The parameter `shortText` should contain the short description of the article.

The inserted basket item is of type `UserArticle` (§5.6.1.7). Additional properties of the newly inserted user article item may be set using the `setItemProperties` operation (§5.6.3.33).

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- Either the `fatherId` or `beforeId` parameter does not represent an UUID.
- Either the `fatherId` or `beforeId` parameter does not identify a basket item and is not the null-UUID.
- The basket item identified by the `fatherId` parameter does not permit child items.
- There was some error during the actual process of inserting the user article item.

5.6.3.18 `insertFolder`

Synopsis:

```
BItemId  insertFolder(SessionId sessionId,
                      BItemId   fatherId,
                      BItemId   beforeId,
                      string    name)
          throws BasketServiceFault;
```

The `insertFolder` operation inserts a new folder item into the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter. It returns the item ID of the inserted folder.

The `fatherId` parameter, if not the null-UUID, must identify the basket item that will become the parent of the new OFML article item. If `fatherId` is the null-UUID (§5.2.1), the root item of the basket item tree will be used as the parent item.

The `beforeId` parameter, if not the null-UUID, must identify an existing basket item. If this item is a descendant of the basket item that has been selected as the parent of the newly inserted item, then the new item will be inserted in front of the parent's child which is either equal to the item identified by `beforeId`, or an ancestor of this item. Otherwise (if `beforeId` is a null-UUID, or if the basket item identified by `beforeId` is not a descendant of the parent item) the new item will be inserted as the last child of the parent item.

The parameter `name` should contain the name of the folder.

The inserted basket item is of type `Folder` (§5.6.1.7). Additional properties of the newly inserted folder item may be set using the `setItemProperties` operation (§5.6.3.33).

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- Either the `fatherId` or `beforeId` parameter does not represent an UUID.
- Either the `fatherId` or `beforeId` parameter does not identify a basket item and is not the null-UUID.
- The basket item identified by the `fatherId` parameter does not permit child items of type `Folder`.
- There was some error during the actual process of inserting the folder item.

5.6.3.19 `insertTextItem`

Synopsis:

```
    BItemId  insertTextItem(SessionId sessionId,
                            BItemId   fatherId,
                            BItemId   beforeId,
                            string    text)
            throws BasketServiceFault;
```

The operation `insertTextItem`, behaving more or less like `insertFolder`, except that the inserted item is a text instead of a folder item, and the parent item does not need to be a folder item.

5.6.3.20 `deleteItems`

Synopsis:

```
    void deleteItems(SessionId                   sessionId,
                     BItemId[]                   itemIds,
                     DeleteItemsOptions          options)
            throws BasketServiceFault;
```

The `deleteItems` operation deletes the basket items identified by the elements of the `itemIds` sequence parameter from the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The options inherited from `ItemSelectionOptions` are handled in the usual manner. The option `subArticles` defined for `DeleteItemsOptions` enables (if specified as `true`) the deletion of individual sub-articles of composite articles.

All elements of the `itemIds` sequence parameter must identify existing basket items. If at least one item ID does not identify an existing item then no items will be deleted.

Duplicate item IDs are considered only once.

The root of the basket item tree cannot be deleted. If specified, it will be silently ignored.

It is not an error if the `itemIds` parameter is an empty sequence.

If one of the identified basket items has independent child items[68] which are not selected for deletion and `DeletionItemsOptions` is specified as `false`, they will not be deleted. Instead, they will replace their deleted parent. The order of remaining basket items, as determined by tree-traversal after deletion, will be identical to the relative order of the same items prior to deletion.

If an item to be deleted is the main article item of a compound article, all items that belong to the compound article are deleted, and independent child items of all deleted items are handled as described above.

If an item to be deleted is a sub-article item of a compound article, the operation silently ignores this item.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- At least one of the elements of the `itemIds` sequence parameter is not a valid string representation of an UUID.
- At least one of the elements of the `itemIds` sequence parameter does not identify an existing basket item.

[68] The Online Configurator supports 'compound articles' that consist of more than one individual basket article item, one of them being the 'main article item'. If the main article item of such a compound article gets deleted, all other items belonging to the compound article (whether or not they are children of the main item) will be deleted too.

5.6.3.21 `mergeBasketArticles`

Synopsis:

```
MergeResult mergeBasketArticles{
      string sessionId,
      string[] itemIds,
      ItemSelectionOptions options
}
```

Items to consider for merging are selected using the 'itemIds' and item selection 'options' parameters. View items are mapped to basket items. Only basket main article items and user article items are considered for merging. Two such items are merged if:

```
- they have the same parent
- they have identical sub-positions (children in basket item tree)
- they have identical fields and additional texts
- they are either not part of a set-article, or part of the same
  set-article
- they have the same calculations
- in case of basket main article items with sub-articles:
  - they represent identical composite articles
  - the above conditions hold for matching sub-articles, except that
    they do not have the same sub-positions
```

The return value of the operation is of complex type MergeResult and contains elements <addedItems>, <removedItems>, <movedItems> and <updatedItems>, one element for each added, removed, moved and updated item.

Obviously, whenever some merging took place there should be at least one removed and one updated item.

With the current implementation there should never be an added item.

Under certain circumstances there may be moved items. Consider two identical composite articles (A1,B1) and (A2,B2) where A? is the main article and B? the sub-article. They shall all be children of the same parent C. If A1 and A2 are merged and B2 has some child D then D becomes a child of C if A2 is merged into A1, thus D is reported as a moved item. If, on the other hand, A1 is merged into A2 then D does not move.

5.6.3.22 `SplitUpCompositeArticles`

Synopsis:

```
BitemId[] splitUpCompositeArticles(SessionId              sessionId,
                                   BitemId[]              itemIds,
                                   ItemSelectionOptions?  Options)
    throws BasketServiceFault;
```

The operation `splitUpCompositeArticles` splits up a composite article (usually a meta type) into individual ordinary articles and folders (in case the composite article contained partial plannings).

The operation first determines a set of selected items based on the specified item IDs and optional item selection options (as of now, option `viewId` is not supported).

For each main article item (root of a composite article) that is part of the set of selected items, the operation then performs the following, starting with the main article item:

1. If the original item is a partial planning, create a folder as predecessor of the original item. The item's short text is used as the folder's label.
2. If the original item is an article, create an ordinary article item as predecessor of the original item, and initialize it with the original article's OFML data, quantity, custom texts and field val-

ues, changes to item calculations (added manual conditions, removed conditions, manually specified condition values), tax information, packaging information, and alternative position state.

3. If the original item is part of a set-article, make the new item a part of the same set-article.
4. Move all children (sub-positions, not sub-articles) of the original item to the new item, preserving their order.
5. Recursively iterate over all sub-article items, repeating above steps until all sub-article items have been converted.
6. Delete the original main article item.

If conversion fails for whatever reason (not expected to happen), items newly created for the current partially converted composit article are deleted. This may result in structural changes (items that had been sub-positions of the main article item may become siblings of the main article item).

The operation returns a sequence of basket item IDs with the same number of elements as passed to the `itemsId` parameter. If the N-th parameter is the ID of an item that was part of a composite article that has been split up, then the N-th return value is the ID of the ordinary article item or folder that represents the original article now. Otherwise, the N-th return value is equal to the N-th parameter if the item still exists, or the NIL-ID if the item has been deleted for whatever reason (should not happen).

5.6.3.23 `moveItems`

Synopsis:

```
    MoveItemsResult[] moveItems(SessionId          sessionId,
                                MoveItemsDirection direction,
                                BItemId[]          itemIds)
            throws BasketServiceFault;
```

The `moveItems` operation moves the basket items identified by the elements of the `itemIds` sequence parameter in the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter. The `direction` argument specifies whether the specified items are to be moved up or down within their current folder, or are to be indented or unindented.

In all cases, the `itemIds` parameter must specify a possibly empty set of existing items all having the same parent item. The top folder must not be part of this set. If one of these conditions is violated, the operations fails with a BasketServiceFault.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The the `itemIds` parameter specifies a set of existing items that don't have the same parent item.
- The top folder is part of the `itemIds` parameter.

5.6.3.24 `relocateItems`

Synopsis:

```
    RelocateItemsResult[] relocateItems(SessionId              sessionId,
                                        BItemId                fatherId,
                                        BItemId                beforeId,
                                        BItemId[]              itemIds,
                                        RelocateItemsOptions   *options)
            throws BasketServiceFault;
```

Move specified list of items to new position in basket item tree.

The new position of the moved items is specified by `fatherId` and `beforeId`. They become children of the item identified by the father ID, and are inserted into the list of the father's children immediately before the item identified by beforeId, unless `beforeId` is either the `NIL` ID, or identifies an item that is not a child of the father, in which case they are appended to the list of children.

The items to move, and their order once they have been moved, is specified by `itemIds`. As the order is significant, duplicate item IDs are not allowed.

In general, the specified items are moved together with all their descendants (i.e. whole sub-trees are moved). The option `flat` (`false` by default) controls the behavior if one of the items to move is a descendant of another item to move (the ancestor). If `true`, the sub-tree rooted at the descendant will be moved to the father. If `false`, the descendant will be moved as part of the sub-tree rooted at the ancestor.

If option `doNotModify` (`false` by default) is `true`, the final step of moving the items is suppressed. Other than that, the operation behaves exactly the same way as with this option set to `false`, including returning the same result and throwing the same exceptions, unless they occur during the actual move of the items (which should not happen).

Parameters:

`fatherId`
: the ID of the new father

`beforeId`
: the insert position within the list of children of the new father, or a NIL UUID (either an empty string or an UUID consisting of all zero)

`itemIds`
: the list of IDs of the items to move

`options`
: the options that affect the behavior of the operation; If missing, or if one of the option values is missing, the default value will be used.

The operation returns an enumeration value that indicates whether or not items have been moved, and if not, why they haven't been moved

5.6.3.25 `convertToSetArticle`

Synopsis:

```
string convertToSetArticle(SessionId       sessionId,
                             BItemId         folderId,
                             OperationMode   opMode)
        throws BasketServiceFault;
```

This operation converts the specified basket folder into a set-article. Upon successful completion, the folder item specified by the `folderId` parameter has been replaced by a new set-article item, all children of the folder item are now children of the set-article item, and a subset of the descendants of the set-article item (usually all) is part of the set-article. The item ID of the set-article item is returned.

The flags in the operation mode parameter `opMode` control the behaviour of the operation as described below.

The following list describes in detail how the set of parts of the new set-article is constructed, how special conditions are handled, and how the actual conversion of the folder to the set-article is finally done:

- Descendants of the folder that can never be part of a set-article due to their type are ignored (except for set-article items, see below).
- If at least one descendant is a set-article item, and the `breakUpSetArticles` flag is set in `opMode`, all such set-article items and their parts are added to the set of new parts.
- For each descendant that is neither a set-article item, a set-article part, or a sub-article item, add the item to the set of new parts.

- If at least one descendant is a sub-article item whose main article item is neither an element of the set of new parts, nor part of a set-article, and the `mainArticles` flag is set in `opMode`, then ensure that the main article items of all such sub-article items are elements of the set of new parts, adding them if necessary.
- If the set of new parts contains one or more set-articles, the set-articles are broken up and the set-article items are replaced in the set of new parts with the folders created in place of the set-article items.
- The new set-article item is created as an immediate sibling of the folder item, all children of the folder item are moved to the set-article item (including all their descendants), and the folder item is deleted.
- All items in the set of new parts become parts of the new set-article item.

5.6.3.26 `breakUpSetArticle`

Synopsis:

```
string[] breakUpSetArticle(SessionId sessionId, BItemId setArtId)
        throws BasketServiceFault;
```

This operation converts the specified set-article item into a basket folder item.

The operation converts a set-article item into a basket folder item as follows: A basket folder item is created as a sibling of the set-article item, immediately preceding the set-article item. All parts are removed from the set article item (i.e. they become independent items). All children of the set-article item are moved to the folder item. The set-article item is deleted. Finally, the ID of the newly created folder item is returned.

5.6.3.27 `collapseSetArticles`

Synopsis:

```
void collapseSetArticles(SessionId sessionId, BItemId[] itemIds)
        throws BasketServiceFault;
```

This operation collapses zero or more set-articles.

Depending on the set-article mode of the basket view and the set-article-collapsed property of the set article item, set-articles are displayed expanded (the set-article item and its parts are visible) or collapsed (only the set-article item is visible). This operation affects the set-article-collapsed property of the specified set-article items, and thus the display of these set-article items in views using SetArticleMode.Dynamic. The operation iterates over all items specified by the itemIds parameter and collects all set-article items whose set-article-collapsed property is false. Then it iterates over all set-article items thus found and sets their set-article-collapsed property to true.

As EAIWS does not yet support basket views, this operation has no effect on the basket item structure. However, it may be useful if the session is saved to a project file and loaded into an application that supports views.

5.6.3.28 `expandSetArticles`

Synopsis:

```
void expandSetArticles(SessionId sessionId, BItemId[] itemIds)
        throws BasketServiceFault;
```

This operation expand set-articles. It is the inverse of collapseSetArticles.

5.6.3.29 `addToSetArticle`

Synopsis:

```
void addToSetArticle(
            SessionId        sessionId,
            string[]         itemIds,
            OperationMode    opMode)
                throws BasketServiceFault;
```

This operation adds zero or more items as set-article parts to the set-article represented by their first set-article item or part ancestor. The flags of the operation mode 'opMode' control the behavior of the operation as described below.

The operation takes a set of items, represented by the item IDs in the 'itemIds' parameter, and tries to convert them to set-article parts of the set-article represented by their first ancestor that is either a set-article item or set-article part. To do so, it creates a mapping from new parts to the set article they will be added to. Once this has been done, all items in this map are added to the respective set-article. The following list describes this process in detail:

1. For each set-article item in the set of items, test whether there is an ancestor that belongs to another set-article. If there is such an ancestor, and if the 'breakUpSetArticles' flag is set in 'opMode', then add the set-article item and its parts to the map of new parts, with the ancestor's set-article being the set-article they will be added to.
2. For each item, that is neither a set-article item, a set-article part, or a sub-article item, test whether there is an ancestor that belongs to a set-article. If there is one, add the item to the map of new parts, with the ancestor's set-article being the set-article the item will be added to.
3. If the 'mainArticles' flag is set in 'opMode', then test for each sub-article item, whose main article item is not already a member of the map of new parts, whether the main article item has an ancestor that belongs to a set-article. If there is one, add the main article to the map of new parts, with the ancestor's set-article being the set-article the main article item (and thus the sub-article item) will be added to.
4. If the map of new parts contains one or more set-articles, the set-articles are broken up and the set-article items are replaced in the map of new parts with the folders created in place of the set-article items, retaining the set-article they will be added to. Furthermore, if the map of new parts contains items to be added to the broken-up set-article, these items will now be added to the same set-article as the broken-up set-article and its parts.
5. Finally, all items from the map of new parts are added to the set-article specified in the map

5.6.3.30 `removeFromSetArticle`

Synopsis:

```
void removeFromSetArticle(
                          SessionId          sessionId,
                          BItemId[]          itemIds,
                          OperationMode[]    opMode)
                throws BasketServiceFault;
```

This operation removes parts of set-articles from their set-article.

The operation searches the specified set of items for all items that are part of a set-article, and removes these items from their set-article. The list below describes the procedure in detail.

The flags in the operation mode 'opMode' control the behavior of the operation as described below.

1. From the set of items specified by the `itemIds` parameter, remove all items not currently part of a set-article.
2. Search the remaining set of items for sub-article items of composite articles whose main article item

is not a member of this set. Add the main article items to the set if the `mainArticles` flag is set in `opMode`.

3. Remove sub-article items from the set.

5.6.3.31 `changeAlternativePositionState`

Synopsis:

```
boolean changeAlternativePositionState(SessionId             sessionId,
                                       BItemId[]             itemIds,
                                       ItemSelectionOptions  options,
                                       string                ppName,
                                       boolean               altPos)
        throws BasketServiceFault;
```

This operation can be used to modify the excluded calculation set of zero or more article items.

The `itemIds` and `options` parameters are used to compute an initial set of selected items. Items other than basket main article items, user article items or set article items are removed from this set. The excluded calculation sets of the remaining items are updated as follows:

If the `altPos` parameter is `true`:

- If the current excluded calculation set is the universal set, it is left unchanged.
- Otherwise, if the value of the `ppName` parameter is an empty string, the excluded calculation set is set to the universal set.
- Otherwise, the value of the `ppName` parameter is added to the excluded calculation set.

If the `altPos` parameter is `false`:

- If the value of the `ppName` parameter is an empty string, the excluded calculation set is set to the empty set.
- Otherwise, if the excluded calculation set is the universal set, it is first replaced with a set containing the names of all price calculations currently used by the basket. Then, the entry equal to the value of `ppName` is removed from the excluded calculation set.

The operation fails if the session ID or one of the item IDs is invalid, or the specified calculation (pricing procedure) name is neither empty nor equal to the name of one of the price calculations currently used by the basket.

5.6.3.32 `getItemProperties`

Synopsis:

```
ItemProperties[] getItemProperties(SessionId                  sessionId,
                                   BItemId[]                  itemIds,
                                   GetItemPropertiesOptions options)
        throws BasketServiceFault;


struct GetItemPropertiesOptions : ItemSelectionOptions {
                        DescrType[] tmDescrMode,
                        string[] extraPriceInfoSelectors
                        boolean priceInfo,
                        string pricingProcedureName,
                        boolean positionNumbers,
                        boolean addStateCodes,
                        GetItemPropertiesTextMode textMode,
                        boolean preferUserDescriptions,
                        boolean separateCurrencies,
                        boolean? IncludeCalculationErrorsInInconsistencies,
                        boolean? InactivePositionState,
                        boolean?  articleClassifications
                                                        }
```

The `getItemProperties` operation returns the item properties (not the OFML or OCD properties) of the basket items specified by the `itemIds` parameter. The basket items must be part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

If `itemIds` parameter is an empty sequence, then the properties of all basket items will be returned. Otherwise, the sequence of returned `ItemProperties` structures contains one entry for each passed item ID, with the order of returned `ItemProperties` structures equal to the order of passed item IDs.

Unless explicitly specified otherwise, all fields of the `ItemProperties` structures (§5.6.1.20) returned by this operation, and all fields of the `FolderProperties` (§5.6.1.21) and `ArticleProperties` (§5.6.1.27) structures referenced by the `ItemProperties` structure, are non-null.

The `folder` and `article` fields of the `ItemProperties` structure are non-null if and only if the item is a folder (field `folder`) or an article (field `article`).

The `finalArticleNumber` and/or `variantCode` fields of the `ArticleProperties` structure may be null if the Online Configurator failed to fetch their values from the product database.

The `currency`, `purchasePrice` and `salesPrice` fields of the `ArticleProperties` structure may all be null if the Online Configurator failed to fetch either price from the product database.

The `options` parameter has no effect if the `itemIds` parameter is empty, as all items are selected in this case anyway. If the itemIds parameter is not empty, the list of selected items is determined as follows:

If the options parameter is missing (null), the list of selected items consists of one entry for each ID in itemIds, in the same order.

If the options parameter is present (non-null), the initial list of selected items consists of one entry for each ID in itemIds, in the same order.

The final order of selected items is determined by a pre-order traversal of the basket item hierarchy.

The option `viewId` of `ItemSelectionOptions` is supported by this operation. If it contains the ID of a view then the behavior of the operation is changed as follows:

- Item IDs passed with parameter `itemIds` are interpreted as IDs of items within this view.
- The options `parentItems` and `subItems`, if set, affect the selection of additional view items.
- Instances of `ItemProperties` returned for view items
  - contain the view item ID instead of the basket item ID in field `itemId`,
  - do not contain fields `visible` and `expanded`,
  - contain the ID of the view item representing the set-article instead of the set-article's basket item ID in field `setArticleId`,
  - use field `positionNumber` to return a position number, and
  - use field `composableGeometries` instead of `composableGeometry` to return composable geometry information.
- Instances of `ArticleProperties` returned for view items differ from those instances returned for basket items as follows:
  - The `quantity` field contains the total quantity of all referenced basket article items in case of merged article view items.
  - The value of the `pricingProcedureName` option is used to determine the value of the deprecated `alternativePosition` field.
  - If the `priceInfo` option is `true`, the `priceInfo` field may be used to return price information

obtained from the article item's calculation named by option `pricingProcedureName`.

  - In case of merged article view items, fields `ofmlUpdateState` and `geometryId` return the value of one randomly chosen basket item. This behavior may change in future versions of EAIWS. Clients should use the geometry IDs returned by operation `getAllItems` (with option `basketItems` set to `true`) instead of the geometry IDs returned by operation `getItem-Properties`.
  - In case of item properties returned for article items of views with merge mode `Compact`, field `subArticles` may be used to return information of sub-articles folded into the view item that represents the whole composite article.

- Instances of `SetArticleProperties` returned for view items differ from those instances returned for basket items as follows:
  - Field `collapsed` takes the set-article mode of the view into account. In case of `Expand` or `Collapse` the value will always be `false` or `true`, respectively. In case of `Dynamic` the value will be equal to the value returned for the basket's set-article item[69].
  - Returned set-article part IDs are IDs of view items, not basket item Ids.

- tmDescrMode - a possible empty list of DescrType; If the list is not empty, a description composed from article text table texts is returned in element <tmDescription> of complex type ItemProperties. The description is built from non-empty texts in the article text table, in row order, except that texts that are marked as invisible and texts not selected by this option are skipped.

Type `GetItemPropertiesOptions` defines the following options:

`priceInfo`

Controls whether or not summary price information should be returned in field `priceInfo` of type `ArticleProperties`. Price information is returned if the value of this option is `true`, return of price information is enabled[70] and valid price information could be determined.

Return of this price information is made optional as it requires computation of calculation data, which may be a bit expensive, especially if the use of group calculations is required, and unnecessary if the client will fetch complete calculation data anyway.

The default value of this option is `false`.

`pricingProcedureName`

Contains the name of the pricing procedure to use to fetch price information. The pricing procedure name is also used to compute the value of the deprecated `alternativePosition` property.

If option `priceInfo` is `true` and return of price information is enabled then the pricing procedure name must be empty and the basket must use exactly one price calculation, or the pricing procedure name must be equal to the name of one of the price calculations used by the basket.

The default value of this option is an empty string.

`positionNumbers`

This option is ignored if `getItemProperties` is called for basket items. If called for view items, the option controls whether or not position numbers are returned using field `positionNumber` of type `ItemProperties`.

The default value of this option is `false`.

`includeCalculationErrorsInInconsistencies`

If true, validation errors in all item calculations of an article item are considered an inconsistency of that article item. If false, only OFML article inconsistencies are used to set fields inconsistencyFlag

69 the basket item, not the view item

70 application feature `egr.eai.ws.basket.ReturnPriceInfo` is available

and inconsistencyReason of complex type ArticleProperties.

The default value of this option is `true`.

`articleClassifications`

The option is used to control whether operation getItemProperties returns article classifications. The default value of this option is `false`.

`inactivePositionState`

If true, operation getItemProperties returns information about inactive positions in field inactivePositionState of type ArticleProperties. Fields alternativePosition and excludedCalcualtions are not returned. Otherwise, if false (the default value), information about inactive positions is returned as before in fields alternativePosition and excludedCalculations, and inactivePositionState is not returned.

The default value of this option is `false`.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- At least one of the elements of the `itemIds` sequence parameter is not a valid string representation of an UUID.
- At least one of the elements of the `itemIds` sequence parameter does not identify an existing basket item.
- The Online Configurator failed to fetch some of the properties due to an internal error.
- If the options parameter is present and duplicate item Ids occure.

5.6.3.33 `setItemProperties`

Synopsis:

```
void setItemProperties(SessionId sessionId,
                       BItemId itemId,
                       ItemProperties properties)
        throws BasketServiceFault;
```

The `setItemProperties` operation can be used to set some of the item properties (not the OFML or OCD properties) of the basket item identified by the `itemId` parameter.

The `itemId` parameter must identify a basket item which is part of the basket item tree maintained by the basket service for the session specified by the `sessionId` parameter.

Fields of the passed `ItemProperties` structure (§5.6.1.20), and fields of the `FolderProperties` (§5.6.1.21) and `ArticleProperties` (§5.6.1.27) structures referenced by the `ItemProperties` structure, are ignored if

- they are null,
- are irrelevant for the basket item's type, or
- correspond to read-only properties of the item.

Thus, depending on the item type (§5.6.1.7), the following fields, if non-null, are used to set the corresponding item properties:

`Folder`

`folder.name`
`folder.showSubTotal`

`Article, Aggregate, PartialPlanning, UserArticle, SetArticle`

`article.quantity`

Setting these properties for a sub-article item actually sets them for the main article item. Setting them for the main article item of a composite article also affects the sub-articles of the composite article (including the sub-articles of sub-articles, if any).

`UserArticle, SetArticle`

```
article.manufacturerId
article.seriesId
article.baseArticleNumber
article.finalArticleNumber
article.variantCode
article.shortText
article.longText
article.featureText
```

`UserArticle`

```
article.currency
article.purchasePrice
article.salesPrice
article.packagingInfo
```

The value of fields that are ignored never causes the operation to fail. The value of the following fields, if not ignored, is validated before any property of the basket item is set, and a failed validation causes the operation to fail before any item property is set:

`article.manufacturerId, article.seriesId`

The manufacturer and series IDs, if not empty, must adhere to the constraints specified in §5.6.1.27.

`article.currency`

The currency must be an ISO 4217 currency code (pseudo-currency code are not supported), even if both the purchase price and sales price are ignored.

`article.purchasePrice, article.salesPrice`

These fields, if not ignored, require the `currency` field to contain a valid ISO 4217 currency code (pseudo-currency code are not supported).

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter is not a valid string representation of an UUID.
- The `itemId` parameter does not identify an existing basket item.
- The validation of fields, as described above, failed.
- The attempt to set one of the properties failed due to an internal error.

5.6.3.34 `getArticleData`

Synopsis:

```
ArticleData getArticleData(SessionId sessionId,
                           BItemId itemId,
                           GetArticleDataOptions* options)
        throws BasketServiceFault;
```

The `getArticleData` operation returns all product data about the current configuration of a single article.

The `itemId` parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

For a description of the content of the returned `ArticleData` structure see 5.6.1.42.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an

open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify an article item.
-
- The `noProperties` option was not set to `true` and the operation was not able to fetch the article's properties.
- For whatever reason the Online Configurator considered it necessary to reevaluate the product data of the article and re-evaluation of the product data failed, possibly due to invalid relation knowledge or a product database access error.

Changed operation getArticleData so it doesn't fetch choice lists of choice properties, thus speeding up the whole operation.

5.6.3.35 `getArticleFeatures`

Synopsis:

```
ArticleFeature[] getArticleFeatures(SessionId sessionId,
                                     BItemId itemId,
                                     GetArticleFeaturesOptions* options)
          throws BasketServiceFault;
```

The `getArticleFeatures` operation is used to fetch the article features of an article item.

The `itemId` parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

For more information about returned article features, see §5.6.1.43.

If the `getArticleFeatures` operation is called for an user article or set article, the returned sequence of article features is empty.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify an article item.

5.6.3.36 `getMultiArticleFeatures`

Synopsis:

```
ArticleFeatures[] getMultiArticleFeatures
                        (SessionId  sessionId,
                         UUID[] itemIds,
                         GetMultiArticleFeaturesOptions? Options)
          throws BasketServiceFault;
```

Operation `getMultiArticleFeatures` is a more powerful variant of operation `getArticleFeatures` (§5.6.3.35). It allows the client

- to query features of multiple articles in one go, and
- to use view item IDs instead of basket item IDs (if field `viewId` of parameter `options` is set accordingly).

The selection of items whose article features are to be returned is performed as usual using the basket or view item IDs specified with parameter `itemIds` and the subset of `ItemSelectionOptions` passed with parameter `options`. Once the initial set of selected items has been determined, non-article items are removed from the set (user- and set-article items are retained even though they always report an empty list of article features (at least with the current implementation)).

It is not an error to explicitly specify the ID of a non-article item, regardless of the set of item selection options being used. In particular, specification of the top folder ID in combination with item selection option `subItems` set to `true` may be used to fetch the features of all article items.

Once the set of article items has been determined, the operation fetches the article features of each article item and returns them using one element of type `ArticleFeatures` (§5.6.1.44) for each article item, with field `itemId` holding the basket or view item ID of the corresponding article item, and field `features` holding the list of individual article features as they would be returned by operation `getArticleFeatures`.

The order of `ArticleFeatures` elements in the return value is unspecified.

Note: Fields `description` and `noInternal` of type `GetMultiArticleFeaturesOptions` (§5.6.1.24) are encoded as attributes, even though the same fields of type `GetArticleFeaturesOptions` (§5.6.1.23) are encoded as elements.

5.6.3.37 getChoiceList

Synopsis:

```
PropertyValue[] getChoiceList(SessionId sessionId,
                              BItemId   itemId,
                              string    propClass,
                              string    propName,
                 GetChoiceListOptions  options
                              )
        throws BasketServiceFault;
```

The `getChoiceList` operation returns the list of allowed property values[71] for a single property.

The `itemId` parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `propClass` and `propName` parameters must identify a property of the article. The returned choice list is empty if the property happens to be an internal property[72] or a property explicitly hidden by relation knowledge, or if the `choiceList` field of the `Property` structure is `false`. Otherwise, the choice list is not empty and contains at least the current property value.

See 5.6.1.38 for a description of the `PropertyValue` structure which is used as the element type of the returned sequence.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify an article item.
- The `propClass` and `propName` parameters do not identify a property of the article.

5.6.3.38 getAllChoiceLists

Synopsis:

```
ChoiceList[] getAllChoiceLists(SessionId sessionId,
                               BItemId   itemId,
                  GetChoiceListOptions  options

                              )
        throws BasketServiceFault;
```

71 If the `addValues` field of the `Property` structure is `true`, then the choice list contains only proposed values.
72 This is condition is quite unlikely as the current version of the basket service does not expose internal properties.

The getAllChoiceLists operation returns a possible empty sequence of ChoiceList structures (§5.6.1.45), one for each property of the article that is visible, editable, and has a choice list[73].

The itemId parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the sessionId parameter.

A BasketServiceFault is returned if any of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.
- The itemId parameter does not represent an UUID.
- The itemId parameter does not identify an article item.

5.6.3.39 setPropertyValue

Synopsis:

```
     string setPropertyValue(SessionId sessionId,
                             BItemId   itemId,
                             string    propClass,
                             string    propName,
                             string    value
                            SetPropertyValueOptions? options)
             throws BasketServiceFault;
```

The setPropertyValue operation is used to assign a new value to an editable property of an article.

The itemId parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the sessionId parameter.

The propClass and propName parameters must identify an editable property of the article.

The value parameter contains the new property value.

If argument options is missing the operation behaves as before (all options controlling computation of change flags are effectively true).

For string properties that do not allow additional values the value must be equal to one of the values from the choice list.

For numeric properties and length properties, the string must consist of an optional minus sign (-) , followed by a sequence of one or more decimal digits, and, if the precision[74] of the property is greater than zero, followed by a decimal point and one or more decimal digits, but no more than specified by the precision[75]. If the property does not allow additional values and has no intervals then the value must be numerically identical to one of the values from the choice list. If the property has at least one interval then the value must be within one of the intervals, or, for OCD 4.0 and later, a member of the choice list.

Setting a property value usually involves the evaluation of the product data's relation knowledge. The setPropertyValue operation has no effect if, for whatever reason, evaluation of the relation knowledge fails.

The setPropertyValue operation returns a string consisting of zero or more characters, each character indicating a particular side effect of the assignment of the new property value. The following characters are defined:

S property **S**hown

An additional property became visible.

H property **H**idden

A property that has been visible before has been hidden, i.e. became invisible.

[73] The visible, editable and choiceList fields of the Property structure are all true.
[74] The precision is the value of the decDigits field of the Property structure.
[75] The Online Configurator may accept other number formats too, but formats other than the one described here may not be accepted by later versions of the Online Configurator.

`V`	other property **V**alue changed Assignment of the new property value caused a change of the value of another visible property.
`v`	invisible property **v**alue changed Assignment of the new property value caused a change of the value of an invisible property.
`U`	property value **U**nchanged Assignment of the new property value did not result in a change of the value of this property. One reason may be that the property was set to the original value. Another reason may be that the change was rejected.
`A`	property value **A**djusted Assignment of the new property value resulted in a change of the value of this property, but the new value is different from the assigned one. One reason may be that a numeric property value has been rounded to the precision of the property as specified by the product data.
`C`	**C**hoice list changed At least one of the property choice lists has changed.
`I`	**I**nterval changed At least one of the interval lists has changed.
`a`	position **a**dded Assignment of the new property value resulted in at least one position to be added to the basket item hierarchy.
`r`	position **r**emoved Assignment of the new property value resulted in the removal of at least one position from the basket item hierarchy.
`u`	other position needs **u**pdate Assignment of the new property value resulted in a change to at least one other position (in addition to the currently configured position).

In case of the ‘a’, ‘r’ and ‘u’ flags, the other position or positions affected always belong to the same hierarchy of composite articles (§5.6.1.8 for more information). Other than that there are no guarantees, in particular there is no guarantee that the other position or positions are immediate sub-articles of the modified position.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify an article item.
- The `propClass` and `propName` parameters do not identify an editable property.
- The property is a numeric or length property, but the new property value cannot be parsed as a decimal number.
- The property has a choice list and no intervals, does not allow additional values, and the new value is not a member of the choice list.
- The property has a choice list and at least one interval, the value is not within one of the intervals, and, for OCD 4.0 and later, the value is not a member of the choice list.
- Evaluation of relation knowledge failed due to invalid relation knowledge or a database access error.

5.6.3.40 `getGeneratedImage`

Synopsis:

```
URL getGeneratedImage(SessionId sessionId,
                      BItemId   itemId,
                      string[]  options)
        throws BasketServiceFault;
```

The `getGeneratedImage` operation is used to compute an image for the article identified by the `itemId` parameter.

The `itemId` parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `options` parameter is a sequence consisting of zero or more option strings, where each option string consists of a key and value, separated by an equal sign (`=`) (e.g. `"format=jpg"`). The following options are available (numbers delimited by horizontal ellipsis (…) specify minimum and maximum values, default values are displayed **bold**):

`tag=`*`string`*`:`***`default`***

The value of this option must be a valid Unicode identifier. Each article item stores a reference to the last image generated for this tag and item until the configuration of the article item is changed[76]. Furthermore, the `saveSession` (§5.4.3.14) and `loadSession` (§5.4.3.15) operations of the session web service save and load these images and each (article) item's mapping from tag to image.

An image returned by operation `getGeneratedImage` when invoked without option `tag`, with an empty tag, or with tag `default` is not used as the default generated article image for the current configuration of the article if the specified format is neither JPEG nor PNG.

`width=`*`int`*`:`*32…**512**…2048*

width of the image in pixel

If the `width` option has been specified, but not the `height` option, the specified value is used for both the width and height of the generated image. If neither option is specified, then the default image width is 512 pixels.

The `width` option must not be used if the `scale` option is used, as the `scale` option, in combination with the `margin` option, determines the width of the generated image.

For SVG, PDF, EPS and PS exports the value of `width` must be specified in point (1/72 inch). The default value is 384 pt. The maximum value is 3370 pt.

`height=`*`int`*`:`*32…**512**…2048*

height of the image in pixel

If the `height` option has been specified, but not the `width` option, the specified value is used for both the width and height of the generated image. If neither option has been specified, then the default image height is 512 pixels.

The `height` option must not be used if the `scale` option is used, as the `scale` option, in combination with the `margin` option, determines the height of the generated image.

For SVG, PDF, EPS and PS exports the value of `height` must be specified in point (1/72 inch). The default value is 384 pt. The maximum value is 3370 pt.

`format=`*`enum`*`:`***JPG***|***PNG***|*TGA*|*TIFF*|*SVG*|*PDF*|*EPS*|*PS*|*MCP*

format of the resulting image file

The default image file format depends on whether or not the `scale` option is used. PNG is used for true to scale images, JPEG otherwise.

Note: The export of SVG, PDF, EPS and PS is on an experimental stage. Important features still

76 In case of a composite article, images stored for aggregates and partial plannings containing the changed article are reset too.

missing are: transparent background, true-to-scale images. The other options supported and allowed are `width`, `height` and `hideSubArticles`. The value of `width` and `height` must be specified in point (1/72 inch). The default value is 384 pt. The maximum value is 3370 pt.

MCP exports Multi Content Picture.

`hideSubArticles=`*`boolean`*

Controls whether or not sub-articles of composite articles are hidden during image generation (and thus are not shown together with the composite article).

The default value depends on the exported image format. In case of `format=MCP`, the default value is `false`. In all other cases, it is `true`.

If the operation is invoked for a sub-article, then this sub-article is not hidden, regardless of the value of this option. If, however, the sub-article happens to be a composite article, then the option affects the hiding of sub-articles of this nested composite article.

`antialias=`*`boolean:`***`false`***

Controls the use of an anti-aliasing algorithm during image generation.

Note that the improved image quality is achieved at the cost of image generation time (about five times slower). Especially for large images this may make a significant difference.

The default value depends on the value of the `resample` option (`true` if `resample=1`, `false` otherwise).

`resample=`*`int:1…`***`2`***`…32`*

The `resample` option allows for the rendering of the image using a resolution greater than specified by the `width` and height `options`, or computed based on the scale factor for true-to-scale images. The required width and height are multiplied by the resampling factor to determine the image dimensions used during rendering. The rendered image is scaled down by the given factor to produce an image with the required width and height.

If the image dimensions used during rendering exceed a certain limit (4096x4096 in EAIWS version 3.0), the resampling factor is reduced automatically.

Rendering with `resample=4` and `antialias=false` usually results in about the same image generation time as `resample=1` and `antialias=true` (the default behavior prior to EAIWS 3.0 RC 2), but produces better results.

`quality=`*`int:10…`***`80`***`…100`*

quality of generated JPEG image

`ambient=`*`float:0.0…`***`0.2`***`…1.0 float:0.0…`***`0.2`***`…1.0 float:0.0…`***`0.2`***`…1.0`*

ambient color (brightness); The individual values are for red, green and blue.

In case of `renderMode=PBR` the option `ambient` can be used to control the brightness of the rendered image. The default value of option `ambient` with `renderMode=PBR` is `0.75 0.75 0.75`. For other render modes the default value continues to be `0.2 0.2 0.2`.

`background=`*`float:0.0…`***`1.0`*** *`float:0.0…`***`1.0`*** *`float:0.0…`***`1.0`***

color of background; The individual values are for red, green and blue.

If the selected image format supports an alpha channel, then the special value `transparent` may be used instead of the color components to produce an image with a transparent background.

`enable2D=`*`boolean:`***`false`***

If set to `true`, 2D-geometries are rendered at height zero.

For raster image formats, the `enable2D` option defaults to `false`. For vector graphics formats, the `enable2D` option defaults to `true`.

`enable2DZBuffer=`*`boolean:`***`false`***

If set to `true`, the Z-buffer is enabled during rendering of 2D-geometries. Consequently, 2D-geometries may be hidden by 3D-geometries.

The value of this option is processed, but otherwise ignored, if the value of the `enable2D` option is `false`.

The default value is `false`. Setting this option to `true` should affect the generated (raster) image if and only if both 2D and 3D geometries are rendered.

`enable3D=`*`boolean`*`:`***`true`***

Controls whether or not to render 3D-geometries.

Note that the `enable2D` option must be set to `true` if this option is set to `false`. Otherwise, the operation fails.

For raster image formats, the `enable3D` option defaults to `true`. For vector graphics formats, the `enable3D` option defaults to `false`.

`layerControl`

The argument of this option must be a string that adheres to the following grammar:

```
layer-control = [ global ] { specific }
global = "\*" | "!"
specific = onoff layer-spec { suffix }
onoff = "\+" | "-"
suffix = "r" | "i"
layer-spec = "'([^']|'')*'" | "\"([^\"]|\"\")*\""
```

(Character sequences enclosed in double quotes represent terminal symbols specified as POSIX Extended Regular Expressions, with the addition that \" represents an ordinary double quote character that does not terminate the character sequence.)

The layer control string is processed left to right. If `global` is present, all layers are initially switched on ('*') or off ('!'). 'layer-spec' is a string whose meaning depends on the possibly empty sequence of suffixes. If the 'r' suffix is present, it is a POSIX Extended Regular Expression, selecting all layers whose complete name matches the regular expression. Otherwise it is an ordinary string that must match a complete layer name. If the 'i' suffix is present, matching is done in a case-insensitive way. It is not an error if suffixes appear more than once. Selected layers are switched on if `onoff` is '+', and off if `onoff` is '-'.

After the export, layers are switched back to their original state.

`renderMode=`*`enum`*`:`***`Textures`***`|`*`Color`*`|`*`GrayScale`*`|`*`BlackWhite`*`|`*`PBR`*

the rendering mode

Option `renderMode=PBR` enables Physically Based Rendering. Option `ambient` can be used to control the brightness of the rendered image. The default value of option `ambient` with `renderMode=PBR` is `0.5 0.5 0.5`. For other render modes the default value continues to be `0.2 0.2 0.2`.

`outline=`*`boolean`*`:`***`false`***

whether or not to outline edges; applies to render modes `Textures` and `Color` only

`fadeBorder=`*`int`*`:`*0…`<max-value>`*

controls soft fading-out of image margin. The parameter specifies the with of the margin in pixels. *<max-value>* is half the minimum of the width and height of the generated image.

If the `scale` option is used to generate a true to scale image, the `fadeBorder` option has no effect, and the upper bound of the option value is not checked.

`shadowPlane=`*`boolean`*`:`***`true`***

whether or not a shadow plane will be displayed below the article.

This option is ignored and the generated image will never contain a shadow plane if the `scale` option is used to generate a true to scale image, or if the viewpoint height is less than or equal to zero.

The following options, prefixed with `shadowPlane.`, are ignored if the generated image does not contain a shadow plane.

`shadowPlane.color=`*`float:0.0…`***`0.5`***`…1.0 float:0.0…`***`0.5`***`…1.0 float:0.0…`***`0.5`***`…1.0`*

affects the color of the shadow plane; The individual values are for red, green and blue.

The option is ignored if the generated image does not contain a shadow plane.

In case of `shadowPlane.filter=DOF` the default value is: `0.25, 0.25, 0.25`

`shadowPlane.smoothness=`*`float:0.001…0.2…1.0`*

Default value depends on shadowPlane.filter selection:

```
Lightmap: 0.02
Gauss, Linear, Invers: 0.2
Gauss2: 1.0
DOF: 0.05
```

affects the smoothness of the shadow plane boundary

Smaller values result in a sharper shadow boundary, whereas larger values result in a more blurred boundary. The option is ignored if the generated image does not contain a shadow plane.

`shadowPlane.mirror=`*`float:`***`0.0`***`…1.0`*

This option, if set to a non-zero value, causes the mirrored image of the rendered object to be displayed on the shadow plane.

`shadowPlane.colorMode=`*`enum:`***`Color`***`|GrayScale`*

This option controls whether the mirrored image on the shadow plane is displayed in color or gray scale.

`shadowPlane.filter=`*`enum:Gauss|Linear|Invers|Gauss2|Lightmap|DOF`*

If not set, or set to an empty string, the shadow is based on the bounding box of the object. This is the default behavior. Otherwise, the shadow is based on the distance between the shadow plane and individual parts of the object. The specified filter selects the algorithm used to spread the shadow over the shadow plane. Available filters are:

`Gauss`	very smooth
`Linear`	moderately smooth
`Invers`	hard at the center, outwards too smooth
`Gauss2`	smooth, based on geometry distance (recommended)
`Lightmap`	To support OSAO-based shadows on the shadow plane
`DOF`	Depth of Field algorithm

`shadowPlane.border=`*`float:`***`-1.0`***`…1.0`*

affects the size of the shadow plane

If the value is zero then the shadow is about as large as the object. Values less than zero result in smaller shadows, values larger than zero in larger shadows.

This option is ignored if the `shadowPlane.filter` option is set.

`shadowPlane.mapSize=`*`int`*

Specifies the size of the texture used for the shadow.

Attention: This parameter has a significant effect on rendering performance!

The default value is computed depending on the image size and the selected shadow plane filter. If `Gauss2 or Lightmap` is selected as the shadow plane filter, the default shadow plane map size is equal to the sum of image width and height divided by two. Otherwise, it is equal to the sum of image width and height first divided by two and then divided by eight (not necessarily the same as division by 16 due to the use of integer arithmetic). The minimum default map size is 1.

The default value for filter modes other than `Gauss2` is much smaller because rendering times grow excessively for values greater than 256, and furthermore other than with `Gauss2`, reasonable results are achieved with smaller map sizes.

The shadow plane map size has been limited to a maximum of 4096 in case of shadow plane filter mode `Gauss2` and 512 for all other filter modes, including no filter. Greater values result in excessive rendering times, especially in case of filter modes other than `Gauss2`.

In case of shadow plane filter mode `DOF` the default value is computed based on *clamp((width + height) / 2, 1, 4096)*[77]

This option is ignored if the `shadowPlane.filter` option is not set.

`shadowPlane.maxHeight=`*`float`*

Specifies the maximum height of objects considered for the generation of shadows. The effect on the shadow is maximal at 0.001 height (i.e. if the object is immediately above the shadow plane), and diminishes up the the specified height.

The default value is the height of the bounding box (relative to the xz-plane) of the object being rendered.

In case of shadow plane filter mode `DOF` the default value is `0.25`.

This option is ignored if the `shadowPlane.filter` option is not set.

`shadowPlane.radius=float`

For shadow plane filter `Lightmap` the default value is computed depending on the height of the bounding box.

`shadowPlane.intensity=`***`float:0.0…0.3…1.0`***

In case of linear color workflow[78] the default value is 1.0.

`shadowPlane.samples=`***`int:1…20…256`***

For lightmap-based shadow planes.

`projectionMode=`*`enum`*`:`**`Perspective`**`|`*`Orthographic`*

sets the projection mode used to render the image

The default projection mode depends on whether or not the `scale` option is used. Orthographic projection is used for true to scale images, perspective projection otherwise. In fact, specifying a projection mode other than `Orthographic` is not allowed for true to scale images.

`zoom=`*`float:1.0…`***`30.0`***`…89.0`*

`alpha=`*`float:-180.0…`***`-15.0`***`…360.0`*

`beta=`*`float:-90.0…`***`20.0`***`…90.0`*

The `zoom` (angle of view), `alpha` (longitude of view point) and `beta` (latitude of view point) parameters are used to compute both the angle of view and the position (view point) of the camera.

The center of the bounding box is always used as the reference point. The view point is located at the axis starting a the reference point and pointing into the direction specified by the `alpha` and `beta` angles.

If orthographic projection is used, a possibly specified zoom is ignored.

If perspective projection is used, the distance between view point and reference point is computed such that, given the specified zoom, the article plus some small margin is always visible. Furthermore, if the latitude of the view point has not been explicitly specified, the height of the view point is limited to about 1.7 meter.

If the `scale` option is not used, the default values for zoom, longitude and latitude are 30.0, -15.0 and 20.0.

[77] *Width* and *height* are the width and height of the generated image.

[78] Server startup file option `egr.eai.gf.color_space=LinearRGB` or option `renderMode=PBR` of operation `getGeneratedImage`.

If the `scale` option is used, the `zoom` option must not be specified. Furthermore, both the longitude and latitude must be multiples of 90.0 degree. The default value for the longitude is zero degree, and the default value for the latitude depends on the longitude. If the longitude is zero, the default latitude is 90.0 degree. Otherwise, it is zero degree.

In any case, if the latitude is ±90.0 degree, then the longitude must be zero.

`scale=`*`float:1.0…10,000.0`*

The `scale` option may be used to generate a true to scale image. The value of the `scale` option species the number of pixels per meter.

Use of the `scale` option is mutually exclusive with the `width`, `height` and `zoom` options and restricts the set of available values for the `projectionMode`, `alpha` and `beta` options.

Image generation fails if the chosen scale results in an image width and/or height greater than 2048 pixels (including the margin).

`margin=`*`int:0…`***5***`…512`*

The `margin` option may be used together with the `scale` option to add a margin of the specified number of pixels around the image. Thus the width and height of the generated image is computed as the minimum required width and height to fit the image representation of the article plus twice the margin.

If the `scale` option is not used, the `margin` option must not be used either.

`enableShaders=`*`boolean:`***true***

This option, if set to `true`, enables the use of GL shading language.

If the `renderMode` option is set to a value other than `Textures`, shaders are always disabled, regardless of the value of the `enableShaders` option.

Starting with EAIWS 3.1, the default value for this option is `true` as the use of shaders does not only result in images of better quality, but also reduces rendering time.

`mcp.obx=`*`boolean:`***true***

Format: MCP

If true, the MCP container contains the file articles.obx with commercial information about the objects.

`mcp.comIDs=`*`boolean:`***false***

Format: MCP

If `mcp.obx` is false, but this option is `true`, the objects.xml file contained in the MCP container uses the `<comID>` element to store the commercial ID (basket item ID) of each object. This may violate the MCP specification, but may be used to optimize cases where the client needs to associate parts of the image with basket positions, but does not need the articles.obx file (because it obtains the information contained in this file in some other way).

If `mcp.obx` is true, the value of this option is ignored.

A client that receives an MCP URL from the `getGeneratedImage` operation can now always access all member files of an MCP file by stripping of the '.eimg' suffix from the MCP URL and appending a slash followed by the member file name.

`ars=`***`boolean:false`***

Enables the use of Article-specific Rendering Setups (ARS)

`ars.camera=`***`boolean:true`***

Enable/disable use of the camera setup specified in the ARS database.

`ars.lighting=`***`boolean:true`***

Enable/disable use of the lighting specified in the ARS database.

`ars.view=<view-id> (default: empty)`

Specify alternative view ID that can be used to select a non-default ARS setup (if provided by the database).

`osao=`***`boolean:false`***

Obsolete! Option is still accepted, but otherwise ignored for image generation..

Enables Object Space Ambient Occlusion (has no effect on OSAO-based shadows on the shadow plane, as described below)

`osao.radius=`***`float:0.002…10.0…`***

Obsolete! Option is still accepted, but otherwise ignored for image generation.

`osao.intensity=`***`float:0.0…0.3…1.0`***

Obsolete! Option is still accepted, but otherwise ignored for image generation..

`osao.mapSize=`***`int:32…1024…4096`***

Controls the total size (width and height in texels) of the lightmap computed for all surfaces of the article.

`osao.smoothness=`***`float:0.001…0.02…1.0`***

`osao.samples=`***`int:1…20…256`***

`viewId`***`=<view-id>`***

If the option is present then its value must be either the NIL-UUID (possibly represented by an empty string) or the ID of a basket view used by the current session.

If the option is not present, or if its value is the NIL-UUID, then the item ID is interpreted as the ID of a basket item. Otherwise the item ID is interpreted as the ID of a view item of the specified view, and the image is generated for the basket item referenced by this view item, or, in case of merged article view items, for one basket item randomly chosen from the set of referenced basket items. An error occurs if the referenced basket item is not an article item.

Furthermore, if the `viewId` option specifies a valid view ID, the `tag` option is not present, has an empty value, or has value `default`, and the `hideSubArticles` option is not present, the option `hideSubArticles=false` is added if the view uses merge mode `Compact` or the type of the referenced article is `PartialPlanning`[79].

The `getGeneratedImage` operation returns an empty string if the product data does not support automatic generation of article images, if the Online Configurator has been configured not to generate article images, if the necessary license feature could not be leased during start-up of the Online Configurator, or if the process of generating the article image failed for whatever reason. Otherwise, the operation returns an URL referencing the generated image. The URL remains valid until the session is explicitly closed by the `closeSession` operation of the session service, or is automatically closed after a configurable time of inactivity (§4.1.6).

If a true to scale image has been generated (by use of the `scale` option), the returned URL contains a query part specifying the width and height of the generated image as well as the local bounding box of the article's 3D geometry. The query part may or may not be stripped from the URL before the URL is used to fetch the image.

The query part is separated from the URL path by a question mark (`?`) and consists of a sequence of key-value pairs separated by ampersands (`&`). Each key-value pair consists of a key, an equals sign (`=`), and integral or floating point number. The keys `width` and `height` are used to return the dimension of the generated image in pixels. Their values are positive integers. The keys `xmin`, `ymin`, `zmin`, `xmax`, `ymax` and `zmax` are used to return the dimension of the local bounding box of the article's 3D geometry within a right-handed coordinate system with a horizontal xy-plane. Their values are floating point numbers in computerized scientific notation or decimal format, depending on the precision and the value in meter after rounding.

The `getGeneratedImage` operation may be called multiple times for the same article item with different options. For each set of options a new image will be generated. Successive calls of `getGeneratedImage` for

79 In other words, if operation `getGeneratedImage` is used to generate the default article image through a view item then, unless explicitly specified otherwise, the value of option `hideSubArticles` is chosen as appropriate for the image to be displayed in the order list.

the same article item with an identical set of options will not regenerate the image unless there was an intervening call of the `setPropertyValue` operation.

If operation `getGeneratedImage` is invoked with the NIL-UUID (or an empty string) as item ID, and the `viewId` option is not present or specifies the NIL-UUID (or an empty string), the specified options are stored for the current session and later used whenever an article image needs to be generated on demand.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify an article item.
- One of the specified options has an invalid value[80] [81].
- If used to set default image generation options and the specified format is neither JPEG nor PNG.

Change behaviour of operations getGeneratedImage and getExportedGeometry of basket service:

If the operation is invoked to get an image or geometry for a set-article item, and the geometry compositor fails to obtain the geometry for at least one set-article part, the operation terminates with a BasketServiceFault instead of returning an empty string (as was the case before). The field `partCompositionFailures` of the fault contains information about *all* parts that should have a geometry, but failed to provide one.

Mechanism that allows HTTP client to control Content-Disposition response header for GET requests referencing files in the session cache.

Client control of the header Content-Disposition for files in the session cache.

5.6.3.41 `getArticleRenderingSetup`

Synopsis:

```
ARSRenderingSetup getArticleRenderingSetup(SessionId sessionId,
                                           string catalogId,
                                           string manuId,
                                           string seriesId,
                                           string baseArtNr,
                                           string *ofmlVarCode,
                                           string[] viewIds)
        throws BasketServiceFault;
```

The operation `getArticleRenderingSetup` is used to access data from ARS (article-specific rendering setup) database[82].

The rendering setup consists of information about the camera setup (position and orientation) and information about lighting (ambient light and light sources as known from OpenGL).

80 The current implementation does not enforce the allowed value ranges specified for the `alpha` and `beta` option values. They should nevertheless be honored by a client of the basket service as this is not guaranteed for future versions of the Online Configurator.
81 If not explicitly mentioned otherwise, the value of an option must always be valid according to the specification, even if the option is ignored, or the operation will fail.
82 For further information see ARS specification.

```
struct ARSRenderingSetup {
    ARSCamera   *camera;                                // element
    ARSLighting *lighting;                              // element
}
```

Reference point, viewpoint and positions of light sources are defined by the origin of some coordinate system and the point's or position's coordinates within that coordinate system. The coordinate system can be cartesian or spherical.

```
enum ARSCoordinateSystem {
    Cartesian,
    Spherical
}
```

The origin can be defined as either the origin of the local coordinate system, the center of the bounding box plus some offset, the reference point or the view point.

```
enum ARSOriginType {
    LocalCoordinateSystem,
    BoundingBox,
    ReferencePoint,
    ViewPoint
}
```

The offset is computed as the size of the bounding box in each direction multiplied by x/y/z scaling factors defined as part of the origin.

```
struct ARSOrigin {
    ARSOriginType   type;                               // attribute
    float           x;                                  // attribute
    float           y;                                  // attribute
    float           z;                                  // attribute
}
```

As mentioned above, coordinates are specified as either cartesian or spherical. The member `coordinateSystem` of `ARSCoordinates` defines the type of coordinates used. The actual data type corresponds to the specified type of coordinate system.

```
abstract struct ARSCoordinates {
    ARSCoordinateSystem coordinateSystem;               // attribute
}
```

If cartesian coordinates are used, the axes of the coordinate system have the same direction as the corresponding axes of the object's local coordinate system.

```
struct ARSCartesianCoordinates : ARSCoordinates {
    float   x;                                          // attribute
    float   y;                                          // attribute
    float   z;                                          // attribute
}
```

If spherical coordinates are used, the zenith direction corresponds to the direction of the y-axis of the object's local coordinate system, and the azimuth is measured counter-clockwise from the positive z-axis (0°) to the positive x-axis (90°).[83]

Azimuth and elevation are specified in degree. EAIWS does not impose any limits on the values of radius, azimuth and elevation.

[83] Conversion of spherical into cartesian coordinates is done as follows (radius r, elevation θ, azimuth φ): $y = r \sin\theta$; $x = r \cos\theta \sin\varphi$; $z = r \cos\theta \cos\varphi$

```
struct ARSSphericalCoordinates : ARSCoordinates {
    float   azimuth;                                        // attribute
    float   elevation;                                      // attribute
    float   radius;                                         // attribute
}
```

The origin type of the reference point is either `LocalCoordinateSystem` or `BoundingBox`. The coordinates of the reference point are always specified as cartesian coordinates.

```
struct ARSReferencePoint {
    ARSOrigin                origin;                        // element
    ARSCartesianCoordinates coordinates;                    // element
}
```

The origin of the viewpoint is either `LocalCoordinateSystem`, `BoundingBox` or `ReferencePoint`.

```
struct ARSViewpoint {
    ARSOrigin       origin;                                 // element
    ARSCoordinates  coordinates;                            // element
}
```

In addition to reference and viewpoint, the camera setup defines the projection mode and zoom.

In perspective mode, the zoom is passed unmodified to `gluPerspective`, defining the field of view angle, in degrees. In orthographic mode, the `top` and `bottom` values passed to `glOrtho` are computed as ±*tan(zoom · π / 180.0)*, thus defining the coordinates for the top and bottom horizontal clipping planes.

```
enum ARSProjectionMode {
    Perspective,
    Orthographic
}

struct ARSCamera {
    ARSReferencePoint   referencePoint;                     // element
    ARSViewpoint        viewpoint;                          // element
    ARSProjectionMode   projectionMode;                     // attribute
    float               zoom;                               // attribute
}
```

Lighting information consists of the ambient light and a possibly empty list of light sources.

```
struct ARSLighting {
    Color               ambientLight;                       // element
    ARSLightSource[]    lightSources;                       // element
}
```

Light sources are defined the same way as in OpenGL.

```
enum ARSLightSourceType {
    Directional,
    Positional
}
```

```
struct ARSLightSource {
    ARSOrigin           origin;                          // element
    ARSCoordinates      coordinates;                     // element
    Color               ambientLight;                    // element
    Color               diffuseLight;                    // element
    Color               specularLight;                   // element
    float               constantAttenuation;             // attribute
    float               linearAttenuation;               // attribute
    float               quadraticAttenuation;            // attribute
    ARSLightSourceType  type;                            // attribute
}
```

Lights are defined by their red, green and blue components, with values being between zero and one, both inclusive.

```
struct Color {
    float   red;                                         // attribute
    float   green;                                       // attribute
    fluat   blue;                                        // attribute
}
```

The catalog ID, manufacturer ID and series ID passed to `getArticleRenderingSetup` must be specified and valid.

The base article number must be specified and must not be empty.

The OFML variant code must be a valid OFML variant code. It should be equal to the OFML variant code of the article instance whose rendering setup is to be determined. If it is missing, an empty OFML variant code is assumed instead.

The list of view IDs may be empty. If so, a list containing only the default view ID (an empty string) is used in-stead.

For each view ID, the operation looks for a rendering setup matching the specified parameters. The first one found is returned. If no rendering setup is found, nothing is returned.

5.6.3.42 `getImages`

Synopsis:

```
ImageInfo[] getImages(SessionId          sessionId,
                      BItemId            itemId,
                      GetImagesOptions   options)
        throws BasketServiceFault;
```

The `getImages` operation is used to fetch all images stored for a particular basket item.

The `itemId` parameter must identify an item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter. The operation may be invoked for any type of item, although, with the current implementation (version 3.1 of the Online Configurator), the returned list of `ImageInfo` instances will be empty except for OFML and OCD article items.

The operation returns the list of image associated with the specified basket item. Each returned instance of `ImageInfo` has a distinct tag, multiple instances may reference the same image.

As a special case, article items may return an `ImageInfo` with an empty tag. If so, the referenced image is said to be the *PBK image*, as this functionality has been implemented for compatibility with pCon.basket and other applications based on pCon.basket technology. See §Fehler: Verweis nicht gefunden for more informa-tion.

The item selection options are used as usual. To preserve backwards compatibility, however, the operation continues to take a single item ID instead of a list of item IDs as argument.

Items of type `Aggregate` may have two default article images, one with sub-articles hidden and one with sub-articles shown (although not necessarily visible). The default article image returned by operation `getImages` depends on the item type and, if a basket view is used, the merge mode of the view:

- The default article image of items of type `PartialPlanning` will always show sub-articles.
- Default article images of items of all other types do not show sub-articles unless the item is a view item and the view uses merge mode `Compact`.

The `getImages` operation never attempts to generate an image. It returns all images generated for a particular item, not only the default article image. However, if the item is an article view item and the view uses merge mode `Articles`, or it is a view item representing a sub-article and the view uses merge mode `SubArticles` or `Compact`, these additional images are not returned.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify a basket item.

5.6.3.43 `getExportedGeometry`

Synopsis:

```
    URL getExportedGeometry(SessionId sessionId,
                            BItemId   itemId,
                            string[]  options
                            )
            throws BasketServiceFault;
```

The `getExportedGeometry` operation is used to export the 3D geometry of the article identified by the `itemId` parameter.

The `itemId` parameter must identify an article item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

The `options` parameter is a sequence consisting of zero or more option strings, where each option string consists of a key and value, separated by an equal sign (`=`) (e.g. `"format=3DS"`). The following options are available (default values are displayed **bold**):

`format=`*enum*:***3DS***|*DWG*|*GFX*|*SKP*|*FBX*|*OBJ*|*DAE*|*IGXC*|*GFJ*|*GLTF*|*USD*|*PEC*

The `format` option specifies the format that will be used by the geometry export. Currently supported formats are `3DS` (the default), `DWG`, `GFX` (for EOX-GFX[84]), FBX and SKP[85].

`format=GLTF`

In addition to the standard options hideSubArticles, layerControl, centerXZ, compression, and scale the following options are supported:

ascii=<boolean> - If false (default value), the export produces a single GLB file. Otherwise the export produces a ZIP file containing a file named geometry.gltf and zero or more additional files referenced with relative paths by the GLTF file.

To get an URL for individual files, the client may replace the .zip suffix of the returned URL with a slash and the name/path of the file as used in the ZIP file.

Option compression=true may not be used with ascii=false. compression=false is ignored if used with ascii=false.

texTrans=<boolean> - If set to true, exports texture coordinates with transformation. This

84 EOX-GFX is a format that does not include the actual geometries, but contains a list of geometries and materials as well as instructions on how to compose them to the final geometry. Furthermore, it may reference (and be delivered together with) an OBX file that contains commercial information for the exported article.

85 Only available on Microsoft Windows.

way at most one set of UV coordinates is written, which most viewers are able to cope with better.

`This export requires license feature egr.eai.server.export.gltf.`

`format=USD`

In addition to the standard options hideSubArticles, layerControl, centerXZ, compression, and scale the following options are supported:

ascii=<boolean> - If false (default value), the export produces a single USDZ file. Otherwise the export produces a ZIP file containing a file named geometry.usda and zero or more additional files referenced with relative paths by the USDA file.

To get an URL for individual files, the client may replace the .zip suffix of the returned URL with a slash and the name/path of the file as used in the ZIP file.

Option compression=true may not be used with ascii=false. compression=false is ignored if used with ascii=false.

texTrans=<boolean> - If set to true, exports texture coordinates with transformation. Files produced with texTrans=true can currently not be processed by ArKit (Apple).

This export requires license feature egr.eai.server.export.usd.

`centerXZ=boolean:false`

Being either true or false, to most geometry exports (all except CCL and EGMS). If set to true (default value is false), the geometry is exported so its bounding box is centred around the Y (vertical) axis. Y coordinates are not affected.

`dwgSymbol=boolean:false`

Formats: GFJ

Default: `false`

Being either true or false, to GFJ geometry export. If true (default value is `false`), the GFJ file references a single 2D DWG file containing the 2D geometry of the exported object.

`hideSubArticles=`*`boolean`*`:`***`false`***

Formats: all

Controls whether or not sub-articles of composite articles are hidden during the export (and thus are not exported together with the composite article).

If the operation is invoked for a sub-article, then this sub-article is not hidden, regardless of the value of this option. If, however, the sub-article happens to be a composite article, then the option affects the hiding of sub-articles of this nested composite article.

`serverBase=`*`url`*

Formats: GFX, IGXC,GFJ

The `serverBase` option, if specified, should be a valid (HTTP) URL. It is used by exports that produce files which contain URLs to other resources, like images or geometries. The relative path of the image or geometry resource (like `OFML/data/…`) is resolved[86] against the server base. The resulting URL is then written to the exported file.

The default value for `serverBase` is `http://`*`hostname`*`:`*`port`*`/`, where *hostname* and *port* are derived from the header of the HTTP request carrying the body of the `getExportedGeometry` operation.

86 Note that the parsing of the serverBase option fails if the URI is relative or opaque, or has a query or fragment component. Furthermore, it automatically appends a slash to the path component if the path component does not already end with a slash.

`geometryExtensions=`*`extension-list`*`:`***`.geo,.3ds,.dxf,.dwg,.obj,.glb,.gltf`***

`imageExtensions=`*`extension-list`*`:`***`.rgb,.tga,.jpg,.bmp`***

Formats: GFX, IGXC, GFJ

These options are used to specify the set of geometry and image file formats the client is willing to deal with as well as the search order. The values of these options must be formatted the same way as the value of the `egr.eai.http.ofml_extensions` start-up file entry (§4.1.14), except that there must be no leading PLUS SIGN. If one of these options contains an extension that is not part of the default set of extensions served by the OFML HTTP server, then the afore mentioned start-up file entry should be used to configure the OFML HTTP server to serve files with this extension, or HTTP requests for these files will result in a 404 (Not Found) HTTP error.

In case of the IGXC and GFJ export, the default value for option `geometryExtensions` is *.obj.* The default list of image extensions for GFJ export consists of *.jpg* and *.png*, in that order.

If the IGXC/GFJ export needs a geometry, it first attempts to find a geometry file within the OFML data, using the specified geometry extension list to select and prioritize geometry formats. If no such file can be found, the export generates the required geometry. If the first extension from the extension list is .glb or .gltf, the export will generate GLB or GLTF geometries, respectively. Otherwise, it will generate OBJ geometries.

`hierarchyMode=`*`enum`*`:`*`Hierarchy`*`|`***`MatrixStack`***`|`*`MaterialStack`*`|`*`Flat`*`|`*`Path`*`|`*`Children`*

Formats: GFX, FBX (only supports *`Hierarchy`*`|`***`Flat`***)

`Flat`

No hierarchy, i.e. all (geometric) primitives are on the same level.

`Hierarchy`

Writes the OFML object hierarchy, but materials and transforms are still assigned to primitives (leave nodes) only.

`MatrixStack`

Writes the OFML object hierarchy, transforms are assigned to top objects (inner nodes), if possible. Materials are assigned to primitives (leave nodes).

`MaterialStack`

Like `MatrixStack`, but materials are assigned to top objects too.

Formats: IGXC

`Path (default)`

If set to `Path`, the geometry objects are exported as an array of JSON objects. Each object has a key named "`Path`" and a value of type string. The string can be used to rebuild the object hierarchy.

`Children`

If set to `Children`, the geometry objects are exported as a tree of JSON objects. Each inner node object of the tree has a key named "`Children`", whose value is an array of child objects.

`obx.enabled=`*`boolean`*`:`***`false`***

Formats: GFX

If set to true, the export produces a ZIP file that contains the GFX file as `geometry.gfx`. `<eoxObject/>` elements that represent the root of the hierarchy of geometries for an article have the attribute `comInfo`. The value of the `comInfo` attribute is a relative URI that is to be resolved against the URI for `geometry.gfx`. The scheme specific part of the resulting URI identifies an OBX file, and the fragment (remaining characters after the number sign (`#`)) the article item within the OBX file through its item identifier.

At present, the OBX file is always named `1.obx`, but future versions of the Online Configurator may use another name, or even multiple files. Therefore the name of the OBX file must always be taken from the value of the `comInfo` attribute.

`obx.encoding=`*`string`*`:`***`UTF-8`***

Formats: GFX

This option allows the specification of the character set to be used when the OBX file is written.

`scale=`*`float`*`:`***`1.0`***

Formats: 3DS, DWG, SKP, FBX, OBJ, DAE

Scale the output geometry by the given factor.

`duplicateFaces=`*`boolean`*`:`***`false`***

Formats: 3DS, FBX, DAE

For each face (triangle), write another face with the same vertices but reverse orientation. Useful, if you want to feed the output to applications, which do not handle oriented faces correctly.

`empty=`*`boolean`*`:`***`true`***

Formats: 3DS

Generate one face for empty meshes. Useful, if output is to be fed to applications which can not handle empty meshes (this includes 3DStudio(Max) itself, at least for some versions).

`use3DProxies=`*`boolean`*`:`***`true`***

Formats: DWG

If `true`, DWGs from the data packages ("Proxy DWGs") are directly included in the output. If `false`, imported DWGs are tessellated to polygon meshes.

This option should be set to `false` if output is to be fed to applications which do not handle ACIS primitives. It should be set to `true` if output is to be fed to AutoCAD, since AutoCAD R17 and above have problems rendering polygon meshes.

`textures=`*`boolean`*

Formats: 3DS, DWG, SKP, FBX, OBJ, DAE

Default:
3DS, SKP, FBX, OBJ, DAE: `false`
DWG: `true`
If `materials=false` or `textureToColor=true` or `acadColorIndex=true` the default value is always `false`.

This option controls whether the export just exports the geometry (value `false`), or exports both the geometry and texture images (value `true`).

For 3DS, OBJ and DAE if the value of this option is false, the returned URL references a single geometry file. Otherwise, the returned URL references a ZIP file. One entry of the ZIP file is named `geometry.<format>`. If the geometry references texture images, these texture images are also packed into the ZIP file, using the same names as done by the geometry.

If the returned URL references a ZIP file, but the client is unable or unwilling to deal with the ZIP file[87], the client is still able to fetch the individual geometry and image files. To do so, it must strip the `.zip` suffix from the returned URL and append `/geometry.<format>`. This URL can then be used to fetch the geometry. The client can then analyse the geometry to get the texture image file names, and, for each image file name, replace `geometry.<format>` with the name of the texture image to form an URL that can be used to fetch the texture image.

87 The format of the ZIP file is kept quite simple. Particularly, it uses neither compression nor encryption. There are no extra fields, comments, or data descriptors, and the crc-32, compressed-size, and uncompressed-size fields of the local file headers are all valid. Thus it is possible to process the ZIP file in a single pass, as there is no need to read the central directory before the entries can be reliably extracted.

If option `textureToColor` is `true` textures are not exported, no matter what the value of the `textures` option.

`materials=`*`boolean`*

Formats: DWG, SKP, FBX, OBJ, DAE, IGXC

Default:
DWG, SKP, FBX, DAE: `true`
OBJ, IGXC: `false`

If `textures=true` or `textureToColor=true and not acadColorIndex=true` the default value is always `true.`

If `acadColorIndex=true and not textures=true` the default value is always `false.`If `true`, export materials, if `false` export just geometries.

For OBJ a value of `true` for this option requires the OBJ export to return a ZIP file.

`ofmlMaterials=`*`boolean`*`:`***`false`***

Formats: IGXC

Write materials in OMATS format.

`textureToColor=`*`boolean`*`:`***`false`***

Formats: DWG, SKP, FBX, DAE

If this option is `true`, average texture colors instead of diffuse colors are exported. Furthermore, textures are not exported, no matter what the value of the `textures` option.

`acadColorIndex=`*`boolean`*`:`***`false`***

Formats: DWG

If this option is `true`, materials will get AutoCAD colors.

`staticMaterials=`*`boolean`*`:`***`false`***

Formats: IGXC

Enable export of static materials only.

`shortMaterialNames=`*`boolean`*`:`***`false`***

Formats: FBX

`layers=`*`boolean`*`:`***`false`***

Formats: IGXC

Write layer information

`no2D=`*`boolean`*

Formats: DWG, SKP, FBX, DAE, GFJ

Default:
SKP, FBX, DAE, GFJ: `true`
DWG, PEC: `false`

If `true`, do not export 2D geometries, i.e. export 3D only. In case of PEC this option is always `false`, so PEC always contains a 2D geometry. It is not possible to turn of the 2D symbol. This is also the case for the dwg file contained in the PEC export (Option `dwg.enabled=true`).

`no3D=`*`boolean`*`:`***`false`***

Formats: DWG

If `true`, do not export 3D geometries, i.e. export 2D only.

`layerControl`

The argument of this option must be a string that adheres to the following grammar:

```
layer-control = [ global ] { specific }
global = "\*" | "!"
specific = onoff layer-spec { suffix }
onoff = "\+" | "-"
suffix = "r" | "i"
layer-spec = "'([^']|'')*'" | "\"([^\"]|\"\")*\""
```

(Character sequences enclosed in double quotes represent terminal symbols specified as POSIX Extended Regular Expressions, with the addition that \" represents an ordinary double quote character that does not terminate the character sequence.)

The layer control string is processed left to right. If `global` is present, all layers are initially switched on ('*') or off ('!'). 'layer-spec' is a string whose meaning depends on the possibly empty sequence of suffixes. If the 'r' suffix is present, it is a POSIX Extended Regular Expression, selecting all layers whose complete name matches the regular expression. Otherwise it is an ordinary string that must match a complete layer name. If the 'i' suffix is present, matching is done in a case-insensitive way. It is not an error if suffixes appear more than once. Selected layers are switched on if `onoff` is '+', and off if `onoff` is '-'.

After the export, layers are switched back to their original state.

`layerRules`

This option is used to summarize the layers inside the DWGs. The rules which apply for summarizing can be defined in an xml file, which is saved in the directory `./etc/elayers/`. By default EAIWS has one layer rule file available under `./etc/elayers/color.xml`.The value of this option must be a relative path name to the `elayer` directory (`./etc/elayers`).

Formats: DWG

`r17=`*`boolean`*`:`***`true`***

Formats: DWG

If `true`, export in AutoCAD R17 format, if `false`, use AutoCAD R16.

`dxf=`*`boolean`*`:`***`false`***

Formats: DWG

If `true`, export in DXF instead of DWG format.

`resolution=`*`enum`*`:`***`Default`***`|`*`Low`*

Formats: SKP, FBX, DAE

With value `Low`, the exports export low-resolution geometries, if available in the OFML data.

Low-resolution geometries are stored in geometry files starting with an underscore. Thus, if the name of the standard geometry file is 'foo.geo', the name of the low-resolution geometry file is '_foo.geo'. If no low-resolution geometry file is found, the standard geometry is used instead.

This mechanism works for all supported geometry formats as long as the geometry is accessed through a qualified name (i.e. `::manu::prog::geo_name`).

`edges=`*`boolean`*`:`***`false`***

Formats: DAE

Activate edges for DAE export.

`annotation=`*`boolean`*`:`***`false`***

Formats: DWG

If `true`, export additional geometries to support snapping in pCon.xcad and pCon.planner 6.

`compression=`*`boolean`*`:`***`false`***

Formats: all exports that return a ZIP file (GFX, 3DS, OBJ, DAE)

This option is recognized by all geometry exports that may return a ZIP file. These exports return a ZIP file if some information produced by the export cannot be stored as part of the geometry file (textures in case of 3DS, textures and materials in case of OBJ, OBX in case of GFX).

If these exports return no ZIP file, the `compression` option has no effect. Otherwise, if the value of this option is `false`, or if the name of the ZIP file entry ends with `.jpg`, `.jpeg` or `.png`, the ZIP file entry is stored without compression. Otherwise, the ZIP file entry is compressed.

Currently, compression level 4 is used as a test with an OBJ file with one (rather small) texture image showed this as the greatest level with a marginal benefit (relative reduction in file size divided by the relative increase in computation time) greater than one.

If the OBJ export returns the URL of a ZIP file, looking like `http://<host>/<path>/<filename>.zip`, the files contained in the ZIP file can also be accessed using an URL where <filename>.zip is replaced by <filename>/<element-name>, with <element-name> being `geometry.obj`, `geometry.mtl`, or `<texture-name>.jpg`.

`compactJSON=`*`boolean`*`:`***`false`***

Formats: IGXC

Omit indentations and newlines.

`jsonArrays=`*`boolean`*`:`***`false`***

Formats: IGXC

Write colors and vectors as JSON arrays

`crc128=`*`boolean`*`:`***`false`***

Formats: IGXC

Use CRC128 checksums for geometries. This also enables the use of a global geometry cache for geometries generated by the export.

`basketIds=`*`boolean`*`:`***`false`***

Formats: IGXC

If true, geometry objects that correspond to a basket article item have a key named "`BasketId`". The value is a 36 character long string representing the item ID of the basket article item formatted as an UUID consisting of only hex digits and minus signs.

Right now this option is ignored if the exported item is not a basket article (i.e. its BasketItemType is neither `Article`, `Aggregate` or `PartialPlanning).`

`skpVersion=enum:SU3|SU4|SU5|SU6|SU7|SU8|SU2013|SU2014|SU2015|SU2016`

Controls the version of the exported SKP file. If no version is specified, or the specified version is not understood by the FAPI converter, the most recent version supported by the SKP SDK will be used.

`revit=boolean:`***`false`***

Formats: GFX

If this option is true, the GFX export will pack the GFX file into a ZIP file (name `geometry.gfx`). Furthermore, if the export selects a DWG file as a part geometry, the DWG will be scaled as necessary and added to the ZIP file. The geometry references within the GFX file will contain relative URIs that must be resolved relative to the root of the ZIP file.

The option `revit`=true should probably be used together with option `geometryExtensions=.dwg`

A client may also modify the ZIP file URL as follows to download the GFX and DWG files:

1. Strip the suffx (.zip) from the URL.
2. Append a slash ('/') character.

3. Append the (path) name of the file as used in the ZIP file.

Thus, if the ZIP file URL is

```
http://host/.../foo.zip
```

the URL of the GFX file would be

```
http://host/.../foo/geometry.gfx
```

The same can be done for the DWG files once their (path) names have been extracted from geometry.gfx.

`edwg=`*`boolean`*`:`***`false`***

Formats: DWG

Right now, the EDWG export is supported for OFML articles (simple and composite) only (i.e. not for OCD articles or sets of OFML articles).

`maxImageSize=`*`integer`*

Format: GFJ, IGXC, PEC

It controls the maximum size (width/height) of texture images referenced by the GFJ/IGXC/PEC file. The option’s value must be a non-negative decimal, hexadecimal or octal integer less than 2^{31}. A value of zero disables the option. Values between zero and N are replaced with N (where N is some arbitrarily chosen small positive integer, currently 16). The default value is zero.

`scaleFilter=`*`enum: Legacy`*`|`***`Linear`***`|`*`BiCubic`*`|`*`Lanczos2`*

Format: PEC

Determines the scale filter to be used if a texture image needs to be rescaled.

`omitPriceData=`*`boolean`*`:`***`false`***

Format: PEC

Controls whether price data should be omitted from document.obx and from OBX snippets embedded in document.dwg (if enabled).

`preview.enabled=`*`boolean`*`:`***`false`***

Format: PEC

Controls whether a preview image should be generated and embedded within the PEC file. The following options are always accepted but ignored unless `preview.enabled` is `true`:

- `preview.width`
- `preview.height`
- `preview.resample`
- `preview.quality`
- `preview.ambient`
- `preview.background`
- `preview.renderMode`
- `preview.enableShaders`
- `preview.fadeBorder`
- `preview.shadowPlane`
- `preview.shadowPlane.border`
- `preview.shadowPlane.color`
- `preview.shadowPlane.smoothness`
- `preview.shadowPlane.mirror`
- `preview.shadowPlane.colorMode`
- `preview.shadowPlane.mapSize`
- `preview.shadowPlane.filter`
- `preview.shadowPlane.maxHeight`
- `preview.shadowPlane.radius`
- `preview.shadowPlane.intensity`
- `preview.shadowPlane.samples`

- `preview.zoom`
- `preview.alpha`
- `preview.beta`
- `preview.projectionMode`

If `preview.enabled` is `true`, these options validated after all other options have been processed. Their allowed values are equal to those of the corresponding image export options with the `preview.` prefix removed (see basket service operation `getGeneratedImage`).

`camera=<camera-specification>`

Format: PEC

Determines the values stored in the `<camera/>` element of `pec.xml`.

The camera specification consists of four (perspective) or five (orthographic) fields separated by colon (`:`). The values of individual fields must be as follows:

1. `perspective` or `orthographic`
2. a three-element array representing the viewpoint
3. a three-element array representing the viewing direction
4. a three-element array representing the up-vector
5. a single number representing half the visible size in y-direction

The vectors representing viewing direction and up-vector don't need to be normalized, but must be non-zero.

Arrays start with a left square bracket (`[`), end with a right square bracket (`]`), and contain numbers separated by comma (`,`).

All numbers must adhere to the usual syntax for floating point numbers. Special numbers like `NaN` and `Inf` are not allowed.

Whitespace at the start and end of the camera specification, around field separators, and around numbers is ignored.

If no `camera` option is present, but `preview.enabled` is `true`, the camera specification is derived from the camera settings used to generate the preview image. Otherwise, no `<camera/>` element is written to `pec.xml`.

`dwg.enabled=`*`boolean`*

Format: PEC

Controls whether the generated PEC file contains an EDWG file (`document.dwg`)

`compression=`*`integer`*

Format: PEC

Controls the compression level used for non-texture entries of the PEC file. The specified compression level must be between 0 and 9, both inclusive, with 0 disabling compression and 9 resulting in maximal compression. The default value is 6.

The `getExportedGeometry` operation returns an empty string if the product data does not support export of geometries, if the Online Configurator has been configured not to export geometries of the selected format, if the license feature necessary for the selected format could not be leased during start-up of the Online Configurator, or if the process of exporting the geometry failed for whatever reason. Otherwise, the operation returns an URL referencing the exported geometry, or a ZIP file containing the geometry. The URL remains valid until the session is explicitly closed by the `closeSession` operation of the session service, or is automatically closed after a configurable time of inactivity (§4.1.6).

The `getExportedGeometry` operation may be called multiple times for the same article item with different options. For each set of options a new geometry will be exported. If `getExportedGeometry` is called successively for the same article item with an identical set of options then the geometry will not be re-exported unless there was an intervening call of the `setPropertyValue` operation.

A `BasketServiceFault` is returned if any of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify an article item.
- One of the specified options has an invalid value.
- If textures are enabled and materials are disabled.
- If textures are enabled and texture-to-color are enabled.
- If texture-to-color is enabled and materials are disabled unless acad-color-index is enabled.
- If acad-color-index is enabled and either textures or materials is enabled.

Change behaviour of operations getGeneratedImage and getExportedGeometry of basket service:

If the operation is invoked to get an image or geometry for a set-article item, and the geometry compositor fails to obtain the geometry for at least one set-article part, the operation terminates with a BasketServiceFault instead of returning an empty string (as was the case before). The field `partCompositionFailures` of the fault contains information about *all* parts that should have a geometry, but failed to provide one.

Mechanism that allows HTTP client to control Content-Disposition response header for GET requests referencing files in the session cache.

Client control of the header Content-Disposition for files in the session cache.

5.6.3.44 `getConfigDependentMedia`

Synopsis:

```
ConfigDependentMediaInfo getConfigDependentMedia(SessionId sessionId,
                                                 BItemId   itemId,
                                                 string    mediaType,
                                                 string    filter,
                                                 string    preferences)
        throws BasketServiceFault;
```

5.6.3.45 `getAllConfigDependentMedia`

Synopsis:

```
ConfigDependentMediaInfo[] getAllConfigDependentMedia(SessionId sessionId,
                                                      BItemId   itemId)
        throws BasketServiceFault;
```

5.6.3.46 `setBasketAppData`

Synopsis:

```
void setBasketAppData(SessionId sessionId, string appKey, string[] data)
        throws BasketServiceFault;
```

The `setBasketAppData` operation is used to attach application-specific (or client-specific) data to the basket. This data can be accessed using the `getBasketAppData` (§5.6.3.47) operation and is part of the project file saved and loaded by the `saveSession` (§5.4.3.14) and `loadSession` (§5.4.3.15) operations. For more information, see §5.3.

A `BasketServiceFault` is returned if one of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- One of the elements of `data` contains an invalid location path, or the location path is not followed by an equals sign (=).

5.6.3.47 `getBasketAppData`

Synopsis:

```
string*[] getBasketAppData(SessionId sessionId,
                           string    appKey,
                           string[]  paths)
        throws BasketServiceFault;
```

The `getBasketAppData` operation is used to fetch application-specific (or client-specific) data attached to the basket which has either been read by the `loadSession` (§5.4.3.15) operation as part of the project file, or previously attached to the basket by the `setBasketAppData` (§5.6.3.46) operation. For more information, see §5.3.

A `BasketServicFault` is returned if one of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- One of the elements of `data` contains an invalid location path.

5.6.3.48 `setItemAppData`

Synopsis:

```
void setItemAppData(SessionId sessionId,
                    BItemId   itemId,
                    string    appKey,
                    string[]  data)
        throws BasketServiceFault;
```

The `setItemAppData` operation is used to attach application-specific (or client-specific) data to a basket item. This data can be accessed using the `getItemAppData` (§5.6.3.49) operation and is part of the project file saved and loaded by the `saveSession` (§5.4.3.14) and `loadSession` (§5.4.3.15) operations. For more information, see §5.3.

The `itemId` parameter must identify a basket item that is part of the basket item tree maintained by the basket service for the session identified by the `sessionId` parameter.

A `BasketServiceFault` is returned if one of the following conditions occurs:

- The `sessionId` parameter is not a valid string representation of an UUID or does not represent an open session.
- The `itemId` parameter does not represent an UUID.
- The `itemId` parameter does not identify a basket item.
- One of the elements of `data` contains an invalid location path, or the location path is not followed by an equals sign (=).

5.6.3.49 getItemAppData

Synopsis:

```
string*[] getItemAppData(SessionId sessionId,
                         BItemId   itemId,
                         string    appKey,
                         string[]  paths)
        throws BasketServiceFault;
```

The getItemAppData operation is used to fetch application-specific (or client-specific) data attached to a basket item which has either been read by the loadSession (§5.4.3.15) operation as part of the project file, or previously attached to the basket item by the setItemAppData (§5.6.3.48) operation. For more information, see §5.3.

The itemId parameter must identify a basket item that is part of the basket item tree maintained by the basket service for the session identified by the sessionId parameter.

A BasketServicFault is returned if one of the following conditions occurs:

- The sessionId parameter is not a valid string representation of an UUID or does not represent an open session.
- The itemId parameter does not represent an UUID.
- The itemId parameter does not identify a basket item.
- One of the elements of data contains an invalid location path.

5.6.3.50 getMultiItemAppData

Synopsis:

```
ItemAppData[] getMultiItemAppData(SessionId                sessionId,
                         BItemId                     itemIds,
                         string                      appKey,
                         string[]                    paths,
                        ItemSelectionOptions         *options)
        throws BasketServiceFault;
```

The operation getMultiItemAppData is used to fetch application specific data of multiple basket items using a single invocation of a web service operation.

The operation determines a set of selected items based on parameters itemIds and options. It then determines the item app data for each selected item the same way as done by operation getItemAppData (§5.6.3.49). Finally, it returns a sequence of ItemAppData elements (in unspecified order), where each element contains the item ID and the application specific data of this item.

5.6.3.51 copy

Synopsis:

```
string[] copy(SessionId        sessionId,
              BItemId[]        itemIds,
              OperationMode    opMode,
              string           uri,
              CopyOptions      options)
        throws BasketServiceFault;
```

The copy operation of the basket web service is used to serialize a sub-set of the basket items to an OBX stream. The sub-set is defined by the itemIds parameter and flags of the opMode parameter. The OBX stream is written to a file specified by the uri parameter.

The set of items serialized to the OBX stream (the set of selected items) is determined as follows, starting with the set of items specified by the `itemIds` parameter (the default value of operation mode flags is `false`):

1. The operation fails if the set of selected items contains a planning item (right now, this should not happen as there is no way to get a planning item into the EAIWS).
2. If the `subPositions` flag is `true` in the operation mode, the operation adds all descendants of selected items to the set of selected items unless they are already explicitly or implicitly selected at the beginning of this step (an item is implicitly selected if it is a sub-article item of a composite article whose main article is explicitly selected, or part of a set-article whose set-article item is explicitly selected).
3. For each selected set-article item, the operation ensures that all parts are selected too, adding them to the set of selected items if necessary.
4. If the `mainArticles` flag is `true` in the operation mode, the operation ensures that for each sub-article item contained in the set of selected items, the main article item is selected too, adding it to the set of selected items if necessary. If the flag is `false`, all sub-article items are removed from the set of selected items.
5. For each main article item contained in the set of selected items, the operation ensures that all sub-article items are contained in the set of selected items, adding them if necessary.

If the `uri` parameter is empty, the file is created within the session's working directory and the operation returns the URL of the file.

If the `uri` parameter is not empty, it must represent a file URL for a local file. The OBX stream is saved to this file unless the configuration of EAIWS does not allow access to the path specified by the URL, the file can not be created, or the file already exists and the `overwrite` option has not been specified.

The `copy` operation of the basket web service fails with a basket service fault if more than one item or composite article is to be copied and the application license feature `egr.eai.basket.copy_paste` is not enabled.

5.6.3.52 `paste`

Synopsis:

```
string[] paste(SessionId      sessionId,
               BItemId        fatherId,
               BItemId        beforeId,
               string         uri,
               PasteOptions   options)
        throws BasketServiceFault;
```

The `paste` operation of the basket web service is used to de-serialize the items contained in an OBX stream and insert them into the current basket at the position specified by the `fatherId` and `beforeId` parameters.

In general, OBX streams contain a forest of items. The root items are inserted as children of the parent item specified by `fatherId`, or the top folder if `fatherId` is the `NIL` ID.

If `beforeId` is the `NIL` ID, or if the item specified by `beforeId` is not a descendant of the parent item, they are inserted at the end the list of children of the parent item. Otherwise, they are inserted into the list of child items immediately before the child item which is either the item specified by `beforeId`, or one of its ancestors.

If the value of the `uri` parameter must be an URI returned by the `copy` operation, an URL returned by the `getUploadURL` operation of the session service, or a file URL for a local file that can be accessed by EAIWS (depends on the configuration of EAIWS).

The `paste` operation of the basket service returns the item IDs of all pasted items.

The operation operation `paste` fails with a basket service fault

- if the BSK/OBX stream contains a set-article and the application license feature `egr.eai.basket.set_articles` is not enabled.
- if the number of items in the OBX stream, not counting basket sub-article items, is greater than one and the license feature `egr.eai.server.multiple_positions` is not enabled.
- if, after paste, the number of items, not counting the top folder and basket sub-article items, is greater than one and the license feature `egr.eai.server.multiple_positions` is not enabled.

5.6.3.53 `pasteContainer`

Synopsis:

```
PasteContainerResult pasteContainer(SessionId              sessionId,
                                    BItemId                fatherId,
                                    BItemId                beforeId,
                                    string                 uri,
                                    PasteContainerOptions   *options)
        throws BasketServiceFault;

struct PasteContainerResult {
    BasketItem[] items;
    string[] originalItemIds;
    string[] itemGeometries;
    string sceneGeometry;
}
```

Arguments of this operation are similar to the arguments of the paste operation, except that the URI must reference a PEC file instead of an OBX file.[88]

The operation imports at least all required resources from the PEC file into the currently loaded project[89] and pastes the contents of the OBX stream contained in the PEC file at the position specified by father-ID and before-ID. The operation may fail if the PEC file is defective for some reason. It always fails if the PEC file does not contain an OBX stream.

Return value is an instance of complex type `PasteContainerResult`. Elements of this type contain the following elements:

- zero or more `<items/>` elements of complex type `BasketItem`
- zero or more `<itemGeometries/>` elements of type `xs:string`. Unless option `ignore-ComposableGeometries` has been set to true, the number of `<itemGeometries/>` elements is always equal to the number of `<items/>` elements, and the n-th element of both types correspond to each other.

  The value of an `<itemGeometries/>` element is either an URI or an empty string. URIs are returned for user article items with an attached geometry and for basket main article items with a geometry checksum that has a valid mapping to a resource stored in the PEC file. For all other items the value is an empty string.

  No `<itemGeometries/>` elements are returned if option `ignoreComposableGeometries` is `true`.
- Exactly one `<sceneGeometry/>` element of type `xs:string`. The value is an URI if the PEC file contains a scene geometry and option `ignoreComposableGeometries` is not `true`. Otherwise the value is an empty string.
- If option `returnOriginalItemIds` is set, `PasteContainerResult` will contain a list of `<originalItemIds/>` elements of type `xs:string`. The number of these elements is equal to

[88] Use of `returnOptionalItemIds=true` is broken in EAIWS 4.2 Beta 4.

[89] The current implementation blindly imports all resources, unless option `ignoreComposableGeometries` is set to true, in which case no resources are imported.

the number of `<items/>` elements, and the Nth elements of both lists correspond to each other.

Returned URIs have scheme `imp`. Operation `resolveURIs` of the session service may be used to convert them into URLs.

Note that saving the project will save only those resources that are directly or indirectly referenced by basket items at the time the project is saved. Thus, GFJs, geometries and textures imported from PEC files and exclusively used by basket main article items will not be saved.

URIs returned by operation `pasteContainer` are guaranteed to remain valid until a new project is loaded by operation `loadSession`, unless they have been resolved by operation resolveURIs, in which case they will remain valid until the session is closed.

5.6.3.54 `listPricingProcedures`

Synopsis:

```
PricingProcedureDescription[] listPricingProcedures(SessionId sessionId,
                                                    boolean   active)
        throws BasketServiceFault;
```

List pricing procedures (calculation schemes). If `active` is `false`, lists the pricing procedures configured for the Online Configurator. If `active` is `true`, lists the pricing procedures added to the current basket.

5.6.3.55 `getPricingProcedure`

Synopsis:

```
PricingProcedure getPricingProcedure(SessionId sessionId,
                                     boolean   active,
                                     string    name)
        throws BasketServiceFault;
```

Returns the description and lines of the pricing procedure with name `name`. With those information details about the pricing procedure definition can be obtained as set in the Online Configurator installation configuration.

5.6.3.56 `addPriceCalculation`

Synopsis:

```
void addPriceCalculation(SessionId sessionId,string ppName)
        throws BasketServiceFault;
```

Adds the price calculation with name `name` to the current basket. Multiple price calculations can be added.

The operation **`addPriceCalculation`** causes a reload of views that use a merge mode other than `None`.

5.6.3.57 `getPriceCalculationSheet`

Synopsis:

```
CalculationSheet* getPriceCalculationSheet(
                    SessionId          ´                sessionId,
                    string                              itemIds,
                    GetPriceCalculationSheetOptions     options,
                    string                              ppName)
        throws BasketServiceFault;
```

Returns calculation sheet for all items identified by `itemIds` and pricing procedure identified by `ppName`. Returns null if item does not have a calculation.

The document calculation sheet (header calculation) is returned if the `itemIds` parameter contains exactly one item ID and this item ID is the NULL-Id (empty string or all bits zero). Otherwise, the set of selected items is determined based on the specified set of item IDs and the item selection options. If the resulting set contains exactly one article item, the operation returns the item calculation sheet for this item. Otherwise it returns a group calculation sheet for all article items contained in the set of selected items. A group calculation is a calculation that summarizes zero or more item calculations.

In case of set-article items, the returned calculation sheet represents the sum of all set-article parts, multiplied with the quantity of the set-article item.

In case of folder items with option `subItems` set to `true`, the returned price calculation sheet will represent the sum of all calculations of article items within the folder, except for article items that are alternative positions.

If the folder contains a set-article item, then parts of the set-article item also contained in the folder are ignored. If the folder contains a part of a set-article, but not the set-article item itself, then the part of the set-article is treated like an ordinary article (i.e. the quantity of the set-article item is ignored).

The `ppName` argument may be an empty string if only one pricing procedure has been added to the basket. `counter` may be -1 if the calculation contains only one condition of the given type.

The option `viewId` of `ItemSelectionOptions` is supported by this operation. If it contains the ID of a view then the behavior of the operation is changed as follows:

- Item IDs passed with parameter `itemIds` are interpreted as IDs of items within this view.
- The options `parentItems` and `subItems`, if set, affect the selection of additional view items, not basket items.
- The set of view items is mapped to a set of basket items, replacing each view item with the set of directly referenced basket items[90].
- Note that folder view items are mapped to basket folder items, but, at least with display mode `Sorted`, the content of both folders does not necessarily consist of the same set of article items, so the use of folder view items may lead to surprising results.
- The values of options `wholeComposite` and `sumUpComposite` are set to `true` if the view uses merge mode `Compact`.
- The value of option `itemCondAmountPerUnit` is set to `true` for each returned group calculation that grouped a whole composite article or a set of identical composite articles. The client may explicitly set this option to ensure consistent behavior for ordinary articles and partial plannings or aggregates with sub-articles.

5.6.3.58 `getPriceCalculationSheets`

Synopsis:

```
ItemCalculationSheet* getPriceCalculationSheets(
                      SessionId                          sessionId,
                      BItemId[]                                 itemIds,
                      GetPriceCalculationSheetOptions    options,
                      string                             ppName)
          throws BasketServiceFault;
```

The operation `getPriceCalculationSheets` is used to fetch multiple price calculation sheets using a single web service operation.

The operation determines a set of selected items based on parameters `itemIds` and `options`. It then attempts to get a price calculation sheet for each selected item.

In case of article items the price calculation sheet contains the calculation for this article item only.

90 Indirectly referenced basket items, like sub-article items indirectly referenced by article view items in merge mode Compact, or set-article parts in case of collapsed set-articles, are not included.

Otherwise, the price calculation sheet contains an aggregate calculation for the selected item and all sub-positions of the selected item.

For each produced price calculation sheet, the operation returns an ItemCalculationSheet element (in unspecified order), containing the item ID and the price calculation sheet.

The option `viewId` of `ItemSelectionOptions` is supported by this operation. If it contains the ID of a view then the behavior of the operation is changed as follows:

- Item IDs passed with parameter `itemIds` are interpreted as IDs of items within this view.
- The options `parentItems` and `subItems`, if set, affect the selection of additional view items, not basket items.
- One instance of type `ItemCalculation` is returned for each selected view item. The `id`-field contains the item ID of the view item. The calculation is constructed from the item calculations of all directly referenced basket items.
- Note that folder view items are mapped to basket folder items, but, at least with display mode `Sorted`, the content of both folders does not necessarily consist of the same set of article items. So calculations returned for folder view items may not contain the result expected based on the view's item structure.
- The values of options `wholeComposite` and `sumUpComposite` are set to `true` if the view uses merge mode `Compact`.
- The value of option `itemCondAmountPerUnit` is set to `true` for each returned group calculation that grouped a whole composite article or a set of identical composite articles. The client may explicitly set this option to ensure consistent behavior for ordinary articles and partial plannings or aggregates with sub-articles.

5.6.3.59 `addManualCondition`

Synopsis:

```
int addManualCondition(SessionId                sessionId,
                       BItemId[]                itemIds,
                       string                   ppName,
                       string                   condType,
                       ItemSelectionOptions     options)
        throws BasketServiceFault;
```

Add a manual condition to the calculation for the items identified by `itemIds` and the pricing procedure identified by `ppName`. If `itemIds` is the Null-ID, add a header condition. Returns the counter of the newly added condition, or -1 if condition was already added to the calculation.

The `ppName` argument may be an empty string if only one pricing procedure has been added to the basket. `counter` may be -1 if the calculation contains only one condition of the given type.

5.6.3.60 `removeCondition`

Synopsis:

```
boolean removeCondition(SessionId               sessionId,
                        BItemId[]               itemIds,
                        string                  ppName,
                        string                  condType,
                        int                     counter,
                        ItemSelectionOptions    options)
        throws BasketServiceFault;
```

Remove the specified condition.

The ppName argument may be an empty string if only one pricing procedure has been added to the basket. counter may be -1 if the calculation contains only one condition of the given type.

The operation fails if called for a folder.

5.6.3.61 setConditionAmount

Synopsis:

```
void setConditionAmount(SessionId                sessionId,
                        BItemId                  itemIds,
                        string                   ppName,
                        string                   condType,
                        int                      counter,
                        decimal                  amount,
                        string                   currency,
                       CondGroupSelectionOptions      options)
        throws BasketServiceFault;
```

Set the amount of the condition. currency must be "%" for conditions with calculation rule Percent, or (currently) equal to the document currency.

To set the condition amount of a header condition, itemIds must be the Null-ID. The set of selected items is determined based on the specified set of item IDs and the item selection options. If the resulting set contains exactly one article item, the operations operate on the corresponding item calculation. Otherwise they operate on a (temporary) group calculation that aggregates the individual item calculations.

The ppName argument may be an empty string if only one pricing procedure has been added to the basket. counter may be -1 if the calculation contains only one condition of the given type.

The operation fails if:

- an attempt is made to set a value with the wrong sign.
- the new amount is not within these bounds.
- It is called for a folder
- the aggregate condition or its amount are not editable.

5.6.3.62 resetConditionAmount

Synopsis:

```
void resetConditionAmount(SessionId              sessionId,
                          BItemId[]              itemIds,
                          string                 ppName,
                          string                 condType,
                          int                    counter,
                         CondGroupSelectionOptions   options)
        throws BasketServiceFault;
```

Resets the condition amount to the initial value (before edited by setConditionAmount).

To reset the condition amount of a header condition, itemIds must be the Null-ID. The set of selected items is determined based on the specified set of item IDs and the item selection options. If the resulting set contains exactly one article item, the operations operate on the corresponding item calculation. Otherwise they operate on a (temporary) group calculation that aggregates the individual item calculations.

The ppName argument may be an empty string if only one pricing procedure has been added to the basket. counter may be -1 if the calculation contains only one condition of the given type.

The operation fails if called for a folder.

5.6.3.63 setQuantityRelation

Synopsis:

```
void setQuantityRelation(SessionId              sessionId,
                         BItemId[]              itemIds,
                         string                 ppName,
                         string                 condType,
                         int                    counter,
                         decimal                quantity,
                         string                 unit,
                        ItemSelectionOptions   options)
        throws BasketServiceFault;
```

Sets the quantity relation for a calculation line (e.g. by piece).

It fails if the new value is either negative or zero or if called for a folder.

The ppName argument may be an empty string if only one pricing procedure has been added to the basket. counter may be -1 if the calculation contains only one condition of the given type.

5.6.3.64 resetQuantityRelation

Synopsis:

```
void resetQuantityRelation(SessionId            sessionId,
                           BItemId[]            itemIds,
                           string               ppName,
                           string               condType,
                           int                  counter,
                          ItemSelectionOptions options)
        throws BasketServiceFault;
```

Resets the quantity relation of a condition to the initial quantity relation (before edited by setQuantityRelation).

The ppName argument may be an empty string if only one pricing procedure has been added to the basket. counter may be -1 if the calculation contains only one condition of the given type.

The operation fails if called for a folder.

5.6.3.65 getConditionTypes

Synopsis:

```
ConditionType[] getConditionTypes(SessionId sessionId,
                                  boolean   active,
                                  string    ppName,
                                  string[]  names)
        throws BasketServiceFault;
```

Returns either all or a specified subset of the conditions of the specified pricing procedure.

The ppName argument may be an empty string if only one pricing procedure has been added to the basket.

5.6.3.66 listTaxSchemes

Synopsis:

```
TaxSchemeDescription[] listTaxSchemes(SessionId sessionId,
                                      boolean   active,
                                      string    country)
        throws BasketServiceFault;
```

Lists short information (symbolic identifier, country and region, and, optional, description of variant) about available tax schemes.

The parameter `country` may be used to restrict the list of returned tax schemes to those for the specified ISO country code. In case of an empty string, all tax schemes are returned.

The parameter `active` is used to select between the default set of tax schemes (`false`), as available after a new project has been created, or the set of tax schemes available in the current project `true`. The latter set consists of the first one plus the tax scheme loaded as part of the project file. If the default set of tax schemes contains a scheme with the same identifier as the project tax scheme, this scheme is hidden by the project tax scheme.

5.6.3.67 `getTaxScheme`

Synopsis:

```
TaxScheme getTaxScheme(SessionId sessionId,
                       boolean   active,
                       string    schemeId)
        throws BasketServiceFault;
```

`getTaxScheme` returns detailed information about the tax scheme specified by the `schemeId` parameter.

In addition to the short information returned by `listTaxSchemes`, this operation returns the currency required by the tax scheme (or an empty string if the tax scheme does not require a particular currency), the tax types (like `VAT`), and, for each type, the tax categories (like `standard_rate` in case of `VAT`) and the tax rate.

The value of attribute `rateUnit` of complex type `taxType` is either `%` in case of relative taxes, or `<cy>/<uom>`, where `<cy>` is a currency code and `<uom>` is an UNECE unit of measure. With the current implementation of pricing procedures, both weight and volume units are meaningful.

The parameter `active` is used to select between the default set of tax schemes (`false`), as available after a new project has been created, or the set of tax schemes available in the current project `true`. The latter set consists of the first one plus the tax scheme loaded as part of the project file. If the default set of tax schemes contains a scheme with the same identifier as the project tax scheme, this scheme is hidden by the project tax scheme.

5.6.3.68 `getCurrentTaxScheme`

Synopsis:

```
string getTaxSchemes(SessionId sessionId)
        throws BasketServiceFault;
```

The operation `getCurrentTaxScheme` returns the identifier of the current tax scheme.

5.6.3.69 `selectCurrentTaxScheme`

Synopsis:

```
void selectCurrentTaxScheme(SessionId sessionId,
                            string    schemeId)
        throws BasketServiceFault;
```

The operation `selectCurrentTaxScheme` makes the specified tax scheme the current tax scheme.

5.6.3.70 `getTaxInformation`

Synopsis:

```
TaxInfo[] getTaxInformation(SessionId sessionId,
                            BItemId   itemId)
        throws BasketServiceFault;
```

The new operation `getTaxInformation` returns all known tax information for the specified article item. The tax information consists of a possibly empty list of pairs of tax type and tax category identifiers (like `VAT` and `reduced_rate`), specifying the tax category to be used by this article for the tax type.

5.6.3.71 `setTaxInformation`

Synopsis:

```
void setTaxInformation(SessionId sessionId,
                       BItemId   itemId,
                       string    taxType,
                       string    taxCategory)
        throws BasketServiceFault;
```

The operation `setTaxInformation` may be used to set the tax category which is to be used for the given tax type for the specified article. The article must be an user article. The tax type must be part of the current tax scheme, and the tax category must be valid for the tax type (according to the tax scheme).

5.6.3.72 `resetTaxConfiguration`

Synopsis:

```
void resetTaxConfiguration(SessionId sessionId)
    throws BasketServiceFault;
```

Reset tax configuration of specified session to same state as after operation openSession. All of the session’s current tax schemes are discarded, including the project tax scheme (tax scheme possibly loaded as part of current project from an OBK file), and effectively replaced by all globally configured tax schemes. Furthermore, the current tax scheme is set according to the session’s current locale.

5.6.3.73 `resetTaxScheme`

Synopsis:

```
void resetTaxScheme(SessionId sessionId, string schemeId)
    throws BasketServiceFault;
```

Reset the specified tax scheme of the specified session to have the same tax types, categories and rates as the globally configured tax scheme with the same ID.

Nothing happens if the specified scheme ID is equal to the ID of the project tax scheme (tax scheme loaded as part of current project from an OBK file) and the global tax configuration does not have a tax scheme with the same ID.

5.6.3.74 `setTaxRate`

Synopsis:

```
void setTaxRate(SessionId sessionId,
                string schemeId,
                string typeId,
                string categoryId,
                decimal rate)
    throws BasketServiceFault;
```

For the specified session, sets the tax rate of the specified tax category to the given value in percent.

5.6.3.75 `tmGetTable`

Synopsis:

```
TMTable tmGetTable {
      string sessionId,
      string viewId,
      string itemId
      }
```

5.6.3.76 `tmGetText`

Synopsis:

```
TMRow[] tmGetText {
      string sessionId,
      string viewId,
      string itemId,
      boolean allLanguages,
      boolean enableLangTags
                  }
```

If called with parameter `allLanguages` set to `false` it falls back to the text stored for the undetermined language if no entry is found for a language from the prioritized list of effective product data languages.

5.6.3.77 `tmSetText`

Synopsis:

```
void tmSetText {
      string sessionId,
      string viewId,
      string itemId,
      string textId,
      string lang,
      string text
      }
```

The operation sets the text for the given language tag. In addition, it also sets the text for all language tags which are proper prefixes of the given one, and removes all texts stored for tags which have the given tag as a proper prefix. (Does not apply if `tmSetText` is used to set a product text (short, long, features) of an article item or the short/long text of a basket folder or text item)

Note: For best interoperability with pCon.basket offline, texts should be set for language tags consisting solely of an ISO 639-1 (Alpha 2) language code. The last text stored that way will also be stored for the undetermined language and will be used by EAIWS as a fallback if no entry matching one of the effective product data languages is found.

However, texts stored for the undetermined language are preserved but otherwise ignored by pCon.basket offline. If an OBK/OBX file created by EAIWS containing such a text is loaded into pCon.basket offline, the text of the row gets changed (a text with an Alpha 2 language code is added or changed), and the file is saved and loaded again in EAIWS, EAIWS will see the new text for the specific language, but will continue to use the old fallback text.

Similarly, if an OBK/OBX file created by pCon.basket offline contains such texts in one or more languages (using Alpha 2 language codes), the file is loaded by EAIWS, and the effective list of product data languages used by EAIWS contains none of these languages, EAIWS will return none of the stored texts (unless operation `tmGetTexts` with parameter `allLanguages` set to `true` is used).

5.6.3.78 `tmSetTextVisibility`

Synopsis:

```
void tmSetTextVisibility {
      string sessionId,
      string viewId,
      string itemId,
      string textId,
      boolean visible
}
```

5.6.3.79 `ResetTaxRate`

Synopsis:

```
void resetTaxRate(SessionId sessionId,
                  string schemeId,
                  string typeId,
                  string categoryId);
```

For the specified session, resets the tax rate of the specified tax category to it’s default value.

The operation has no effect if the specified scheme ID is equal to the ID of the project tax scheme (tax scheme possibly loaded as part of current project from an OBK file) and there is no globally configured tax scheme with the same ID. This is because default values are not stored as part of the OBK file but taken from the global tax configuration.

5.6.3.80 `updateBasketArticles`

Synopsis:

```
UpdateBasketArticleResult[] updateBasketArticles(
                  SessionId                    sessionId,
                  BItemId[]                    itemId,
                  string[]                     catalogIds,
                  UpdateBasketArticleOptions   options)
        throws BasketServiceFault;
```

The specified item IDs must reference existing items, but Ids of items other than basket article items are ignored.

The operation determines a set of basket main article items based on the set of specified item IDs and the `wholeComposite` attribute of the options argument. If no options are specified, or the `wholeComposite` attribute has not been specified, then whole-composite defaults to false. If whole-composite is true, then a sub-article item specified by one of the item IDs results in the selection of its main article item.

(The attributes `subItems` and `parentItems` of complex type `ItemSelectionOptions` are ignored.)

Once all basket main article items have been determined, the operation determines the update state for each article item, possibly using the set of specified catalog IDs to restrict the set of OFML catalogs as described above in the documentation of `UpdateState`.

If the update state is `Updatable`, and the effective value of the update option is true, the article is updated. If the update state is `Migratable`, and the effective value of the migrate option is true, the article is migrated.

If the article is updated or migrated, and update or migration succeeded, the returned update state for this article will be `UpToDate`. If update or migration failed, the returned update state will be `Invalid`. In all other cases, the returned update state will be the update state originally determined for this article.

The order of returned elements of complex type `UpdateBasketArticleResult` is undefined and not guaranteed to be reproducible (two subsequent calls of `updateBasketArticles` do not necessarily return the elements in the same order, even if both `update` and `migrate` options are false).

Basically, update and migration mean the instantiation of an OFML article based on the installed OFML (product) data and the configuration (base article number, variant code, ...) of a basket article item read from a BSK/OBX file, followed by the transfer of the current article data from the OFML article item to the basket article item, replacing the old data read from the BSK/OBX file.

In case of an update, the configuration of the article item read from the BSK/OBX file and the configuration of the OFML article are supposed to be identical. In case of migration, the base article number of the OFML article should be the same, but other properties may be different.

(Subsequently, the term 'OFML catalog' is used as a synonym for a catalog profile and all OFML packages referenced from the catalog profile, or for a subset of the OFML packages referenced from a package profile (If the OFML and commercial manufacturer IDs of all referenced packages as well as the set of manufacturer display names are identical, the subset consists of all referenced packages. As this is usually the case, a package profile usually results in a single OFML catalog.))

The support for update and migration results in or affects the following four use cases:

- Update of an article. To update an article, EAIWS must first determine an OFML catalog that contains the article, followed by the re-instantiation of the OFML article, followed by the transfer of the article data from the OFML article to the basket article item.

  In case of multiple registered catalogs that contain different versions and/or distribution regions for the same article, it may be necessary to specify a subset of catalogs to allow EAIWS the selection of a single OFML catalog.
- Migration of an article. This is basically the same as the update of an article, except that the configuration of the migrated article is not identical with the original article.
- Determination of the update state. This determines whether a basket article item can be updated or migrated with respect to a certain set of OFML catalogs.
- Reconfiguration of an article. This is done when a user selects an article, displays the property editor, and changes a property. The web service operations involved are (at least) 'getArticleData' and 'setPropertyValue'. The first operation re-instantiates the OFML article, the second operations sets the new property value and copies the article data from the OFML article into the basket article item.

  For the first operation (re-instantiation of the OFML article) to succeed, the article item must be 'Updatable' with respect to all registered OFML catalogs. If this is not the case, it must be updated or migrated first, possibly with a subset of OFML catalogs specified.

5.6.3.81 `getReferenceCurrency`

Synopsis:

```
string getReferenceCurrency(SessionId sessionId)
        throws BasketServiceFault;
```

All exchange rates are defined with respect to the reference currency. If, for instance, the reference currency is 'EUR', and the exchange rate for 'USD' is 1.4303, then 1 EUR = 1.4303 USD.

5.6.3.82 `getExchangeRates`

Synopsis:

```
ExchangeRate[] getExchangeRates(SessionId      sessionId,
                                string[]    currencies)
        throws BasketServiceFault;
```

With the operation `getExchangeRates` all or a subset of the currently defined exchange rates are accessible.

The argument `currencies` represents zero or more currency codes.

If no currency has been specified, the operation returns the exchange rates of all currencies known to the currency converter.

If at least one currency has been specified, the operation fails with a `BasketServiceFault` if at least one currency is not a valid ISO 4217 currency code (excluding pseudo currencies (all currency codes starting with 'X' except 'XAF', 'XOF', 'XPF' and 'XCF')). The exchange rates are returned in the same order as the currency codes, one instance of type `ExchangeRate` for each currency. If one of the currencies is not a known currency (not known to the currency converter), the attribute `rate` of the returned exchange rate is missing.

5.6.3.83 `setExchangeRates`

```
void setExchangeRates(SessionId          sessionId,
                      ExchangeRate[]     exchangeRates,
                      string?            referenceCurrency)
        throws BasketServiceFault;
```

Operation to set zero or more exchange rates.

For each specified instance of `ExchangeRate`, the attribute `currency` must be an ISO 4217 currency code known to the currency converter, and the attribute `rate` must be present and specify a number greater than zero. If the currency is equal to the reference currency, the rate must be 1.0.

If the operation terminates normally, all exchange rates have been updated as specified. Otherwise, if the operation terminates abnormally with a `BasketServiceFault`, no exchange rates have been modified. Otherwise (abnormal termination with another fault) the behavior is undefined.

If argument `referenceCurrency` is present and contains a non-empty string, it must be one of the ISO 4217 currency codes returned by operation `getExchangeRates`. If so, it is set as the new reference currency, the conversion rate of this new reference currency is set to 1.0, and the conversion rates of all other currencies are updated to reflect the new reference currency.

New exchange rates specified by parameter `exchangeRates` will be set after the reference currency has been changed.

The implementation attempts to make sure that changing reference currency and exchange rates in one operation is atomic.

Note that changing the reference currency, and changing it back to the original reference currency, may result in slightly different conversion rates due to rounding of (intermediate) exchange rates to at most six significant digits.

5.6.3.84 `convertToUserArticles`

```
BItemId convertToUserArticles(SessionId               sessionId,
                              BitemId                 itemIds,
                              ItemSelectionOptions    options)
        throws BasketServiceFault;
```

Operation to convert a basket article into an user article.

Determines a set of selected items based on the specified item Ids and optional item selection options. Then, for each basket main article item with the set, converts the basket article item into a user article item. In case of composite article items, the main article and each sub-article is converted into an independent user article.

After successful conversion the original basket article items are deleted.

The operation returns a sequence of basket item IDs with the same number of elements as passed to the `itemIds` parameter. If the N-th parameter is the ID of a basket article item, then the N-th return value is the ID of the user article item created for this basket article item. Otherwise, the N-th return value is equal to the N-th parameter if the item still exists, or the NIL-ID if the item has been deleted for whatever reason (should not happen).

5.6.3.85 `getBasketColumns`

```
BasketColumn[] getBasketColumns(SessionId      sessionId,
                                ColumnId[]     columnIds)
          throws BasketServiceFault;
```

Return information about the basket columns specified by the `columnIds` parameter. If no column ID has been specified, then information about all columns configured for the current basket is returned.

All attributes and elements defined for the `BasketColumn` type are present in the returned basket column elements.

The operation fails with a `BasketServiceFault` if

- The session ID, or one of the column IDs, is not a properly formatted UUID.
- The session ID does not reference an existing session.
- One of the column IDs does not reference a column within the basket of the specified session.

5.6.3.86 `removeBasketColumns`

```
void removeBasketColumns(SessionId       sessionId,
                         ColumnId[]      columnIds)
          throws BasketServiceFault;
```

Remove zero or more columns from the basket of the specified session.

The implementation of this operation attempts to make sure, but cannot guarantee, that no change to the column configuration took place in case of an abnormal termination.

The operation fails with a `BasketServiceFault` if

- The session ID, or one of the column IDs, is not a properly formatted UUID.
- The session ID does not reference an existing session.
- One of the column IDs does not reference a column within the basket of the specified session.
- One of the specified columns cannot be removed. This is usually because the column is a pre-defined/built-in column.

5.6.3.87 `addBasketColumns`

```
BasketColumn[] addBasketColumns(SessionId      sessionId,
                                BasketColumn[]  columns)
          throws BasketServiceFault;
```

Add zero or more columns to the basket of the specified session.

The columns, and their initial configuration, are specified by parameter `columns`.

If the `id` attribute of a column is an empty string or the NIL UUID, then a random UUID will be generated for this column.

The `type` attribute must be specified and must not be `Undefined` or `Builtin`.

The `itemAttrId` attribute should be left unspecified. If specified, it must be `UserDefined`.

The `defaultColumn` attribute, if specified, must be either an empty string or a properly formatted UUID. If not specified or an empty string it will be replaced by the NIL UUID. The NIL UUID is used to indicate that there is no default column.

The `readOnly` attribute, if not specified, defaults to `false`.

The `name` and `title` elements must be both present and contain non-empty strings.

The `defaultValue` element, if not present, defaults to an empty string.

It is an error if the `defaultColumn` attribute contains an UUID other than the NIL UUID, and the `default-Value` element contains a non-empty string.

Columns are added, one after another, in the order they appear in the `columns` parameter. If addition of a column fails, then all columns added so far by this operation are removed again, and the operation fails.

Possible reasons for the addition of a column to fail are:

- There is already a column with the same ID.
- A non-NIL default column ID has been specified, and there is no column with this ID.

If the operation succeeds, it returns information about all added columns. The number and order of elements in the return value is the same as in parameter `columns`. The information returned is the same as would be returned for the same columns by operation `getBasketColumns (§Fehler: Verweis nicht gefun-den).`

The operation fails with a `BasketServiceFault` if

- The session ID is not a properly formatted UUID.
- The session ID does not reference an existing session.
- For any of the error conditions described above.

5.6.3.88 `setBasketColumnProperties`

```
void setBasketColumnProperties(SessionId       sessionId,
                               BasketColumn[]   columns)
          throws BasketServiceFault;
```

Set properties of zero or more columns of the specified session's basket.

The `id` attribute of each column must be a properly formatted UUID and must identify an existing column.

The `type` attribute, if specified, must not be `Undefined` or `Builtin`.

The `itemAttrId` attribute, if specified, must be `UserDefined`.

The `defaultColumn` attribute, if specified and non-empty, must be a properly formatted UUID and either identify an existing column or be the NIL UUID.

Note that a specified, but empty, `defaultColumn` attribute is not interpreted as the NIL UUID. To reset the default column, an actual NIL UUID must be specified.

The `name` and `title` elements, if present, must be non-empty.

It is an error if the `defaultColumn` attribute contains an UUID other than the NIL UUID, and the `default-Value` element contains a non-empty string.

The implementation of this operation attempts to make sure, but cannot guarantee, that no change to the column configuration took place in case of an abnormal termination.

Columns are modified, one after another, in the order specified by parameter `columns`. For each column, properties are modified as follows, in the given order:

- If the `type` attribute is specified, then the column type is set to the given value.
- If the `name` element is present, the column name is set to its value.
- If the `title` element is present, the column title is set to its value.
- If the `readOnly` attribute is specified, then the read-only property of the column is adjusted accord-ingly.
- If the `defaultColumn` attribute is specified and non-empty, the default column of the current

column is set accordingly. If the attribute contains the NIL UUID, then the default column will be reset. Otherwise, the default column is set to the column with the specified ID and the default value of the column, if any, is reset.

- If the `defaultValue` element is present (empty or not), the default value of the column will be set accordingly.

If any of these steps fails for whatever reason, the `setBasketColumnProperties` operation fails immediately. No configuration change done so far is undone.

The operation fails with a `BasketServiceFault` if

- The session ID is not a properly formatted UUID.
- The session ID does not reference an existing session.
- For any of the error conditions described above.

5.6.3.89 `getItemFields`

```
ItemField[] getItemFields(SessionId             sessionId,
                          BitemId[]             itemIds,
                          ColumnId[]            columnIds,
                          GetItemFieldsOptions  *options)
        throws BasketServiceFault;
```

Get the values for the fields defined by the intersection of one or more basket columns and a set of selected basket items (rows)0 of the specified session's basket.

Basket items whose fields are to be returned are selected based on the specified item IDs and the item selection options that may be given with the `options` parameter. If no item ID is specified, then all currently existing basket items are selected, including the top folder.

The operation returns field values for the specified columns or, if not columns have been specified, for all columns found in the session's basket. If a column is specified more than once, then fields of this column may be returned multiple times.

If a view ID has been specified (field `viewId` of type `ItemSelectionOptions` inherited by type `GetItemFieldsOptions`), then the item IDs must be view item IDs, the position number will be returned for the position number column, and images returned for items of views with merge mode Compact will include sub-articles.

There is no guarantee about the order of returned fields. For each field with a non-empty value, the item ID, column ID and value are returned.

If one of the column IDs specifies a user-defined column of type `Number`, and the column references a variable (directly or indirectly through its default value or column), or a default predefined column, then formatting of numeric values will be modified so the result adheres to the number syntax specified for column type `Number`. Most importantly, this enforces the use of the period (U+002E) as decimal separator, disables the use of grouping, disables the insertion of units (currency, percent sign, UNECE units of measure) into the field's value, and disables unit conversion and rounding to specified precision in case of packaging information. Furthermore, in case of the `TaxRate` variable and a valid quantity relation, it causes division of the amount by the quantity relation (for example, if the condition amount is 3 EUR, and the quantity relation is 2 KGM, then field's value will be 1.5).

The operation fails with a `BasketServiceFault` if

- The session ID, one of the item IDs, or one of the column IDs is not a properly formatted UUID.
- The session ID does not reference an existing session.
- One of the item IDs does not reference an item currently existing in the session's basket.
- An item ID is specified more than once (optional).
- One of the column IDs does not reference a column within the session's basket.

Operation 'getItemFields' properly hande a specified view ID (attribute 'viewId' of complex type 'ItemSelectionOptions' inherited by complex type 'GetItemFieldsOptions'). If a view ID has been specified, then the item IDs must be view item IDs, the position number will be returned for the position number column, and images returned for items of views with merge mode 'Compact' will include sub-articles.

5.6.3.90 `setItemFields`

```
void setItemFields(SessionId         sessionId,
                   ItemField[]       fields)
          throws BasketServiceFault;
```

Set the values of zero or more fields identified by column and item ID.

The implementation of this operation attempts to make sure, but cannot guarantee, that no field value has been changed in case of an abnormal termination of the operation.

The operation fails with a BasketServiceFault if

- The session ID, or one of the column and item IDs is not a properly formatted UUID.
- The session ID does not reference an existing session.
- One of the column IDs does not identify an user-defined column of the session's basket.
- One of the column IDs identifies a read-only column.
- One of the item IDs does not a currently existing item of the session's basket.
- One of the values does not conform to the requirements defined for the corresponding column type.

5.6.3.91 `resetItemFields`

```
void resetItemFields(SessionId            sessionId,
                     BitemId[]            itemIds,
                     ColumnId             columnId,
                     string*              data,
                     ItemSelectionOptions *options)
          throws BasketServiceFault;
```

Reset the values of fields defined by the intersection of one user-defined basket column and a set of selected basket items (rows) of the session's basket.

Basket items whose fields are to be reset are selected based on the specified item IDs and the item selection options that may be given with the `options` parameter. If no item ID is specified, then all currently existing basket items are selected, including the top folder.

If the parameter `data` is not specified, then an empty string will be used instead.

The implementation of this operation attempts to make sure, but cannot guarantee, that no field value has been changed in case of an abnormal termination of the operation.

The operation fails with a `BasketServiceFault` if

- The session ID, one of the item IDs, or the column ID is not a properly formatted UUID.
- The session ID does not reference an existing session.
- One of the item IDs does not reference an item currently existing in the session's basket.
- An item ID is specified more than once (optional).
- The column ID does not reference a user-defined column within the session's basket.
- The column ID identifies a read-only column.
- The new value specified with parameter `data` does not conform to the requirements defined for the column type (empty strings are always valid).

5.6.3.92 `OapGetArticleData`

Synopsis:

```
OAPArticleData? oapGetArticleData(SessionId     sessionId,
                                  BitemId itemId)
        throws BasketServiceFault;
```

Get the OFML Aided Plannning (OAP) article data for the basket article item specified by `itemId`.

No OAP article data is returned if `itemId` does not identify a basket article item, no OAP data is available for the article, for there was some error accessing the OAP data.

The OAP specification should be consulted for detailed information about the returned data.

5.6.3.93 `OapGetInteractors`

Synopsis:

```
OAPInteractor[] oapGetInteractors(
            SessionId   sessionId,
            BitemId     itemId,
            string[]    interactorIds
            float       dpr )
        throws BasketServiceFault;
```

Get information about currently active interactors. If no interactor IDs are specified, then all of the article's interactors are considered.

No interactor information is returned if itemId does not identify a basket article item, no OAP data is available for the article, for there was some error accessing the OAP data.

The OAP specification should be consulted for detailed information about OAP interactors.

`Dpr`
(default value 1.0).

5.6.3.94 `OAPInteractor`

Synopsis:

```
struct OAPInteractor {
    string                id;
    OAPSymbolType         symbolType;
    OAPSymbolSize         symbolSize;
    Vector3?              *symbolOffset;
    string[]              actionIds;
    string[]              directActionIds;
    OAPActionResult[]     directActions;
    OAPSymbolDisplay[]   symbols;
}
```

`directActionIds`
The sequence of action IDs returned by OFML is split into the sequence of direct action IDs and the sequence of ordinary action IDs. (The direct action IDs are the IDs of actions that are expected to affect the behavior of the interactor in some way.)

`directActions`

This sequence contains zero or one element of type 'OAPActionResult' for each direct action ID.

5.6.3.95 `OAPMessageAction`

`Synopsis:`

```
OAPMessageAction extends OAPAction {
    string              text
}
```

5.6.3.96 OAPSymbolType

Synopsis:

```
enum OAPSymbolType {
PropChange,
PropEdit,
CreateChild,
CreateChildren,
CreateSibling,
CreateSiblings,
DeleteObj,
Transform,
Translate,
Rotate,
RotatePX,
RotateNX,
RotateX180,
RotatePY,
RotateNY,
RotateY180,
RotatePZ,
RotateNZ,
RotateZ180,
Logics,
CheckRules,
Info,
Other,
Add,
Delete,
Edit,
Flip,
MovePlane,
MoveAxis,
MoveDir,
RotateNY90,
RotatePY90,
Material,
ChangeDimHorizontal,
ChangeDim2Left,
ChangeDim2Right,
ChangeDimVertical,
ChangeDimDown,
ChangeDimUp,
PosHorizontal,
Pos2Left,
Pos2Right,
PosVertical,
PosDown,
PosUp,
OnOff,
StartDimChange,
Video,
Attention
}
```

5.6.3.97 SymbolSize

Syno

psis:

```
enum SymbolSize { Small, Medium, Large }
```

5.6.3.98 OAPSymbolDisplay

Synopsis:

```
struct OAPSymbolDisplay {
    Vector3 symbolOffset;
    Vector3 *direction;
    Vector3 *orientationX;
    double  *viewAngle;
    boolean *HiddenMode;
}
```

hiddenMode
The default value of this attribute is 'false'.

5.6.3.99 oapProcessActions

Synopsis:

```
OAPActionResult[] oapProcessActions(SessionId        sessionId,
                                    string[]         actionIds,
                                    boolean          exec,
                                    OAPActionContext context)
        throws BasketServiceFault;
```

Query for process actions. Depending on parameter `exec` the operation behaves as follows:

`false`
: Process list of actions and return information up to and including first active action.

`true`
: Process list of actions and return information about inactive and active actions up to and including first active action that can not be executed by the server.

`context.self` must always contain the basket item ID of the article whose actions are to be queried or executed.

`context.interactor` must be set to the ID of the interactor if the list of actions has been obtained from an interactor.

`context.dpr` holds the display pixel ratio. The default value is 1.0. The display pixel ratio is used to select images returned with actions suitable for the client's display.

The fields of `OAPActionResult` are defined as follows:

`id`
: the ID of the action

`state`
: the state of the action; possible values are:

  `Disabled`: The action is disabled. This state may be returned with execution of actions enabled or disabled.

`Enabled`: The action is enabled. No attempt has been made to execute the action. This state may be returned only with execution of actions disabled.

`Success`: The action was executed successfully. This state may be returned only with execution of actions enabled.

`Failure`: Execution of the action failed. This state may be returned only with execution of actions enabled.

`NotResponsible`: The server does not feel responsible to execute the action. This may be because the action must be implemented by the client, or because execution of the action by the server is not implemented. This state may be returned only with execution of actions enabled.

An `OAPActionResult` with state `Enabled`, `Failure` or `NotResponsible` will always be the last returned action result.

`actionData`
: The OAP action data. No action data is returned if the action state is `Disabled`. See OAP specification for more information.

`objects`
: A list of basket item IDs that contains the result of resolution of object definitions contained in `actionData.objects`.

`addedItems`
: The basket item IDs of items added during execution of this action.

`removedItems`
: The basket item IDs of items removed during execution of this action.

`movedItems`
: The basket item IDs of items moved during execution of this action. Moved items are items whose position within the basket item tree has changed.

`updatedItems`
: The basket item IDs of items updated during execution of this action. Updated items are items whose article data has changed.

`referenceObject`, `newObjectPosition` and `newObjectRotation`

Fields `referenceObject`, `newObjectPosition` and `newObjectRotation` are used in instances of `OAPActionResult` holding the result of `CreateObj` actions with placement mode `AttachAreas`. Clients must use these fields to determine the position and rotation of the newly created object, and adjust its position and rotation accordingly, if all of the following conditions are met:

- Field `actionData` of the corresponding instance of `OAPActionResult` contains an action of type `OAPCreateObjectAction`.
- Field `parent` of this action is missing.
- Field `initialPlacement` of this action contains a placement of type `OAPAttachAreasPlacement`.

`referenceObject`

This field contains the basket item ID of the reference object.

`newObjectPosition`

This field contains the position of the origin of the local coordinate system of the newly created object within the local coordinate system of the reference object.

`newObjectRotation`

This field contains the rotation of the newly created object relative to its own local coordinate system. The rotation axis[91] runs through the origin of the local coordinate system. The angle is given in radian with positive magnitudes representing counter-clockwise rotations[92].

The current version of EAIWS will execute the following actions:

`CreateObjectAction` if the initial placement is of type `OAPDataDefinedPlacement`, the application placement specifies a parent, and the parent resolves to an item of `BasketItemType Aggregate` or `PartialPlanning`.

`MethodCallAction` if all selected items are OFML article items.

`DeleteObjectAction` if all selected items are sub-article items of basket composite articles.

`TranslateObjectAction and RotateObjectAction` if all selected items are OFML article items.[93]

The fields of OAPPropEditAction are defined as following:

`visibleOnly`

`editableOnly`

If `true`, these fields restrict the visibility of properties in OAP property editors. If `visible` and `editable` are the property-specific flags reported for individual properties (type `Property`), then the property must be visible if and only if the expression

```
(visible || !visibleOnly) && (editable || !editableOnly)
```

is `true`. Or, perhaps more intuitive, they must be hidden if

```
(visibleOnly && !visible) || (editableOnly && !editable)
```

is `false`.

5.6.3.100 `OapGetActionData`

Synopsis:

```
OAPAction[] oapGetActionData(SessionId    sessionId,
                             BitemId      itemId,
                              string[]    actionIds,
                              float?      dpr
                             string?       interactor
                             )
        throws BasketServiceFault;
```

`dpr`

(display pixel ratio). The default value is 1.0. The display pixel ratio is used to select images returned with actions suitable for the client’s display.

`interactor`

ID of the interactor that triggered an action

91 The euclidean norm of the rotation axis is one unless the angle is zero, in which case it is zero.

92 The current implementation returns angles with magnitudes ranging from 0 to 2π. Nevertheless, clients should at least support angles with magnitudes ranging from 2π to -π.

93 The semantics of these actions is not well defined right now. The current implementation treats translations as extrinsic (along the axes of the coordinate system of the parent) and rotations as intrinsic (around the axes of the local coordinate system). Rotations are converted to up to three elementary rotations around y, x and y. Translation and rotation along/around disabled axes is disabled, and rules are evaluated. Future implementations will probably ignore disabled translation and rotation axes and will not evaluate rules. Translations will probably be intrinsic, or there will be a parameter to select between extrinsic and intrinsic translations.

5.6.3.101 `OapSetClientCapabilities`

Synopsis:

```
void oapSetClientCapabilities(SessionId                 sessionId,
                             OAPClientCapability[]   capabilities)
throws BasketServiceFault;
```

This operation is used to inform the server about the set of OAP client capabilities supported by the calling client. See description of enumeration type `OAPClientCapability` (§ 5.6.1.106) for more information. The default set of client-supported OAP capabilities is empty.

5.6.3.102 `GetBasketConfig`

Synopsis:

```
BasketConfig getBasketConfig(SessionId sessionId)
    throws BasketServiceFault;

typedef string ViewId;
typedef string ColumnId;
```

5.6.3.103 `changeBasketConfig`

```
void changeBasketConfig(SessionId sessionId, BasketConfig basketConfig)
    throws BasketServiceFault;
```

This operation can be used to change the basket's default view or currency[94].

To change the default view, the attribute `defaultView` must be set to the ID of a current view of the basket. Setting the default view has no immediate effect, but it will be saved as part of the OBK file and applications that honour this information will use it to select the view initially displayed.

To change the basket's currency the attribute `currency` must be set to a valid and supported currency code.

The operation fails without any change to the basket configuration if

A default view ID has been specified that is not a valid UUID or neither the NIL-UUID nor the ID of a current view of the basket.

A currency has been specified that is not a valid currency code (pseudo currencies are not accepted) or, if it is a valid currency code, currencies are not editable or the currency is not supported by the basket's currency converter.

5.6.3.104 `getBasketViewConfigs`

```
BasketViewConfig[] getBasketViewConfigs(SessionId sessionId,
                                        ViewId[]  viewIds)
    throws BasketServiceFault;
```

`getBasketViewConfig` is used to query the configurations of all views of the current basket, or a subset of the view configurations.

5.6.3.105 `changeBasketViewConfig`

```
void changeBasketViewConfig(SessionId        sessionId,
                            BasketViewConfig viewConfig)
    throws BasketServiceFault;
```

94 The currency can also be changed with operation `setCurrency`.

The operation can be used to change all configuration options of a basket view. The only fields of `BasketViewConfig` that can obviously not be used to change a view configuration are `viewId`, `removable` and `editable`.

The field `viewId` must be present and contain the ID of an existing editable view. Fields `removable` and `editable`, if present, are ignored.

The view will always be reloaded (i.e. reconstructed from the basket item tree) if the operation effectively changes the display mode, an expand mode, the merge mode or the set-article mode, whether or not this change or changes would affect the structure of the current view.

The operation fails if

no valid view ID has been specified (invalid UUID or non-existent view)

the view is not editable

the specified display mode is `Undefined`

the specification of visible columns is invalid (invalid UUID, non-existent column, duplicate columns)

- the specified column widths are invalid (column ID does not specify visible column, column width less than or equal to zero, duplicate column widths)

5.6.3.106 `setItemConditionDescription`

Synopsis:

```
void setItemConditionDescription(
                                    string sessionId,
                                    string[] itemIds,
                                    string ppName,
                                    string condType,
                                    int counter,
                                    string lang,
                                    string description,
                                    ItemSelectionOptions options
                                 )
```

5.6.3.107 `addBasketView`

```
BasketViewConfig addBasketView(SessionId        sessionId,
                               BasketViewConfig template)
    throws BasketServiceFault;
```

The operation is used to add a new view to the basket. The basket view configuration passed as the `template` parameter contains the configuration options to be used for this view. The following default values are used for options that are not explicitly specified:

`viewId`	randomly generated UUID
`displayMode`	`Planning`
`expandGroups`	`true`
`expandPlanningFolders`	`true`
`expandBasketFolders`	`false`
`expandPartialPlannings`	`false`
`expandAggregates`	`false`
`mergeMode`	`None`
`autoColumnWidth`	`true`
`hiddenDiscounts`	`false`
`setArticleMode`	`Expand`

`name`	must be specified and must not be empty
`visibleColumns`	the list of standard columns (position number, manufacturer, series, article number, description, quantity, single net price, total net price)
`columnWidths`	empty

The operation returns the complete configuration of the newly added view.

The operation fails if

- No non-empty name has been specified.
- no valid view ID has been specified (invalid UUID or ID of already existent view)
- the specified display mode is `Undefined`
- the specification of visible columns is invalid (invalid UUID, non-existent column, duplicate columns)
- the specified column widths are invalid (column ID does not specify visible column, column width less than or equal to zero, duplicate column widths)

5.6.3.108 `startOFMLDebugging`

```
void startOFMLDebugging(
      string sessionId,
      string[] debugModes,
      int traceLevel,
      int detailLevel,
      string outputDevice,
      string[] debugClasses
)
```

`debugModes`

A list of zero or more of the following: Semantics (semantic errors), Collision (detected collisions), TableErr (errors reported by oiTable()), ExplWarn (explicit warnings), Func (function trace), Func2 (utility function trace), Warn (warnings), Info (further information), Time (time (in seconds) used by function), Time2 (start and end time of function), and Progress (reports calls to xOiSendProgressEvent2App()).

`traceLevel`

Function trace level (depth), must be non-negative

`detailLevel`

Detail level must be greater than zero

`outputDevice`

Either log or file (log if empty or missing).

If set to log, messages appear as NOTICE in application log file and, if configured, in session log (but note that the default size of the session log is probably insufficient to hold all the debug messages generated by operations like insertOFMLArticle or setPropertyValue).

If set to file, messages are appended to file debug.out in the current working directory (which should be the same directory as the current directory of EAIWS when it was started). If the file can't be opened for writing, messages are written to stderr, potentially interfering with the communication between FApi-Shell and EAIWS. Thus a client should use file if and only if it knows that FApi-Shells have permission to create a file in their current directory, or to write to the file if it already exists (and that the client has permission to read the file).

Furthermore, for EAIWS to accept file the application property egr.eai.gf.cobra.enable_file_io must be set to true.

`debugClasses`

The list of class names for which to produce debug messages. Unless the list is empty debug messages are limited to classes whose name is given in this list.

Class names must adhere to the syntax for OFML identifiers.

5.6.3.109 `stopOFMLDebugging`

```
void stopOFMLDebugging(
      string sessionId
)
```

5.6.3.110 `removeBasketViews`

```
void removeBasketView(SessionId sessionId, ViewId[] viewIds)
    throws BasketServiceFault;
```

The parameter `viewIds` must specify zero or more IDs of views of the basket. All specified views will be removed.

The operation fails, not removing any view, if at least one of the specified views is not removable.

5.6.4 Image Generation

5.6.4.1 Differences Between Online Configurator and pCon.basket

The handling of generated article images differs between Online Configurator and pCon.basket.

The client of the Online Configurator generates an article image invoking the `getGeneratedImage` operation (§5.6.3.40) for a particular article item, specifying (implicitly or explicitly) the desired rendering and camera settings as well as a tag. The Online Configurator computes an image identifier based on the current configuration of the article item and the specified settings. It then asks the global file cache (§4.1.16) whether it contains an image matching this identifier. If so, the image is used. Otherwise, an image is generated and stored in the cache. In either case, the article item stores a reference to the image under the tag specified by the `getGeneratedImage` operation. Consequently, there may be multiple images per article item, and different article items representing the same article configuration may reference different images, even under the same tag.

pCon.basket, on the other hand, maintains a per-project mapping from article configurations to generated article images. Whenever a generated image is needed for a particular article item, the article's current configuration is looked up in this mapping, and if found, the referenced image is used as the generated article image for this item. If not found, an image is generated using user defined global rendering and camera settings, the image is stored in the global mapping, and used as the generated article image for the item. Consequently, there is only one generated image per article item, generated on demand, and all article items containing the same configuration of the same article share the same image.

For compatibility with pCon.basket, the Online Configurator also supports the per-project mapping from article configurations to article images. Images stored in this mapping shall be called *PBK images*. The mapping is updated by the `getGeneratedImage` operation whenever an image is generated using the `default` tag. The mapping as well as the images referenced by the mapping are stored by the `saveSession` operation (§5.4.3.14) of the session web service, and restored by the `loadSession` operation (§5.4.3.15). A position's *PBK image*, if known, is returned by the `getImages` operation (§5.6.3.42) using an empty string as tag.

Other than pCon.basket, the `saveSession` operation of the Online Configurator's session web service does not attempt to generate any missing *PBK images*.

5.6.4.2 Client control of the header Content-Disposition for files in the session cache

Added mechanism to HTTP server that allows HTTP client to control `Content-Disposition` response header for GET requests referencing files in the session cache.

To add a `Content-Disposition` response header, the client must pass a URL with a query component that contains a `contentDisposition` parameter and an optional `filename` parameter. The value of the `contentDisposition` parameter must be either `inline` or `attachment`.

The `filename` parameter is ignored if its value contains a C0 (U+0000 to U+001F) or C1 (U+0080 to U+009F) control character, DELETE (U+007F), or a forward (U+002F) or backward (U+005C) slash. All other characters are allowed, even if they result in an invalid filename on the client system.

Furthermore, for the filename *not* to be ignored, both the `filename` specified by the filename query parameter and the actual filename must have an extension, and both extensions must be equal, or the mapping to a content type must produce the same result, which in turn must be different from `application/octet-stream`.

The mechanism is intended for use with URLs returned by operations `getGeneratedImage` and `getExportedGeometry` of the basket web service. The behavior (whether or not a `Content-Disposition` response header is produced) is undefined if it is used with other URLs.

The `contentDisposition` and filename query parameters must be encoded according to `application/x-www-form-urlencoded` as specified in Section 5.2, *application/x-www-form-urlencoded* serializing, of the WHATWG URL specification.

The `filename` parameter of the `Content-Disposition` response header field is encoded according to RFC 6266. `filename=` is used for filenames consisting entirely of code points less than U+0100, `filename*=` for all other filenames. UTF-8 is used as the encoding for `filename*=`.

5.7 Project Service

5.7.1 Type Definitions

Price lists, number schemes and project groups are identified by symbolic names. Identifiers for price lists and number schemes must be valid Unicode identifiers. Identifiers for project groups must be one or more non-empty sequences of upper-case ASCII letters and digits, starting with a letter, separated by colon.

A user role is a non-empty string that does not start or end with white space.

A number scheme pattern is a non-empty string that does not start or end with white space and contains exactly one %<width>n placeholder and at most one %<width>y placeholder. <width> is a sequence of one or more decimal digits interpreted as a decimal number that must be greater than zero and less than 100.

The current year is maintained by EAIWS (it cannot be set with operation updateProjectNumberScheme). The zone ID is used to convert the current time (in UTC) into a date and time in the given time zone to determine the current year. When the number scheme is used to create a new project number and the current year stored in the number scheme is uninitialized or less than the year derived from the current time scheme is set to the minimum value before the project number is generated.

Project group permissions consist of a user role and flags for create, read, write, delete and list rights:

create: the user is allowed to create a project in the group
read: the user is allowed to read (open) projects in the group
write: the user is allowed to write (save) projects in the group
delete: the user is allowed to delete projects in the group
list: the user is allowed to list all projects in the group

Language tag maps represent mappings from language tags (subset of IETF language tags consisting of language, script and region) to non-empty display names. They must have at least one entry for the undetermined language ('und').

5.7.1.1 `ProjectData`

Synopsis:

```
struct ProjectData {
    string        uri;
    string        projectName;
    string        projectNumber;
    string        description;
    string        externalReferenceNumber;
    string        externalReferenceText;
    string        customerNumber;
    string        externalCustomerId;
    string        customerRelatedRemarks;
    string        company;
    string        generalAgreementNumber;
    string        keywords;
    AddressData[] addresses;
    string        projectGroup;
    string        projectId;
    int           projectVersion;
    string        priceList;
    ProjectState  projectState;
    dateTime      created;
    dateTime      lastModified;
    date          projectDate;
    date          validToDate;
    string        createdByUser;
    string        modifiedByUser;
    boolean       changed;
    boolean       locked;
    string        lockHolder;
    dateTime      lockTime;
    boolean       projectNumberEditable;
    ProjectText[] projectTexts;
    string[]      languages;
  }
```

`projectState`

Project state

`created`

date and time the project has been created

`lastModified`

date and time the project has been created

`keywords`

A single Keyword must consist of a non-empty sequence of Unicode code points belonging to character classes LETTER, MARK, NUMBER, SYMBOL, PUNCTUATION_CONNECTOR (e.g. '_') or PUNCTUATION_DASH (e.g. '-'). When keywords are set, leading and trailing white-space is allowed but will be removed. Embedded white-space and sequences of embedded white-space are replaced by a single space character (U+0020).

`uri`

Allow the URI of the current project to be explicitly set. If not set, or set to null, the path of the project file (OBK file) will be used to construct the URI.

Operations loadSession and saveSession of the session service set the URI the the URL (file, http, https) used to load or save the session. Similarly, operations loadProject and saveProject set the URI to the URI used by the project database (scheme 'pst').

`projectDate`

Each project has a project date. The initial project date is derived from the project's create date.

`locked`

Present in return value of operation 'getProjectData'; Note that the value 'true' means that the active/loaded project held the lock at some time during the execution of operation 'getProjectData'. At the time the client sees the value 'true' the lock may already have been stolen by someone else.

`lockHolder`

May be present in return value of operation 'listProjects'; The attribute is present if somebody holds the lock or this project. The attribute value represents the user associated with the session at the time the lock was acquired (the name may be empty if no user had been authenticated).

`lockTime`

May be present in return value of operation 'listProjects'; The attribute is present if somebody holds the lock for this project. The attribute value represents the time the lock was acquired.

Contains attributes 'lockHolder' and 'lockTime' now if somebody holds the lock for the current session's project.

optional attributes 'projectGroup', 'projectId' and 'projectVersion' to return the project group, ID (an UUID) and version of a project.

Changed: Contains the project's change state (i.e. if true, the project has changed since the last load or save).

`projectNumberEditable`

The attribute is always present in return value of operations 'getProjectData' and 'listProjects'. It is ignored in input parameter of operation 'setProjectData' (the reason the attribute is optional).

`projectTexts`

If the element is present, it contains a sequence of projectText elements of type ProjectText.

`languages`

Containing zero or more `language` elements of type string. The value of each `language` element must be an IETF BCP47 language tag (subtags other than language, script and region are ignored)

5.7.1.2 `ProjectState`

Synopsis:

```
enum ProjectState {
      Undefined,
      InProgress,
      Offered,
      Ordered,
      NotCommissioned,
      ApprovalRequired,
      WaitingForApproval,
      ApprovalDenied,
      Approved
}
```

5.7.1.3 `ProjectTextType`

Synopsis:

```
enum ProjectTextType {
       HeaderText,
       FooterText
       }
```

5.7.1.4 `AddressData`

Synopsis:

```
struct AddressData {
      string             addressNumber;
      string             addressId;
      string             title;
      string             name1;
      string             name2;
      string             name3;
      string             name4;
      string             street;
      string             street2;
      string             countryCode;
      string             postalCode;
      string             location;
      string             district;
      string             regionCode;
      string             poBox;
      string             taxCode;
      string             taxCodeEU;
      string             taxCodeUSA;
      CommAddress[]      commAddresses;
      ContactData[]       contacts;
      AddressType        addressType;
```

```
    }
```

The format of `postalCode` and `poBoxPostalCode` follows the OEX (OFML Business Data Exchange) Global specification:

Maximum Length:	10 characters
Allowed Characters:	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ space and – within. Whereas it is not permitted for several spaces/hyphens to follow one another.

5.7.1.5 CommAddress

Synopsis:

```
struct CommAddress {
      CommAddrType        type;
      ScopeInfo           scope;
}
```

5.7.1.6 ContactData

Synopsis:

```
struct ContactData {
      string              contactNumber;
      string              title;
      string              firstName;
      string              lastName;
      CommAddress[]       commAddress;
      int                 id;
      ContactType         contactType;
}
```

5.7.1.7 AutoSaveOptions

Synopsis:

```
struct AutoSaveOptions (
      boolean onSessionClose,
      boolean onProjectClose,
      boolean periodical,
      duration interval,
      duration idleTime
)
```

`onSessionClose`

If true, automatically save the project when the session is closed (default false)

`periodical`

If true, periodically save the project (default false)

`interval`

<iso-8601-duration> the time between two periodical auto-saves (default PT10M, minimum PT1M)

`idleTime`

<iso-8601-duration> the time a session must have been idle before a periodical auto- save kicks in (default PT10S, minimum PT2S)

`onProjectClose`

Auto-save the project (The default value is false) Can also be enabled with the startup file option. From the point of view of the current web service API a project is closed after a new project has been successfully opened by one of the following operations:

SessionWebService: loadSession, loadEmptySession
ProjectWebService: newProject, loadProject

5.7.1.8 `CountryData`

Synopsis:

```
struct CountryData (
      string displayName
      string iso3166Code
)
```

5.7.1.9 `DeleteProjectsOptions`

Synopsis:

```
struct DeleteProjectsOptions (
      boolean ignoreVersionNumbers,
      boolean failIfNonExistent,
      boolean failIfPermissionDenied,
      boolean failIfNotLockable,
      boolean ignoreLocks
)
```

5.7.1.10 `DisplayName`

Synopsis:

```
struct DisplayName : string (
      string language
)
```

5.7.1.11 `FormattedText`

Synopsis:

```
FormattedText extends string (
       string language,
       TextFormat format
       )
```

Attribute `language` is interpreted as a RFC 5646 language tag, but sub-tags other than language, script and region are discarded

5.7.1.12 `ListCompletionsOptions`

Synopsis:

```
struct ListCompletionsOptions (
      string                  prefix
      string                  projectGroupPattern,
      AddressType             addressType,
      ContactType             contactType,
      int                     limit,
      ProjectAttribute        filterAttribute,
      boolean                 ignoreCase
                               )
```

5.7.1.13 `ListProjectsOptions`

Synopsis:

```
struct ListProjectsOptions (
      string                  projectURI,
      string                  searchExpression,
      ProjectFilter           filter,
      ProjectSortSpec[]       orderBy,
      AddressType[]           resultAddressTypes
      string                  projectGroupPattern,
      AddressType             addressType,
      ContactType             contactType,
      int                     offset,
      int                     limit
)
```

`addressType and contactType`

If 'addressType' is set, then addresses of all other types are excluded from the full text search, and address columns are available for filtering and sorting. Similarly, if 'contactType' is set, contact columns are available for filtering and sorting. If an address does not have the specified address or contact, the corresponding fields are undefined.

`ResultAddressTypes`

Can be used to limit the types of addresses returned for each project found. If no address type is specified, then addresses of all types are returned.

`projectURI`

If the URI contains the version parameter, then only the given version is returned. Otherwise all versions are returned (unless limited by 'offset' and 'limit').

Option 'projectGroupPattern', if specified, must match the URI. Option 'searchExpression' is ignored. All other options ('filter', 'orderBy', 'offset' and 'limit') can be used as usual.

`Full-text search.`

To search the project database for a certain combination of search terms, set field 'searchExpression' of type ListProjectsOptions the the search expression. Fields 'filter' and 'projectGroupPattern' may be used to filter the search results, 'orderBy' may be used to order the filtered results, and 'offset' and 'limit' may be used to restrict the eturned projects to a chunk of the ordered results.

In its simplest form, the search expression consists of a sequence of white-space separated terms. Each normalized term matches search index entries starting with the same character sequence and containing at most two additional characters. For a project to be selected, at least one of the search terms must match an index entry referencing the project.

Furthermore, one or more white-space separated search terms may be enclosed in QUOTATION MARKs. For a project to match the search expression, for each of the enclosed search terms a matching index entry must be found referencing the project (exact match). If the search expression contains at least one search term enclosed in quotation marks, simple search terms are ignored (unless preceded by HYPHEN-MINUS).

Finally, both simple and quoted search terms may be preceded by a HYPHEN-MINUS. Basically, these 'negative' search terms select a set of projects that is subtracted from the set of projects selected by the 'positive' search terms. If there are no 'positive' search terms, it is subtracted from the set of all available projects.

(Note that due to normalization, results for unquoted search terms may be somewhat surprising. For instance, normalization of LATIN SMALL LETTER SHARP S results in two LATIN SMALL LETTER S. Thus, 'Straß' matches 'Straße', but 'Stra' does not.)

The exact process of normalization is subject to change. Right now it basically consists of the following steps:

1. convert all PUNCTUATION_DASH to HYPHEN-MINUS
2. convert all PUNCTUATION_CONNECTOR to LOW LINE
3. convert APOSTROPHE (punctuation) to ACUTE ACCENT (symbol)
4. remove all leading and trailing PUNCTUATION except PUNCTUATION_CONNECTOR
5. convert to upper case, then convert to lower case
6. compatibility decomposition, followed by canonical composition (Unicode normalization form NFKC)

5.7.1.14 `Permission`

Synopsis:

```
struct Permission (
      string? userRole,
      boolean create,
      boolean read,
      boolean write,
      boolean delete,
      boolean list
)
```

5.7.1.15 `PriceList`

Synopsis:

```
struct PriceList (
      string            id,
      string            startup,
      boolean           availableToCurrentUser
      userRole[]        userRoles,
      DisplayName[]     DisplayNames,
)
```

`id`

The symbolic name

`startup`
: The basename of the startup file

`userRoles`
: A possibly empty list of user roles that are allowed to use the price list

`displayName`
: A language tag map for the display name of the price list

`availableToCurrentUser`
: Used to indicate whether or not this price list can be used by the current session's user.

5.7.1.16 `ProjectFilter`

Synopsis:

```
struct ProjectFilter : ProjectFilterNode (
      ProjectFilterNode arguments
      ProjectFilterOperator operator,
      boolean ignoreCase
)
```

5.7.1.17 `ProjectFilterAttribute`

Synopsis:

```
struct ProjectFilterAttribute : ProjectFilterNode (
      ProjectAttribute attribute
)
```

5.7.1.18 `ProjectFilterDateValue`

Synopsis:

```
struct ProjectFilterDateValue extends ProjectFilterValue (date date)
```

5.7.1.19 `ProjectFilterDateTimeValue`

Synopsis:

```
struct ProjectFilterDateTimeValue : ProjectFilterValue (
      dateTime value
)
```

5.7.1.20 `ProjectFilterNode`

Synopsis:

```
abstract ProjectFilterNode ( )
```

5.7.1.21 `ProjectFilterStateValue`

Synopsis:

```
struct ProjectFilterStateValue : ProjectFilterValue (
      ProjectState value
)
```

5.7.1.22 `ProjectFilterStringValue`

Synopsis:

```
struct ProjectFilterStringValue : ProjectFilterValue (
      string value
)
```

5.7.1.23 `ProjectFilterValue`

Synopsis:

```
abstract ProjectFilterValue : ProjectFilterNode ( )
```

5.7.1.24 `ProjectGroup`

Synopsis:

```
struct ProjectGroup (
      ProjectNumberScheme numberScheme,
      Permission currentUserPermissions
      string id,
      boolean folder,
      PriceList[] priceLists,
      Permission[] permissions,
      DisplayName[] simpleNames,
      DisplayName[] fullNames,
      DisplayName[] subgroupTitles,
      Permission currentUserPermissions,
)
```

`id`

The symbolic name

`folder`

A boolean that indicates whether or not this project group acts as a container for other project groups

`numberScheme`

An optional element that contains the number scheme used by the project group, if any

`priceLists`

The list of price lists that may be used with the project group.

`permissions`

The list of permissions (user roles and their associated access rights) that apply to this group and projects in this group

`simpleNames`

A language tag map for the simple name of the project group

`fullNames`

A language tag map for the full name of the project group. By default, the full name of a project group that does not have a parent group is equal to the simple name, and the full name of a project group with a parent group is the concatenation of the full name of the parent group and the simple name of the group (the exact form of concatenation is defined by the user interface). However, if the map of full names contains a name for the language currently in use, the UI should use this name as the full name.

`subgroupTitles`

A language tag map for the titIf 'addressType' is set, then addresses of all other types are excluded from the full text search, and address columns are available for filtering and sorting. Similarly, if 'contactType' is set, contact columns are available for filtering and sorting. If an address does not have the specified address or contact, the corresponding fields are undefined.le used for subgroup selection

`currentUserPermissions`

Used to return the effective permissions of the current session's user regarding this project group.

5.7.1.25 `ProjectNumberScheme`

Synopsis:

```
struct ProjectNumberScheme (
      string id,
      string pattern,
      long minimumValue,
      long maximumValue,
      long nextValue,
      int currentYear,
      string zoneId
)
```

`id`

The symbolic name

`pattern`

A template for the project numbers generated by this number scheme

`minimumValue`

The minimum value used to replace %n in the pattern

`maximumValue`

The maximum value used to replace %n in the pattern

`nextValue`

The next value usedIf 'addressType' is set, then addresses of all other types are excluded from the full text search, and address columns are available for filtering and sorting. Similarly, if 'contactType' is set, contact columns are available for filtering and sorting. If an address does not have the specified address or contact, the corresponding fields are undefined. to replace %n in the pattern

`currentYear`

The current value used to replace %y in the pattern

```
zoneId
```

The time zone ID used to decide when exactly a new starts, nextValue is reset to minimumValue and currentYear is set to the current year

5.7.1.26 `ProjectSortSpec`

Synopsis:

```
struct ProjectSortSpec (
            ProjectAttribute attribute,
            boolean ascending,
            boolean undefFirst,
            boolean ignoreCase
)
```

5.7.1.27 `ProjectText`

Synopsis:

```
ProjectText (
       FormattedText[] texts
       ProjectTextType type
 )
```

The array of formatted texts (\`texts\`) holds (different) translations for the project text type stored in attribute \`type\`.

In return values all formatted texts use the same text format, and there are no duplicate language tags (although there may be duplicate language sub-tags).

If type \`ProjectText\` is used to set or update project texts, the texts are not required to use the same format, nor are the language tags required to be distinct (the texts are set in order, so all texts must be syntactically correct and adhere to the content model, but a later text for the same language tag will overwrite a previous one)

5.7.1.28 `SaveProjectOptions`

Synopsis:

```
struct SaveProjectOptions (
            string projectGroup,
            string priceList,
            boolean noVersionCheck,
            boolean autoDelete
)
```

```
noVersionCheck
```

(default value false).

Normally, if a project is loaded from the project store, and saved again, and the version number of the loaded project is not equal to the version number of the most recent version of the project at the time the project is saved again, the save operation fails. This may happen because another session saved a more recent version in the meantime, or because the loaded version was not the most recent version when loaded.

If option 'noVersionCheck' is set to true then this version check is disabled and a new version of the project is saved.

`autoDelete`

If set, the saved version will be automatically deleted the next time the project is saved (default false).

5.7.1.29 `SubdivisionData`

Synopsis:

```
struct SubdivisionData (
      string iso3166names,
      string[] localVariants,
      string languageCodes,
      string displayName
      string category,
      string iso3166Code,
      string regionCode,
      string parentSubdivision
)
```

`category`

The subdivision category as defined by ISO 3166-2

`iso3166code`

The subdivision code as defined by ISO 3166-2

`regionCode`

The subdivision code with the country prefix removed; This code must be used to set the region code of addresses

`parentSubdivision`

The subdivision code of the parent subdivision (or missing if this is a top level subdivision)

`languageCodes`

non-empty list of language codes as defined by ISO 3166-2 for the subdivision; All codes are alpha-2 codes, except for 'und' (which is used in cases where ISO 3166-2 uses '-' as language code).

`Iso3166names`

non-empty list of subdivision names as defined by ISO 3166-2. This list contains either one element or the same number of elements as the list of language codes (if there are different names for at least two language codes).

`LocalVariants`

possibly empty list of local variants of the names; If empty than no local variants are defined for this subdivision. Otherwise the rules are the same as for 'iso3166names'.

`DisplayName`

A name derived from Unicode CLDR data based on the sessions current locale.

5.7.1.30 `AddressType`

Synopsis:

```
enum AddressType {
      SoldTo,
      ShipTo,
      BillTo,
      Payer,
      Carrier,
      Supplier,
      EndUser,
      InstallationCompany,
      InstallationLocation,
      Branch,
      InCharge
}
```

5.7.1.31 `CommAddrType`

Synopsis:

```
enum CommAddrType {
      Phone,
      Fax,
      Mobile,
      WWW,
      Email
}
```

5.7.1.32 `TextFormat`

Synopsis:

```
enum TextFormat {
       None,
       IML,
       XML,
       JSON,
       HTML
}
```

5.7.1.33 `ScopeInfo`

Synopsis:

```
enum ScopeInfo {
      Business,
      Private
}
```

5.7.1.34 `ContactType`

Synopsis:

```
enum ContactType {
      Sale,
      Warehouse,
      Installer,
      Support,
      Employee,
      Client
}
```

5.7.1.35 `ProjectAttribute`

Synopsis:

```
enum ProjectAttribute {
      CreateTime,
      LastModifiedTime,
      ProjectState,
      ProjectName,
      ProjectNumber,
      Description,
      ExternalReferenceNumber,
      ExternalReferenceText,
      CustomerNumber,
      ExternalCustomerId,
      CustomerRelatedRemarks,
      Company,
      GeneralAgreementNumber,
      Keywords,
      ProjectDate,
      ValidToDate,
      CreatedBy,
      LastModifiedBy,
      AddressNumber,
      AddressId,
      Title,
      Name1,
      Name2,
      Name3,
      Name4,
      Street,
      Street2,
      CountryCode,
      PostalCode,
      Location,
      District,
      RegionCode,
      PoBox,
      TaxCode,
      TaxCodeEU,
      TaxCodeUSA,
      ContactNumber,
      ContactTitle,
      ContactFirstName,
      ContactLastName
}
```

Enumeration values 'CreatedBy' and 'LastModifiedBy' allow filtering by user who created a project, or by the last user who modified a project.

The data type of both new attributes is 'String', i.e. the second operand in comparisons involving these attributes must be stored in an instance of complex type 'ProjectFilterStringValue'.

The user name of the unauthenticated user is an empty string, not NULL. Thus, looking for projects created by an unauthenticated user must be done with "created_by = """ instead of "created_by IS UNDEF" (OCF-Console syntax).

5.7.1.36 `LockingMode`

Synopsis:

```
enum LockingMode {
      Default,
      NoLock,
      TryLock,
      Lock,
      ForceLock
}
```

5.7.1.37 `ProjectFilterOperator`

Synopsis:

```
enum ProjectFilterOperator {
      IsDefined,
      IsUndefined,
      And,
      Or,
      Not,
      Equal,
      NotEqual,
      Distinct,
      NotDistinct,
      Less,
      LessEqual,
      Greater,
      GreaterEqual,
      Between,
      NotBetween,
      Like,
      NotLike
}
```

5.7.1.38 `AuthMessageType`

Synopsis:

```
enum AuthMessageType {
      SCRAM_SHA_256_first,
      SCRAM_SHA_256_final
}
```

5.7.1.39 `ProjectState`

Synopsis:

```
enum ProjectState {
      Undefined,
      InProgress,
      Offered,
      Ordered,
      NotCommissioned,
      ApprovalRequired,
      WaitingForApproval,
      ApprovalDenied,
      Approved
}
```

5.7.2 Faults

Synopsis:

```
void ProjectServiceFault(
      string message
)
```

5.7.3 Operations

5.7.3.1 `saveProject`

Synopsis:

```
string saveProject(SessionId sessionId, SaveProjectOptions options)
        throws ProjectServiceFault;
```

Operation `saveProject` fails now if:

The project has not been loaded from the project store and the project store already contains a project with the same project ID.

The project apparently has been loaded from the project store, but the project has not been found in the store (should not happen unless the project has been deleted from the store).

The project has been loaded from the project store and the version that has been loaded is not the most recent version of the project. This may happen if the project, when loaded, was not the most recent version, or meanwhile some other session has saved a more recent version.

5.7.3.2 `listProjects`

Synopsis:

```
string listProjects(SessionId sessionId,
                   ListProjectsOptions options)
        throws ProjectServiceFault;
```

5.7.3.3 `loadProject`

Synopsis:

```
string loadProject(
      SessionId sessionId,
      string uri,
      LockingMode? lockMode,
)
        throws ProjectServiceFault;
```

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. When invoked with a session context ID, the operation sets the project key of the session context to the key of the new or loaded project, but never close the session context's old 'current project', regardless of whether it had been identified by a project key stored in the session context or, in the absence of such a key, was (and remains) the session's current project.

On the other hand, if these operations are invoked with a session ID, and if the new project can be created or loaded successfully, then the new project will always become the session's current project, and the old current project is always closed, regardless of whether there is a session context referencing this project. If there is such a session context, operations invoked for this session context will subsequently fail if they require a valid project until the project key of the session context has been reset or is set to the key of one of the projects currently held open by the session.

`lockMode`

The default value of this argument is 'Default'.

5.7.3.4 `closeProjects`

Synopsis:

```
void closeProjects(
                  SessionId sessionId,
                  UUID[]    projectKeys)
      throws ProjectServiceFault;
```

Closes all the projects specified as the (possibly empty) list of project keys.

The operation fails if the list of project keys contains duplicates, contains the key of the session's current project, or contains an invalid key (i.e. a key which doesn't match one of the keys of the session's current set of open projects).

5.7.3.5 `getProjectKeys`

Synopsis:

```
UUID[] getProjectKeys(SessionId sessionId)
      throws ProjectServiceFault;
```

Get the non-empty list of all projects currently held open by the session.

The first returned project key is the key of the session's current project (i.e. the key of the project used by operations invoked with a session ID, or invoked with the ID of a session context whose project key is unspecified). The order of the remaining project keys, if any, is unspecified.

In theory, the first element may be the NIL-UUID (represented by an empty string). This happens if the session has no current project, a state which should not be achievable with the current web service API, and is difficult but not impossible to achieve with the internal API used by plugins.

5.7.3.6 `setCurrentProject`

Synopsis:

```
void setCurrentProject(SessionId sessionId,
            UUID      projectKey,
            boolean?  replaceOnly)
      throws ProjectServiceFault;
```

Set the session's current project to the project specified by the given project key, and close the old current project unless both projects are the same (instances in memory) or argument `replaceOnly` has been specified as true.

Argument `projectKey` must reference one of the projects currently held open by the session.

Argument `replaceOnly`, if not specified, defaults to `false` (i.e. the old current project, if different from the new one, is not only replaced, but also closed).

5.7.3.7 `setProjectData`

Synopsis:

```
     void setProjectData(
            SessionId   sessionId,
            ProjectData data
)
```

5.7.3.8 `SetProjectText`

```
      void setProjectText(
                  string sessionId,
                  ProjectText text,
                  boolean update
      )
```

The operation `setProjectText` updates or replaces translations for a single project text type.

If parameter `update` is false then all translations for the given project text type are removed before new translations are set.

The operation is atomic. If one of the new project texts uses an unsupported format (`None` and `HTML`) or does not conform to its supposed format, the operation has no effect on the project.

5.7.3.9 `addPriceList`

Synopsis:

```
void addPriceList(
      string sessionId,
      string id,
      string startup,
      string displayName
)
```

5.7.3.10 `addProjectGroup`

Synopsis:

```
void addProjectGroup(
      string sessionId,
      string id,
      boolean folder,
      string simpleName,
      string subgroupTitle
)
```

5.7.3.11 `addProjectNumberScheme`

Synopsis:

```
void addProjectNumberScheme(
      string sessionId,
      string id,
      string pattern,
      long minValue,
      long maxValue
)
```

The minimum value must be greater than or equal to zero, the maximum value must be greater than or equal to the minimum value, and the next value must be between minimum and maximum value (both inclusive). If no minimum value and/or no maximum value is passed to operation addProjectNumberScheme then the minimum value defaults to 1 and the maximum value defaults to (2^63)-1.

5.7.3.12 `authenticateUser`

Synopsis:

```
string authenticateUser(
      string sessionId,
      AuthMessageType messageType,
      string message
)
```

5.7.3.13 `deleteProjects`

Synopsis:

```
string[] deleteProjects(
      string sessionId,
      string[] uris,
      DeleteProjectsOptions options
)
```

The `deleteProjects` operation takes a set of project (version) URIs and options. The URIs must be valid URIs with scheme `pst`. They may or may not contain a version parameter. If the version parameter is present the URI references that particular version (unless option `ignoreVersionNumbers` is set to true, in which case all version numbers in URIs are ignored).

If the set of URIs contains an URI without a version parameter, then all URIs referencing a particular version of the same project are ignored.

If the set of URIs contains an URI without a version parameter, or contains URIs for all versions of the same project, the whole project is deleted.

Deleting a project or project version deletes all database records and all associated OBK files.

The operation behaves atomically. If it fails with an exception than no projects or project versions have been deleted.

The following options are supported:

`ignoreVersionNumbers` (default: `false`) All version numbers in URIs passed to the operation are ignored. Multiple URIs for the same project but with different version numbers do not cause an error.

`failIfNonExistent` (default: `true`) Normally, the operation fails if one of the specified projects or project versions does not exist. If this option is set to false then requests to delete non-existent projects and project versions are silently ignored.

`failIfPermissionDenied` (default: `true`) Normally, the operation fails if the user authenticated for the current session does not have permission to delete one of the specified projects or project versions. If this option is set to false then these projects or project versions are not deleted, but projects and project versions the user has permission to delete are still deleted.

`failIfNotLockable` (default: `true`) Normally, the operation fails if one of the projects to delete, or one of the projects whose version is to be deleted, is currently ocked. If this option is set to false then these projects or project versions are not deleted, but projects that are not locked, and project versions whose project is not locked, are still deleted.

`ignoreLocks` (default: `false`) If this option is set to true then projects and project versions are deleted even if they are currently locked.

5.7.3.14 `getAddressData`

Synopsis:

```
AddressData[] getAddressData(
string sessionId,
AddressType[] addressTypes
)
```

5.7.3.15 `getContactData`

Synopsis:

```
ContactData[] getContactData(
      string sessionId
)
```

5.7.3.16 `getCountries`

Synopsis:

```
CountryData[] getCountries(
      string sessionId
)
```

5.7.3.17 `getCountrySubdivisions`

Synopsis:

```
SubdivisionData[] getCountrySubdivisions(
        string sessionId,
        string countryCode
)
```

5.7.3.18 `getPriceLists`

Synopsis:

```
PriceList[] getPriceLists(
        string sessionId,
        string idPattern
)
```

5.7.3.19 `getProjectData`

Synopsis:

```
    ProjectData getProjectData(
            string      sessionId,
            TextFormat textFormat
)
```

If the argument is present, and has a value other than `None`, the returned instance of ProjectData contains a possibly empty element projectTexts, containing all project texts converted to the specified format.

Allow use of operation setProjectData of project service to set all project texts.

If the instance of ProjectData contains the element `projectTexts`, then all current project texts are removed and the specified project texts are set in order of appearance.

The operation is atomic. If one of the new project texts uses an unsupported format (`None` and `HTML`) or does not conform to its supposed format, the operation has no effect on the project.

5.7.3.20 `getProjectGroups`

Synopsis:

```
ProjectGroup[] getProjectGroups(
        string sessionId,
        string idPattern
)
```

5.7.3.21 `getProjectNumberSchemes`

Synopsis:

```
ProjectNumberScheme[] getProjectNumberSchemes(
        string sessionId,
        string idPattern
)
```

5.7.3.22 `getProjectTexts`

Synopsis:

```
ProjectText[] getProjectTexts(
              string sessionId,
              ProjectTextType[] textTypes,
                    TextFormat textFormat
      )
```

The operation returns all available translations for the given project text types converted to the given format.

If `textTypes` is empty then all known project text types are used instead.

All values of TextFormat are supported except `None`.

An instance of ProjectText is returned for text types that have at least one translation defined.

5.7.3.23 `listCompletions`

Synopsis:

```
      string[] listCompletions(
            string sessionId,
            ListCompletionsOptions options
      )
```

For now this method can only be used to list filter completions, but it will be extended to list completions of search terms.

5.7.3.24 `newProject`

Synopsis:

```
      void newProject(
            string sessionId,
            string projectGroup,
            string priceList
      )
```

Instead of a session ID the operation can also be invoked with a session context ID as their first argument (see also §5.4.3.5). If so, the session ID stored in the session context will be used as the actual session ID. When invoked with a session context ID, the operation sets the project key of the session context to the key of the new or loaded project, but never close the session context's old 'current project', regardless of whether it had been identified by a project key stored in the session context or, in the absence of such a key, was (and remains) the session's current project.

On the other hand, if these operations are invoked with a session ID, and if the new project can be created or loaded successfully, then the new project will always become the session's current project, and the old current project is always closed, regardless of whether there is a session context referencing this project. If there is such a session context, operations invoked for this session context will subsequently fail if they require a valid project until the project key of the session context has been reset or is set to the key of one of the projects currently held open by the session.

The price list, if specified, must be available for the project group. If not specified, the project group must either have no associated price list, in which case the project continues to use the session's current configuration, or exactly one price list. If a price list has been determined for use, and the price list specifies a startup file, then the current session configuration is replaced with the configuration read from the startup file and the new project will use this configuration. Otherwise (price list specifies no startup file) the new project continues to use the current session configuration.

Loading a new session configuration and replacing the current project with a new empty project is an atomic operation (except in case of errors caused by bugs in the implementation).

For a given project group and price list. The operation fails without further action if:

- The specified project group does not exist
- No price list has been specified and the project group supports multiple price lists

- A price list has been specified, but the project group does not have the specified price list
- The project group has been configured for automatic generation of project numbers and generation of a new project number fails for whatever reason.

5.7.3.25 `removeAddressData`

Synopsis:

```
AddressType[] removeAddressData(
      string sessionId,
      AddressType[] addressTypes
)
```

5.7.3.26 `removeContactData`

Synopsis:

```
int[] removeContactData(
      string sessionId,
      AddressType addressType,
      int[] contactIds
)
```

5.7.3.27 `removePriceList`

Synopsis:

```
void removePriceList(
      string sessionId,
      string id
)
```

5.7.3.28 `removeProjectGroup`

Synopsis:

```
void removeProjectGroup(
      string sessionId,
      string id
)
```

5.7.3.29 `removeProjectNumberScheme`

Synopsis:

```
void removeProjectNumberScheme(
      string sessionId,
      string id
)
```

5.7.3.30 `setAddressData`

Synopsis:

```
void setAddressData(
      string sessionId,
      AddressData address
)
```

5.7.3.31 setAutoSaveOptions

Synopsis:

```
AutoSaveOptions setAutoSaveOptions(
      string sessionId,
      AutoSaveOptions? options
)
```

Sets auto-save options for projects loaded from and saved to the project store (newly created projects are not auto-saved until they have been saved for the first time).

Project versions saved by auto-save are marked as such and are automatically deleted the next time the project is saved.

Auto-save should usually be configured in the session startup file, but can be changed afterwards on a per-session basis using the project store method `setAutoSaveOptions()`.

Options: Argument and all the options specified within the `options` argument are optional. The return value represents the current set of options (i.e. the previous set of options with the new options, if any, applied).

5.7.3.32 setContactData

Synopsis:

```
void setContactData(
      string sessionId,
      ContactData contact
)
```

5.7.3.33 updatePriceList

Synopsis:

```
void updatePriceList(
      string sessionId,
      PriceList priceList
)
```

5.7.3.34 updateProjectGroup

Synopsis:

```
void updateProjectGroup(
      string sessionId,
      ProjectGroup projectGroup
)
```

Special handling of numberScheme and priceLists by operation updateProjectGroup:

To set/reset the number scheme of a project group, use element `<numberScheme>` with attribute `id` set to the id of the number scheme to use (or to an empty string if the number scheme is to be reset). All other attributes and nested elements are ignored.

Similarly, to set the price lists of a project group, use elements `<priceList>` with attribute `id` set to a price list ID. Again, all other attributes and nested elements are ignored.

5.7.3.35 `updateProjectLock`

Synopsis:

```
boolean updateProjectLock(
      string sessionId,
      LockingMode lockMode
)
```

Used to update the lock of the session's current project. The actual operation and the meaning of the return values depends on the locking mode:

`Default` - query the state of the lock, i.e. whether or not the project holds the lock; returns true if the project holds the lock

`NoLock` - release the lock if it is held by the project; returns true if the project previously held the lock

`TryLock` - try to lock the project; returns true if the lock could be acquired (or was already held by the project), false otherwise

`Lock` - like TryLock, but fails with an exception instead of returning false

`ForceLock` - always lock the project, stealing the lock from the current lock holder if necessary

Note that the return value does reflect the state of the lock at some point during the execution of the operation. In particular, a return value of true in case of *Lock does not necessarily that the project still holds the lock when the operation returns (someone else may have stolen it in the meantime).

Increased project lock timeout from 5 to 10 minutes, and refresh timeout from 3 to 5 minutes.

5.7.3.36 `updateProjectNumberScheme`

Synopsis:

```
void updateProjectNumberScheme(
      string sessionId,
      ProjectNumberScheme numberScheme
)
```

6 Statistic Event Manager

With the statistics event manager in EAIWS statistic information can be sent to a HTTP client. The client can then for example save the events to a database and do statistical evaluations. To configure the client please see §4.1.28.

Basically, the event manager receives statistics events from the application and simply attempts to feed them into a bounded blocking queue limited to at most 1024 entries. (The queue actually stores groups of events, so it may contain more than 1024 events if events are reported in groups, as often happens during property changes of composite articles).

If the queue fails to accept new events because its capacity has already been reached, the events are discarded, and an error message is eventually logged (after 1000 discarded events or 10 minutes since the last error message of that kind).

A separate thread takes events from that queue and delivers them to local event queues and the HTTP client event queue (if configured, see §4.1.28). One of these queues discarding events due to overflow has no effect on other queues.

The statistics framework defines two (abstract) base classes for events:

Synopsis:

```
[ uuid(d108dd08-1e07-3ca0-9202-615aab9e63df), extensible ]
struct StatsEvent
{
    Instant timestamp@1;
}


[ uuid(61ca0e23-6f68-3af9-a686-d775b56074e9), extensible ]
struct SessionStatsEvent : StatsEvent
{
    UUID sessionId@10;
}
```

All statistics events *must* be derived from `StatsEvent`. All session-specific events *should* be derived from `SessionStatsEvent`.

The statistics framework defines three events related to the management of sessions:

Synopsis:

```
[ uuid(4fd35df3-f9a0-48fd-8da5-70994b2f9985), extensible ]
struct SessionOpenEvent : SessionStatsEvent
{
    String localeName@100;
    String zoneId@101;
}


[ uuid(e06c9897-699b-4216-9d8f-1a5a5d95eada), extensible ]
struct SessionCloseEvent : SessionStatsEvent
{
}


[ uuid(3e4df265-391d-491a-98a4-30ec4745ef16), extensible ]
struct SessionConfigEvent : SessionStatsEvent
{
    String? localeName@100;
    String? ZoneId@101;
    }
```

Open and close events are emitted whenever a session is opened or closed. Configuration events result from operation `setLocale` (§5.4.3.10) of the session web service.

The statistics framework defines five events related to the insertion, deletion and configuration of basket articles.

Synopsis:

```
[ uuid(75b076cf-10b9-3bc8-b441-142f73af878c), extensible ]
struct BasketStatsEvent : egr.eai.fw.statistics.SessionStatsEvent
{
}


[ uuid(901b332b-c7d6-3c5c-bd0e-b914bb99faed), extensible ]
struct BasketArticleEvent : BasketStatsEvent
{
    String manufacturerId@100;
    String seriesId@101;
    String baseArticleNumber@102;
    String distributionRegion@103;
    String finalArticleNumber@104;
    String ofmlVariantCode@105;
    Boolean isSubArticle@106;
}


[ uuid(e94a83b6-5cfd-4f00-80cd-0da52752b35a), extensible ]
struct InsertArticleEvent : BasketArticleEvent
{
}


[ uuid(024ace31-0c50-4acf-908e-3968db66b47f), extensible ]
struct DeleteArticleEvent : BasketArticleEvent
{
}


[ uuid(f8a453f6-b616-35b2-a41a-ebedc1385009) ]
struct PropertyChange
{
    String propertyName@1;
    GDRValue oldValue@2;
    GDRValue newValue@3;
}


[ uuid(949ae94c-7358-40ee-835f-d964e1d09e6d), extensible ]
struct ConfigArticleEvent : BasketArticleEvent
{
    String? oldManufacturerId@200;
    String? oldSeriesId@201;
    String? oldBaseArticleNumber@202;
    String? oldDistributionRegion@203;
    List<PropertyChange> propertyChanges@204;
}
```

`BasketStatsEvent` is the abstract base class of all events emitted by the basket module.

`BasketArticleEvent` is the abstract base class of all article-related basket events.

Events of type `InsertArticleEvent` are emitted whenever a new OFML article is inserted by operation `insertOFMLArticle` (§5.6.3.16) of the basket web service, as a result of the `CreateObject` OAP action, or as a result of a property change (see below).

Events of type `DeleteArticleEvent` are emitted as a result of a property change.

Events of type `ConfigArticleEvent` are emitted as a result of a property change.

Property changes are usually triggered by

- operation `setPropertyValue` (§5.6.3.39) of the basket web service
- operation `updateBasketArticles` (§5.6.3.80) of the basket web service
- OAP actions `PropChange` (§5.6.1.105), `MethodCall` (§5.6.1.139), `TranslateObject` (§5.6.1.58), and `RotateObject` (§5.6.1.57).

Each property change compares the configurations of the main article and the set of sub-articles of the affected composite article before and after the property change. Articles that no longer exist (according to their item ID) are put in the set of deleted articles, articles that did not exist before are put in the set of inserted articles, and articles that did exist before and still exist, but differ in manufacturer ID, series ID, base article number, distribution region, final article number, OFMLVarCode, and/or PropVarCode a put in the set of changed articles.

Changed article items always result in a `ConfigArticleEvent`. Fields `oldManufacturerId`, `oldSeriesId`, `oldBaseArticleNumber`, and `oldDistributionRegion` are set only if the respective values differ before and after the property change. If manufacturer ID, series ID and base article number have not changed, the old and current PropVarCodes are compared and changed properties are stored with their old and new values in `propertyChanges` (added or removed property values are not reported).

The sets of inserted and deleted sub-articles are then compared to remove pairs of identical articles which have been both inserted and deleted. Furthermore, pairs of inserted and deleted articles which are identical except for their PropVarCode are removed from the sets of inserted and deleted sub-articles but result in a `ConfigArticleEvent` (necessary as insert and delete events don't include the PropVarCode). Finally, `InsertArticleEvent` and `DeleteArticleEvent` are generated for articles remaining in the sets of inserted and deleted articles.

7 SSL Setup

7.1.1 HTTPS Support

Please note: The EAIWS supports only HTTP or HTTPS per domain. Mixed operation is not possible.

The actual HTTPS handling must be performed by an reverse proxy. HTTP is used for communication between EAIWS and reverse proxy. The reverse proxy has to send the x-forwarded-proto header with the value of https to the EAIWS.

7.1.2 Soap connection via WSDL

The technologies used for the EAIWS do not allow to change the protocol used in the WSDL at runtime. If the client has to parse the WSDL, an EAIWS plug-in is required. This plug-in provides all WSDL endpoints via an alternative URL with corrected links to the actual SOAP service.

The url prefix for this is: /EAIWS/WSDL/

This means that the WSDL must be requested via the plug-in URL /EAIWS/WSDL/EAI/Session?WSDL (instead of /EAI/Session?WSDL).

Please contact EasternGraphics in case the plug-in is required