# Data validation

Automatic data validation process

The automatic data validation process refers to the first validation of incoming data. The positive result of the automatic validation process places the file in a valid state and makes it available for submission.

The data validation process is a combination of a cascade of validation steps:

1. inbound file format structure validation (e.g. XML files verified through XML schema validation);
2. metadata validation (e.g. validation of the data to ensure that it is loadable on the target database tables, the validation of the correct catalogues or domain-specific hierarchies and the proper use of the custom data elements available in the compound elements);
3. business rules validation (more complex validation verifying logical implications and the connection between values reported in different tables).

Inbound file

File format structure validation

Metadata validation

BR validation

Valid data

1. Steps of the automatic validation process

These three validation steps have to be defined to allow data providers to submit their data successfully.

The configuration of the three validation steps is performed following a top-down approach, starting from a general ‘subject-specific’ model and narrowing it down to a specific domain and then to a specific instance of data collection. This approach is described in Figure 26.

* Subject level. Contains:
* the metadata definition with all elements of the model with their relevant data type;
* the attributes of the compound elements supported by the data structure with their associated multiplicity. The definition focuses on whether the attributes are mandatory or optional;
* the validation rules that apply to the entire data model;
* the file formats supported by the general data model.
* Data domain level. Contains:
* restriction in the use of compound elements (e.g. reduction of multiplicity of an attribute from repeatable to single or change of requirements from optional to mandatory elements);
* additional domain validation rules;
* restriction on the use of some file formats for a specific domain (based on data collection configuration).
* Data collection instance (DCI). Further restriction on the compound element attributes, additional validation rules and file format restrictions, similar to data domain levels.

The data collection metadata validation provides a cascade of restriction to the incoming data using the diagram of Figure 27.

Subject level

Data Domain level

Data Collection Instance (DCI)/Segment

1. Metadata validation cascade restrictions

For example, the SSD provides the harmonised reference model for the subject of analytical laboratory results. This model can be redefined in the domain of chemical occurrence to be used for different data collection instances for different reporting years. The scope of the subject level is to provide the common data model to be used to attach the extract–transform–load (ETL) process to migrate the data into the EFSA DWH. The levels beyond (data domain and data collection) provide validation specific to the individual data domains and collections.

The definition of the metadata of each table within a data collection should be made available for downloading from the receiver system (the structures of the resource files containing the data collection configuration and the definition of a table are detailed in section 11).

Frequency of review

The changes to the data collection metadata validations as defined above should be performed under the supervision of the relevant networks that use this guidance. This process shall be extended to networks which are proposing to apply the SSD2 and/or GDE2 to new domains.

A maintenance process is needed to allow the:

* addition and/or deprecation of metadata definitions applicable at the subject level/domain level or data collection level;
* addition and/or deprecation of attributes of compound elements;
* addition of and/or changes to business rule definitions.

The frequency and timing of such updates are of particular concern to data providers, since it is not always feasible to amend reporting requirements once the data have been collected. Ideally, changes will be relatively infrequent, enabling the data collection metadata validation definitions to be in use at the time of capture. In principle, before including the changes, a review of the data collection metadata validation definitions should be performed by the relevant networks.

It is recommended that the implementation schedule for data collection metadata validation definition changes should be mainly annual (*major release*) and take into account the ability of MSs to implement such changes. In addition, it is anticipated that each data collection domain will need to make changes at a time of year which best coincides with their data collection schedules. In several instances, these annual schedules are well established within the relevant networks and a draft annual schedule is proposed below in Table 58.

Each network will collate their comments and consider the requests that have an impact on their specific domain.

1. Proposed annual plan for the update of the data collection metadata validation definitions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data collection domain | Annual reporting deadline for data submission | Deadline for terms to include in next release | Deadline for discussion within the network | Submission of updates from the network | Submission of updated version for final comments | Deadline for SSD revision publication |
| Biological monitoring | 31st May |  |  |  |  |  |
| Pesticides | 31st August |  |  |  |  |  |
| Chemical contaminants | 1st October | 15th October | Up to the networks | No later than 30th November | 31st January | 28th February |
| Additives | Ad hoc basis |  |  |  |  |  |
| Food contact materials | Ad hoc basis |  |  |  |  |  |

EFSA should set up an internal data collection co-ordination group to discuss the metadata validation definition amendments so that decisions are always taken in an harmonised way, taking into account all the requirements and implications in the different data collection domains. When necessary, this group may involve additional experts from the MSs or data providers.

Before the new annual version of the data collection metadata validation definitions is released, the collated suggestions should be circulated to the relevant networks for their comments and approval. In order to streamline the process, the networks could establish a specific subgroup to deal with these requests. Once comments are gathered, the EFSA data collection co-ordination group should make the final decision and publish the updated data collection-specific metadata validation rules.

In addition, it is anticipated that there may be instances where unplanned calls for data are launched which require specification of ad hoc urgent data collections. In this case, preliminary modifications to the data collection metadata validation definitions can be made. This type of update should be kept to a minimum owing to the impact that they may cause on the data providers. The addition of preliminary modifications to data collection metadata validation rules may also require the publication of additional ‘minor releases’ of the controlled terminology catalogues during the year. This process would also be controlled by the EFSA data collection co-ordination group.

Validation against business rules

Business rules are used to check the usability of reported data. This method is a validation procedure that can be automatically applied in a data collection system when a file has been transmitted. The term ‘business rule’ is the preferred term instead of ‘validation rule’, although the latter term is used frequently in the literature. One reason is that some quality checks are related to a specific business only. A criterion might apply to one data collection but not to another. The other reason is to avoid confusion. When an XML file is ‘valid’, this means that it complies with the schema of the data model. However, a valid XML file does not necessarily comply with the business rules of a data collection.

The business rules can generate an error or a warning. Transmissions containing only warnings should be accepted by the receiver, whereas transmissions containing one or more errors should be rejected.

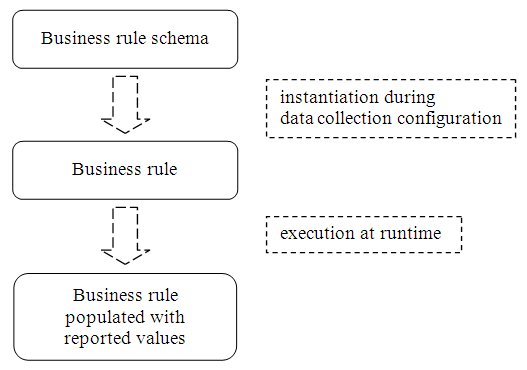
As described in section 10.1 above, the following general criteria must be fulfilled in addition to the business rules for data collection:

* When a data element is mandatory in a data model, a value must be reported (file format structure validation).
* The type of reported value must match the specification (metadata validation).
* If a value is supposed to contain a code of a catalogue, then a reported code must exist, be active and be allowed for the data element.

Logical structure of business rules

When data are validated by the system, the submitted values are compared with predefined values. These comparisons follow similar structures, which makes it possible to define patterns that need only to be configured. For each data collection, data managers instantiate the business rule template in accordance with given scientific specifications. When a file is transmitted to the data collection system, these instantiated business rules are populated automatically with the reported values and then executed. Figure 28 illustrates this logical structure of business rules.

This approach of using a generic template speeds up the development process of business rules, since the set up becomes merely a configuration of a particular data collection. Moreover, the business rules of a data collection can be easily distributed to the reporting countries that might also want to implement them in their reporting system. The business rule files that shall be applied to a specific data collection are not directly executable but can be seen as configuration files for a program. They shall be retrievable from the data collection system.



1. Logical structure of business rules

General notes

The specified syntax for the business rules is a trade-off to meet the following requirements:

* Usability: It must be possible to express the criteria that have to be met by the reported data in words. Furthermore, how to set up the business rules and how to implement them in a data collection system must be clear for data managers.
* Flexibility: With respect to the diversity of all data collections that are currently processed in the data collection system, or will be in the future, a certain amount of flexibility is needed to ensure the checking of more sophisticated conditions. However, this implies a higher degree of complexity in the syntax.
* Parsability: A computer program must be able to parse and to interpret unambiguously the business rules in order to populate and execute them with the reported values.

The business rules shall be implemented in XML. The structure of the business rules (template) is defined in the XML schema ‘businessRule.xsd’. All enumerated values needed in this schema are maintained in a separate XML schema called ‘businessRulesEnumerations.xsd’, which is included in the ‘businessRule.xsd’ schema. This approach allows easy extensibility for the lists of values without changing the main document, ‘businessRule.xsd’. The following sections explain how to use the defined structures and lists of values in order to set up valid business rules.

Please note that, for one data collection, more than one XML file with business rules can be defined. For example, one file could contain general checks that are applicable to all data collections that use the same data model, such as the validation of dates. A second file could contain business rules that apply to only the specific data collection.

Although a thorough investigation of the requirements has been performed and examples have been implemented based on the business rules currently in place as proof of concept, this specification will be revised if additional requests arise.

## Syntax of business rules

### businessRuleSet

The root node in the business rules XML file is called *businessRuleSet* (see Figure 29). It contains the reference to the schema ‘businessRules.xsd’, the element *name* to describe the content of the file, and one or many *businessRule* elements wrapped in a *businessRulesList*. Each *businessRule* element contains the specification of exactly one business rule.

Example of basic structure of an XML file containing business rules

<?xml version=“1.0” encoding=“UTF-8”?>

<businessRuleSet

xmlns:fn=<http://www.w3.org/2005/xpath-functions>

xmlns:xs=<http://www.w3.org/2001/XMLSchema>

xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance> xsi:noNamespaceSchemaLocation=“businessRules.xsd”>

<name>Here goes a descriptive name for the business rule set (max. 100 characters)</name>

<businessRulesList>

<businessRule> […] </businessRule>

…

<businessRule> […] </businessRule>

### businessRule

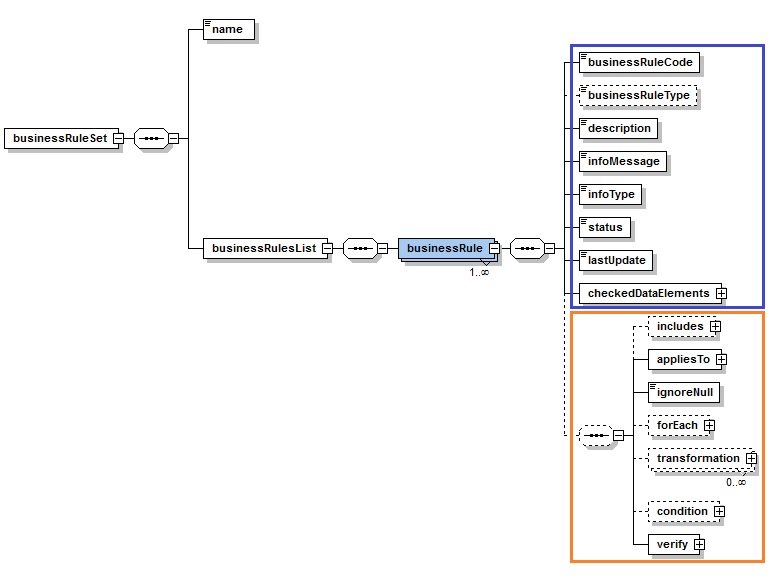
Each business rule is defined within a *businessRule* element. Logically, a business rule definition is made up of two parts: the first part is the specification from a scientific point of view (blue box in Figure 29) and the second part is the technical specification (orange box in Figure 29). In order to set up a valid business rule, the following elements must be provided within a *businessRule* element: *businessRuleCode*, *description*, *infoMessage*, *infoType*, *status*, *lastUpdate* and *checkedDataElements*.

These mandatory elements form, together with the optional element *businessRuleType*, the scientific specification. This describes what should be considered when data are prepared for transmission, but it does not describe how the business rule can be implemented for automatic validation. This is done in the technical specification.

The technical specification is a sequence of the following elements: *includes*, *appliesTo*, *ignoreNull*, *forEach*, *transformation*, *condition* and *verify*. The technical specification is optional as it is possible that complex checks can not be implemented within the business rule syntax described in this document, but they require an implementation that will be handled through a call to an external engine/software (see section 10.4.10).

If the technical specification is given, the elements *appliesTo*, *ignoreNull* and *verify* are mandatory. In the *verify* element, the criteria that should be verified by the business rule are specified.

When a business rule is set up, it is important to outline the scope which shall be taken into account when this rule is executed. A business rule can be applied to a single record or to a set of records. For example, whether or not the reported sampling area is within the reported sampling country can be checked within one record. However, when more than one record for one sample is reported, it is clear that, in all these records, the same sampling information must be provided. To check this, it is necessary to sub-select all records of that sample. A subset of records can be determined if records have the same values in certain data elements; in the example given, that would be the unique identifier of the sample. The elements *includes*, *appliesTo* and *forEach* provide the functionality to select the appropriate subset of records in which the criteria should be verified. The *condition* element allows a certain criterion to be met only in a certain condition. Only if the condition is true will the criterion be verified. Within the *transformation* element, it is possible to transform the submitted dataset so that it is suitable for the business rule check.



1. The element *businessRuleSet* contains a *name* element and one to many *businessRule* elements wrapped in a *businessRulesList*. The blue box contains the elements for the scientific specification; the orange box contains the elements for the technical specification

### businessRuleCode

The element *businessRuleCode* contains the unique identification code of the specific business rule. In the acknowledgement message, it is used as a reference to the business rule that has been violated in a submitted dataset. It can have a maximum length of 100 characters.

### description

The element *description* describes the scientific requirement that needs to be fulfilled by the reported data. The text must comply with the requirements of well-formed XML files (e.g. the symbols < and & are reserved for markup). It can have a maximum length of 2 000 characters.

### infoMessage

The element *infoMessage* contains the text that is returned to the user when the business rule check fails. The text must comply with the requirements of well-formed XML files (e.g. the symbols < and & are reserved for markup). It can have a maximum length of 255 characters.

### infoType

The element *infoType* indicates the type of business rule, which can be either an ‘error’ or a ‘warning’. Business rules with *infoType* ‘error’ are mandatory data quality requirements. A dataset that violates a business rule of this type will be rejected by the data collection system. Business rules with *infoType* ‘warning’ are recommended data quality requirements. A dataset will be accepted with warnings by the data collection system, although a business rule of this type is not fulfilled (see Table 59).

1. Info types

|  |  |
| --- | --- |
| *infoType* | Description |
| Error | All data must comply with this business rule, otherwise the dataset will be rejected |
| Warning | The data should be revised and possibly improved but the dataset will be accepted |

### status

Usually a business rule has the status ‘active’ in the element *status*. When a business rule is deprecated, it can be set to ‘inactive’, meaning that it should no longer be applied. In this way, all business rules can be kept in the same file and need not be archived. The status ‘pending’ is preliminary to the status ‘active’ and can be used when a new business rule is still under development (see Table 60).

1. Status of a business rule

|  |  |
| --- | --- |
| *Status* | Description |
| Active | The business rule shall be applied |
| Inactive | The business rule was applied in the past but should no longer be applied |
| Pending | For maintenance: the business rule has not been approved yet, and shall not be applied |

### lastUpdate

The element *lastUpdate* contains the date when the business rule was last amended. Its type is xs:date, taking the date in the format YYYY-MM-DD.

### checkedDataElements

The element *checkedDataElements* refers to the data elements that are checked by the business rule. As shown in Figure 30, the element *checkedDataElements* can contain one or more *dataElement* elements. If a business rule is violated, this is a good starting point for efforts to rectify the data. Nevertheless, it is possible that another correction needs to be made in some other data element.



1. Element *checkedDataElements*

Example of the technical specification part containing mandatory elements of a business rule

<businessRule>

<businessRuleCode>example2</businessRuleCode>

<description>Example of a business rule with mandatory elements;</description>

<infoMessage>A result code should be provided;</infoMessage>

<infoType>error</infoType>

<status>pending</status>

<lastUpdate>2016-07-31</lastUpdate>

<checkedDataElements>

<dataElement>resultCode</dataElement>

</checkedDataElements>

</businessRule>

### businessRuleType

The element *businessRuleType* is meant to be used when the business rule is validated by an external engine/software (e.g. validation of molecular typing data based on Bionumerics) rather than the automatic validation of the data collection system. In this case, only the scientific specification should be provided and the *businessRuleType* element must contain the value ‘external’. In all the other cases, an internal validation is considered as default value and the type of business rule does not need to be specified.

### Example of business rule validated by external validation

<businessRule>

<businessRuleCode>example3</businessRuleCode>

<businessRuleType>external</businessRuleType>

<description>This business rule is validated by BioNumerics system;</description>

<infoMessage>Image not conforming to BioNumerics standard;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2016-07-31</lastUpdate>

<checkedDataElements>

<dataElement>tiffImage</dataElement>

</checkedDataElements>

</businessRule>

### dataElement and repeatableDataElement

The element *dataElement* can contain any name of a data element that is defined in the data model. As described in the SSD2 Guidance 2.0 (EFSA, 2013) in Figure 1 ‘Data element structure in SSD2’, a ‘data element’ can be a ‘simple data element’, a ‘repeatable data element’ or a ‘compound data element’.

A compound data element contains one or more simple data elements, which are called ‘base terms’, ‘attributes’ or ‘facets’. Attributes, facets and base terms within compound data elements need to be referred into the dataElement by using a dot notation:

compoundDataElementName.simpleDataElementName

The base term of a compound data element should always be referred to as ‘.base’, to distinguish it from the other facets and attributes (e.g. ‘anMatCode.base’, base term of the matrix analysed, and ‘anMatCode.ingred’, facet collecting ingredients and/or flavour note).

Repeatable data elements have the exclusive feature of being allowed to contain a single value or a list of values. For investigating a list of values in a repeatable data element, quantifiers shall be used.

If checks need to be performed on the full content of a repeatable or compound data element (i.e., to check whether the data element is null or not), then the name of the repeatable or compound field needs to be specified within a *dataElement* element.

The element *dataElement* is contained in several other elements where it always has the same meaning. Namely these are the following elements: *checkedDataElements*, *forEach*, *catalogueAttribute*, *aggregation*, *operand* and *transformationOperand*.

The element *repeatableDataElement* is used in the *quantification* element.

In the particular case in which data elements are transposed using the *transformation* element, reported values can become data elements.

### includes, include, databaseTable and dataCollection

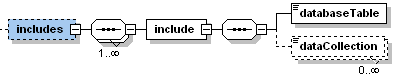
The element *includes* allows data that were collected in other data collections to be considered in the scope to which the business rule applies. In other words, the business rule can also look at records outside the default scope (database table and data collection of the current dataset).

The *includes* element should not be used when the business rule applies only to records in the same database table or data collection to which the current dataset belongs. It should only be used to include records from other database tables and data collections.

Within a database table, several data collections can be stored. When the whole additional table should be included, it is sufficient to specify its name in the element *databaseTable* as defined in the data collection system. If specific data collections from the additional table should be included, they should be specified in *dataCollection* elements. This sequence is wrapped by an *include* element. The *includes* element can incorporate more than one *include* element (see Figure 31).

There is an overlap between the system variables sysDatabaseTable and sysDataCollection in the *appliesTo* element (see section 10.4.13 below).

In cases where the element *appliesTo* is specified, the business rule shall be applied to the reporting country or organisation only (system variables sysCountry and sysOrganisation, respectively). This must also be considered when data from other data collections or tables are included using the element *includes*.



1. Element *includes*

### appliesTo and systemVariable

The element *systemVariable* can contain only the predefined values from Table 61. System variables must be used to specify the scope to which the business rule applies. One or more *systemVariable* elements must be wrapped by the *appliesTo* element (see Figure 32).



1. Element *appliesTo*
2. System variables as defined in the data collection system to specify the context of a business rule

|  |  |
| --- | --- |
| *systemVariable* | Description |
| sysRecord | The business rule applies to a record in the uploaded dataset |
| sysDataCollection | The business rule applies to the data collection to which the dataset has been uploaded |
| sysDatabaseTable | The business rule applies to the table in the database to which the dataset has been uploaded; a table can be shared among more than one data collection. If sysDatabaseTable is specified, the business rule applies to the whole table regardless the data collection |
| sysCountry | When sysDataCollection or sysDatabaseTable are specified as the system variable, it can be further specified that the business rule applies to the reporting country |
| sysOrganisation | When sysDataCollection or sysDatabaseTable are specified as the system variable, it can be further specified that the business rule applies to the reporting organisation |

Example of XML file including additional database tables and data collections of the reporting organisation

<businessRule>

<businessRuleCode>example</businessRuleCode>

[…]

<includes>

<include>

<databaseTable>FINAL\_MOPER\_11</databaseTable>

</include>

<include>

<databaseTable>MOPER</databaseTable>

<dataCollection>MOPER.PPP\_2012\_FINAL</dataCollection>

<dataCollection>MOPER.PPP\_2013\_FINAL</dataCollection>

</include>

</includes>

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysOrganisation</systemVariable>

</appliesTo>

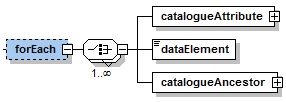
[…]

</businessRule>

### forEach

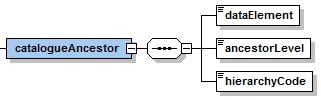
With the *forEach* element, records can be grouped depending on reported values. All records that have the same value or the same combination of values are grouped together.

At least one *dataElement* or *catalogueAttribute* or *catalogueAncestor* element must be defined in a *forEach* element (see Figure 33). A considered value can be either a reported value in the specified *dataElement* or a value in *catalogueAttribute,* meaningan attribute in a catalogue that needs to be queried from there by using the reported value in the specified data element (see section 10.4.19.2).



1. Element *forEach*

Sometimes it is in the scope of a business rule to group together records with the same parent/ancestor in a catalogue. This can be done with the *catalogueAncestor* element: the level that should be considered to aggregate values must be specified in the *ancestorLevel* element; since a catalogue can contain more than one hierarchy, the name of the hierarchy must be reported in the *hierarchyCode* element (see Figure 34).



1. Element *catalogueAncestor*

Generally, a *forEach* can only be in combination with *appliesTo* a ‘sysDataCollection’ or a ‘sysDatabaseTable’, and never with a ‘sysRecord’.

Example of records grouped together on the basis of the reported ‘sampId’

[…]

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysCountry</systemVariable>

</appliesTo>

[…]

<forEach>

<dataElement>sampId</dataElement>

</forEach>

[…]

Example of records grouped together on the basis of the level 1 (in ‘microParam’ hierarchy) of the reported ‘zoonosis’

[…]

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysCountry</systemVariable>

</appliesTo>

[…]

<forEach>

<catalogueAncestor>

<dataElement>zoonosis</dataElement>

<ancestorLevel>1</ancestorLevel>

<hierarchyCode>microParam</hierarchyCode>

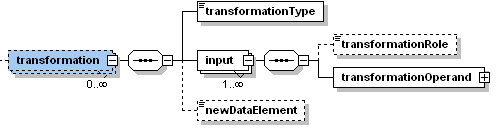
</catalogueAncestor>

</forEach>

[…]

### transformation, transformationType, input, transformationRole, transformationOperand and newDataElement

The *transformation* element (see Figure 35) can be interpreted as a function to transform reported data in order to be able to check its correctness. The structure is generic to allow the definition of further transformation types if needed in the future. In the element *transformationType*, the type of transformation is specified (see Table 62). In the *input* element, the inputs for the transformation are defined. This is done by defining the role of the input in the *transformationRole* element (see Table 63) and the operand in the *transformationOperand*. The *transformationRole* is optional in cases where no distinction between several inputs is needed. In order to reference the transformation output, a name of a new data element can be specified in *newDataElement* element.



1. Element *transformation*
2. Transformation types

|  |  |
| --- | --- |
| *transformationType* | Description |
| Transposition | Makes values reported within a data element into new data elements and associates values in another data element with these new data elements |

1. Transformation roles

|  |  |
| --- | --- |
| *transformationRole* | Description |
| transposedDataElement | Data element of which the values are transposed to new data elements |
| transposedValueDataElement | Data element of which the values become the values of the transposed data elements |

A transposition is useful when related values are not within one record but spread over several linked records. It is mandatory to specify the scope of the business rule by providing this link between the records in the *forEach* element when this kind of transformation should be performed. A transposition takes two inputs: first input has the *transformationRole* ‘transposedDataElement’ and takes a *dataElement* as *transformationOperand*; the values of this data element are transposed to new data elements. The second input has the *transformationRole* ‘transposedValueDataElement’ and takes another *dataElement* as *transformationOperand*. The values of this second data element become the values of the transposed data elements.

Table 64 and Table 65 give an example of how this function works.

The number of animals is reported in one record and the number of herds/flocks is reported in another record, as can be seen in the example dataset in Table 64. The animal species is reported in the data element ‘matrix’ (‘Pigs’ in the example). The number is inserted in the data element ‘population’, and the data element ‘unit’ references the unit of the number; in the example, this is ‘animal’ and ‘herd/flock’.

1. Example of dataset

|  |  |  |
| --- | --- | --- |
| Matrix | Unit | Population |
| Pigs (code A006641A) | Animal (code G199A) | 1 000 |
| Pigs (code A006641A) | Herd/flock (code G202A) | 50 |

In order to compare the reported numbers regarding the matrix ‘Pigs’, the data elements ‘unit’ and ‘population’ need to be transposed. The values in the data element ‘unit’ become new data elements, whereas the values in the data element ‘population’ are arranged appropriately as values in the new data elements, as can be seen in the example dataset after transposition in Table 65, in which the data elements ‘unit’ and ‘population’ disappear.

1. Example of dataset after transposition

|  |  |  |
| --- | --- | --- |
| Matrix | Animal (code G199A) | Herd/flock (code G202A) |
| Pigs (code A006641A) | 1 000 | 50 |

Example of XML file to transpose data before verifying a business rule

[…]

<forEach>

<dataElement>matrix</dataElement>

</forEach>

<transformation>

<transformationType>transposition</transformationType>

<input>

<transformationRole>transposedDataElement</transformationRole>

<transformationOperand>

<dataElement>unit</dataElement>

</transformationOperand>

</input>

<input>

<transformationRole>transposedValueDataElement</transformationRole>

<transformationOperand>

<dataElement>population</dataElement>

</transformationOperand>

</input>

</transformation> […]

### condition and verify

The two elements *condition* and *verify* contain exactly the same structure (see Figure 36 and Figure 37). The *verify* element holds the quality criteria that must or should be fulfilled by the reported data depending on the *infoType*. The expression that can be created based on the enclosed elements must or should be fulfilled, otherwise the data collection system raises an error or a warning, respectively.



1. Element *condition*

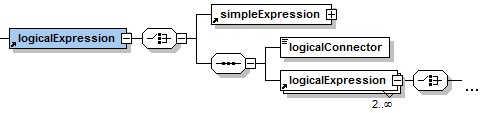


1. Element *verify*

Although the *condition* element is constructed and executed in the same way as the *verify* element, it has a completely different purpose. The *condition* element is an option to limit the records that must comply with the business rule: some requirements given in the *verify* element must be true only under certain conditions. For example, some values are only relevant if some other value is reported. The *condition* and the *verify* elements together form an if-statement.

The *condition* element is optional. If no *condition* element is defined in a business rule, the criteria in the *verify* element must be satisfied by all records in the scope.

The two elements *condition* and *verify* contain a *logicalExpression* that can be either a single expression, or two or more expressions linked by a logical connector, i.e. AND or OR (see Figure 38). An expression can be evaluated as being either TRUE or FALSE. In the following paragraphs, how to set up a simple*Expression* and a *logicalExpression* element will be explained.



1. Element *logicalExpression*

### logicalExpression and logicalConnector

Two or more expressions can be linked by logical connectors. Every expression is evaluated to a true value first, then these results are combined logically creating a single true value for the logical expression as a whole.

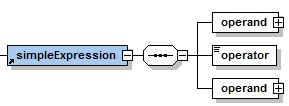
To set up a *logicalExpression* made of two ore more expressions, a *logicalConnector* element must be specified; possible logical connectors are reported in Table 66. In addition, it must contain at least two further *logicalExpression* elements that recursively reference to the same structure (see Figure 38). Logical connectors have an implicit precedence. An AND is always considered before an OR if not amended by the use of brackets. However, in the business rule structure defined here, brackets are not applicable. The logical expressions shall be evaluated according to their level in the hierarchy of the XML tree, i.e. in the reverse order of their opening tags.

1. Logical connectors

|  |  |
| --- | --- |
| *logicalConnector* | Description |
| and | Logical AND |
| or | Logical OR |
| nand | Logical “NOT AND” (negation of the conjuction) |
| nor | Logical “NOT OR” (negation of the [disjunction](https://en.wikipedia.org/wiki/Logical_disjunction)) |

### simpleExpression and operand

A *simpleExpression* element is always composed of three elements: two *operand* elements and an *operator* element (see Figure 39). If a particular operator requires a left and a right operand, the first operand element has to be intended as the operand on the left side of the operator and the second as the one on the right. An *operand* element has the type *operandType* (see section 10.4.21).

**

1. Element *simpleExpression*

Example of two logical expression linked by the connector AND

[…]

<verify>

<logicalExpression>

<logicalConnector>and</logicalConnector>

<logicalExpression>

<simpleExpression>

<operand>

[…]

</operand>

</simpleExpression>

</logicalExpression>

<logicalExpression>

<simpleExpression>

<operand>

[…]

</operand>

</simpleExpression>

</logicalExpression>

</logicalExpression>

</verify>

[…]

### ignoreNull

In cases where no value is reported for an optional data element, its value will be NULL in the database. This means that an operand in an expression can be NULL, leading to the situation that a NULL value is compared with a value other than NULL. To decide whether the business rule should take into account missing values, the element *ignoreNull* can be used. Only two values are possible: ‘yes’, which means that all missing values are ignored, and ‘no’, which means that null values are taken into account. If missing values are to be ignored only for certain variables, then the value in *ignoreNull* element should be set to ‘no’ and a condition should be specified.

### operator

Operators link two operands with each other, forming an expression. The expressions are always read from left to right. Table 67 shows the operators that can be used in the *operator* element.

Instead of using symbols, the operators are spelled out in full. Although they become quite verbose, this helps to avoid problems that would occur in XML files with left angle brackets (symbol <) and right angle brackets (symbol >) to express less or greater than. To improve the readability of an expression, all of the operators contain a verb and are written in camel case (medial capitals). However, the interpretation of operators should not be case sensitive. Operators do not contain any blanks owing to parsability. For each operator its inversion also exists.

1. Operators

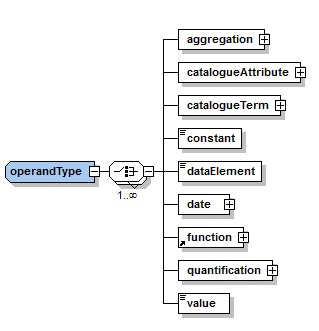
| Operator | Description |
| --- | --- |
| is | Is, is a |
| isNot | Is not, is not a |
| isEqualTo | Is equal to |
| isNotEqualTo | Is not equal to |
| isLessThan | Is less than |
| isGreaterThan | Is greater than |
| isLessThanOrEqualTo | Is less than or equal to |
| isGreaterThanOrEqualTo | Is greater than or equal to |
| isInList | Is included in the following list |
| isNotInList | Is not included in the following list |
| isLike | Matches the following pattern (wildcards: ‘%’ to substitute zero or more characters and ‘\_’ to substitute a single character) |
| isNotLike | Does not match the following pattern (wildcards: ‘%’ to substitute zero or more characters and ‘\_’ to substitute a single character) |
| hasAsParent | Has as parent in the catalogue hierarchy |
| hasNotAsParent | Does not have as parent in the catalogue hierarchy |
| hasAsAncestor | Has as ancestor in the catalogue hierarchy (includes the parent) |
| hasNotAsAncestor | Does not have as ancestor in the catalogue hierarchy (includes the parent) |
| matchesRegEx | Matches the pattern of the following regular expression |
| matchesNotRegEx | Does not match the pattern of the following regular expression |

### Type operandType

Operands are inputs for simple expressions, transformations and functions. In simple expressions and functions, they are called *operand*; in transformations, there is the *transformationOperand*.

All operands have the same structure based on the type *operandType* (see Figure 40). Some operands are only appropriate in conjunction with certain operators, transformation types or function types. It is the responsibility of the data manager, who sets up the business rules, to implement effective business rules.

The *operand* and *transformationOperand* elements must contain at least one of the following elements, but also combinations of these elements are possible: *aggregation*, *catalogueAttribute*, *catalogueTerm*, *constant*, *dataElement*, *date*, *function*, *quantification* and *value*.



1. Type *operandType*

#### *value*

Often, values are specified and then compared with the actual reported values. The *value* element can take text values as well as numerical values, i.e. any literal value. In particular, catalogue values (codes) go into *value* elements. Hence, regular expressions—for example ^[\t]+|[\t]+$, which matches a text that starts or ends with a whitespace—are in fact texts themselves and can therefore be written into *value* elements. Whether only one *value* element or a sequence of *value* elements is appropriate as an operand depends on the operator.

Please note that quotation marks should not be used, since the specified value is enclosed by XML tags.

Example of a data element compared to a value

<businessRule>

<businessRuleCode>example\_for\_value</businessRuleCode>

<description>The value in 'Result value' (resVal) must be greater than 0;</description>

<infoMessage>resVal is less than 0;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>resVal</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<ignoreNull>yes</ignoreNull>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>resVal</dataElement>

</operand>

<operator>isGreaterThan</operator>

<operand>

<value>0</value>

<operand>

</simpleExpression>

</logicalExpression>

</verify>

</businessRule>

Example of data element compared to a list of values (based on a specified condition)

<businessRule>

<businessRuleCode>example9</businessRuleCode>

<description>If the value in 'Product treatment' (prodTreat) is 'Milk pasteurisation' (T150A), then the value in 'Product code' (prodCode) must be equal to 'Milk' (P1020000A), or 'Milk Cattle' (P1020010A), or 'Milk Sheep' (P1020020A), or 'Milk Goat' (P1020030A), or 'Milk Horse' (P1020040A), or 'Milk Others' (P1020990A);</description>

<infoMessage>prodCode is not milk of animal origin, though prodTreat is milk pasteurisation;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>resVal</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<ignoreNull>yes</ignoreNull>

<condition>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>prodTreat</dataElement>

</operand>

<operator>isEqualTo</operator>

<operand>

<value>T150A</value>

<operand>

</simpleExpression>

</logicalExpression>

</condition>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>prodCode</dataElement>

</operand>

<operator>isInList</operator>

<operand>

<value>P1020000A</value>

<value>P1020010A</value>

<value>P1020020A</value>

<value>P1020030A</value>

<value>P1020040A</value>

<value>P1020990A</value>

<operand>

</simpleExpression>

</logicalExpression>

</verify>

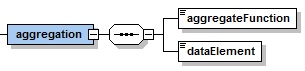
</businessRule>

*aggregation* and *aggregateFunction*

The aggregate functions *sum*, *max* and *min* can be applied only to data elements taking numerical values, whereas the *count* and *countDistinct* functions can be applied to data elements of any type. They are specified in the *aggregateFunction* element (see Table 68) that is inside the element *aggregation*, while the *dataElement* element contains the name of the variable that has to be aggregated (see Figure 41). If a sub-set of records has to be evaluated, then a *forEach* element shall be used together with the *aggregation* element. Repeatable data elements are a special case: if the values within a repeatable data element are to be checked by the business rule (for example, count of values), then the *aggregation* is not suitable for this purpose, instead a *quantification* element should be used. However, if the repeatable data element itself is to be counted, the *dataElement* element inside the *aggregation* should be used. Aggregate functions cannot be applied to compound data elements, only to the simple data elements within them.

1. Aggregate functions

|  |  |
| --- | --- |
| *aggregateFunction* | Description |
| count | Number of items |
| countDistinct | Number of distinct items (duplicates are not counted) |
| max | Maximum of a set of numerical values |
| min | Minimum of a set of numerical values |
| sum | Sum of numerical values |



1. Element *aggregation*

Example of aggregate function (distinc count of a data element)

[…]

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysOrganisation</systemVariable>

</appliesTo>

<ignoreNull>yes</ignoreNull>

<forEach>

<dataElement>sampId</dataElement>

</forEach>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<aggregation>

<aggregateFunction>countDistinct</aggregateFunction>

<dataElement>anPortSeq</dataElement>

</aggregation>

</operand>

<operator>isEqualTo</operator>

<operand>

<value>1</value>

</operand>

<simpleExpression>

<logicalExpression>

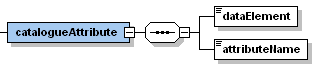
</verify>

[…]

*catalogueAttribute* and *attributeName*

In cases where a transmitted data element is in a catalogue but the business rule is defined not by the catalogue value but by some other attribute which can be found in the catalogue, this can be defined by using the *dataElement* and *attributeName* elements within the *catalogueAttribute* element (see Figure 42). The allowed values in the *attributeName* element depend on how a catalogue is implemented in the data collection system.

The idea behind the *catalogueAttribute* is to take a reported code and to look up the value of the specified attribute in the catalogue for exactly that entry. This attribute value is then compared with a value reported in another *dataElement* (as in Example 11), or it can be compared with a fixed *value*.



1. Element *catalogueAttribute*

Example of value of an attribute in the catalogue

<businessRule>

<businessRuleCode>example</businessRuleCode>

<description>The ‘Area of sampling’ (sampArea) must be within the‘Country of sampling’ (sampCountry);</description>

<infoMessage>sampArea is not within sampCountry;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>sampArea</dataElement>

<dataElement>sampCountry</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<ignoreNull>yes</ignoreNull>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<catalogueAttribute> <dataElement>sampArea</dataElemen>

<attributeName>countryCode</attributeName>

</catalogueAttribute>

</operand>

<operator>isEqualTo</operator>

<operand>

<dataElement>sampCountry</dataElement>

</operand>

</simpleExpression>

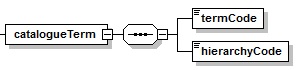
</logicalExpression>

</verify>

</businessRule>

*catalogueTerm, termCode* and *hierarchyCode*

The *catalogueTerm* element (see Figure 43) allows referencing a term in a catalogue to be used as an operand in a logical expression. For example, it could be used to determine if a value in a *dataElement* has a specific ancestor in a reference catalogue or hierarchy (i.e. a specific parent or parent of the parent). In order to specify the hierarchy that should be traversed, the element *hierarchyCode* must be used. In case where the master hierarchy is applied, the code correspondes to the name of the catalogue. The code of the searched parent or ancestor term goes in the *termCode* element.



1. Element *catalogueTerm*

Example of ancestor of a value in a catalogue hierarchy

[…]

<condition>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>zoonosis</dataElement>

</operand>

<operator>hasAsAncestor</operator>

<operand>

<catalogueTerm>

<termCode>RF-00000304-MCG</termCode>

<hierarchyCode>microParam</ hierarchyCode >

</catalogueTerm>

</operand>

</simpleExpression>

</logicalExpression>

</condition>

[…]

*constant*

In order to communicate that certain text is not just text but a keyword that has a certain predefined meaning and that needs to be interpreted in a special way by a computer program, the *constant* element should be used instead of the *value* element. Table 69 lists the constants currently available and their definitions. The constants *null*, *unique*, *constant* and *validDate* should be used in combination with the operators *is* and *isNot*.For example, to check whether a data element is provided, use ‘*isNot null*’. In this way, it is possible to force a value to be reported in an optional data element. The constant *currentDate* and *currentYear* will most probably be used with the operators *isLessThan* and *isLessThanOrEqualTo*. For the *dataCollectionYear* constant, other comparisons also make sense, e.g. when checking historical data.

1. Constants

|  |  |
| --- | --- |
| *constant* | Description |
| constant | Keyword to indicate that, within the specified context, a value or a combination of values is constant (i.e. value(s) stay the same over several sub-selected records) |
| currentDate | Keyword to refer to the current date when the business rule is applied |
| currentYear | Keyword to refer to the current year when the business rule is applied |
| dataCollectionYear | Keyword to refer to the year of the data collection to compare it with a reported year comprising a four-digit number |
| null | Keyword to indicate that a value is not reported |
| validDate | Keyword to check if a sequence of a *day*, a *month* and a *year* element is a valid date in a calendar, or a sequence of a *month* and a *year* element is a valid partial date |
| unique | Keyword to indicate that, within the specified context, a value or a combination of values is unique (i.e. occurs only once) |

Example of constant operand

[…]

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>analysisY</dataElement>

</operand>

<operator>isLessThanOrEqualTo</operator>

<operand>

<constant>currentYear</constant> </operand>

</simpleExpression>

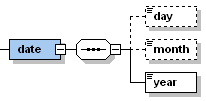
<logicalExpression>

</verify>

[…]

*date*, *day*, *month* and *year*

When evaluating dates, it is essential to know in which order the day, the month and the year are given. For this reason, a sequence of the elements *day*, *month* and *year* is defined within a *date* element (see Figure 44). The *day* and the *month* element are optional, providing the possibility to check partial dates. Valid dates can include *year* only; *year* and *month*; or *year*, *month* and *day*; however, *day* only, *month* only, *day* and *month*, or *day* and *year* are not valid combinations.



1. The *date* element is a sequence comprising the elements *day*, *month* and *year*

Example of validation of the date of analysis

[…]

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<date>

<day>analysisD</day>

<month>analysisM</month>

<year>analysisY</year>

</date>

</operand>

<operator>is</operator>

<operand>

<constant>validDate</constant> </operand>

</simpleExpression>

<logicalExpression>

</verify>

[…]

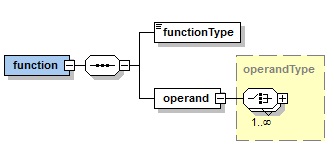
*function* and *functionType*

The *function* element is used to modify or combine the values of one ore more data elements (see Figure 45). It requires two inputs: a *functionType* and an *operand* that could contain one or many elements of type *operandType*. The mathematical operation that should be applied is specified in the element *functionType*. Table 70 lists the predefined values. However, this list will be extended in the future if necessary.

The numeric data elements or values that should be used in the calculation are specified in the *operand* element. Usually, a *dataElement* element in which numerical values are reported or a static number provided in a *value* element will be reasonable as operands.

Similar to operators, if for some functions it makes a difference whether an operand is on the left or on the right side of the operation, the list of operands must follow the correct mathematical order (e.g. for subtraction, the minuend would be the first and the subtrahend the second in the list of operands).

A *function* element can recursively contain another *function* element as an operand if needed.



1. Element *function*
2. Function types are mathematical operations

|  |
| --- |
| *functionType* |
| Addition |
| Subtraction |
| Multiplication |
| Division |

Example of function

[…]

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>analysisY</dataElement>

</operand>

<operator>isLessThanOrEqualTo</operator>

<operand>

<function>

<functionType>addition</functionType> <operand>

<dataElement>repYear</dataElement>

<value>1</value>

</operand> </function>

</operand>

</simpleExpression>

<logicalExpression>

</verify>

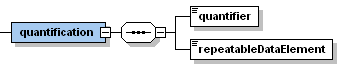
[…]

*quantification* and *quantifier*

Repeatable data elements can contain more than one value. To make it possible to check such a list of values, quantifiers are to be used. A quantifier determines whether a comparison must be fulfilled by all values, by at least one value, by exactly one value or by no value within the repeatable data element (see Table 71). A *quantifier* elementand a *repeatableDataElement* element, which refer to the name of a repeatable data element as defined in the data model, form together a *quantification* element (see Figure 46).

1. Quantifiers

|  |  |
| --- | --- |
| *quantifier* | Description |
| allValues | The evaluation of the operator applies to all values within the repeatable data element |
| atLeastOneValue | The evaluation of the operator applies to at least one value within the repeatable data element |
| exactlyOneValue | The evaluation of the operator applies to exactly one value within the repeatable data element |
| noValue | The evaluation of the operator does not apply to any value within the repeatable data element |



1. Element *quantification*

Example of quantification

[…]

<condition>

<logicalExpression>

<simpleExpression>

<operand>

<quantification>

<quantifier>atLeastOneValue</quantifier>

<repeatableDataElement>actTakenCode</repeatableDataElement>

</quantification>

</operand>

<operator>isEqualTo</operator>

<operand>

<value>R</value>

</operand>

</simpleExpression>

</logicalExpression>

</condition>

[…]

## Examples for business rules

This section extends examples already presented in the above sections and introduces new ones, to show how to use the above-specified structure to write complete business rules. All examples are based on the SSD2 data model with exceptions of examples 2 and 5, which refers to the prevalence data model, and example 6, which refers to the animal population data model. These are used in the biological monitoring data collection.

The *description* field explains the requirements of the business rule.

### Example 1: a data element can become mandatory depending on another data element’s value

<businessRule>

<businessRuleCode>example1</businessRuleCode>

<description>If the value in the data element ‘Type of result’ (resType) is

‘Non Detected Value (below LOD)’ (LOD), then a value must be reported in the data element ‘Result LOD’ (resLOD);</description>

<infoMessage>resLOD is missing, though resType is ‘Non Detected Value (below LOD)’ (LOD);</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>resLOD</dataElement>

<dataElement>resType</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<ignoreNull>no</ ignoreNull>

<condition>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>resType</dataElement>

</operand>

<operator>isEqualTo</operator>

<operand>

<value>LOD</value>

</operand>

</simpleExpression>

</logicalExpression>

</condition>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>resLOD</dataElement>

</operand>

<operator>isNot</operator>

<operand>

<constant>null</constant>

</ operand >

<simpleExpression>

</logicalExpression>

</verify>

</businessRule>

### Example 2: logical expression, ancestor in the catalogue hierarchy and a given list

<businessRule>

<businessRuleCode>example2</businessRuleCode>

<description>If the value in ‘Zoonotic agent’ (zoonosis) is in level\_1 ‘Salmonella’ (RF-00000304-MCG) and the value in ‘Matrix’ (matrix) is

‘Gallus gallus (fowl) - breeding flocks for broiler production line - adult’ (A041001A), or

‘Gallus gallus (fowl) - breeding flocks for egg production line - adult’ (A041019A), or

[…]

‘Turkeys - fattening flocks - before slaughter’ (A041278A),

then a value in ‘Target verification’ (target) must be reported;</description>

<infoMessage>target is missing, though zoonosis is a Salmonella and matrix is an animal species for

which the target verification is mandatory;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>target</dataElement>

<dataElement>zoonosis</dataElement>

<dataElement>matrix</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<condition>

<logicalExpression>

<logicalConnector>and</logicalConnector>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>zoonosis</dataElement>

</operand>

<operator>hasAsAncestor</operator>

<operand>

<catalogueTerm>

<termCode>RF-00000304-MCG</termCode>

<hierarchyCode>microParam</ hierarchyCode >

</catalogueTerm>

</operand>

</simpleExpression>

</logicalExpression>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>matrix</dataElement>

</operand>

<operator>isInList</operator>

<operand>

<value>A041001A</value>

<value>A041019A</value>

[…]

<value>A041278A</value>

</operand>

</simpleExpression >

</logicalExpression>

</logicalExpression>

</condition>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>target</dataElement>

</operand>

<operator>isNot</operator>

<operand>

<constant>null</constant>

</operand>

</simpleExpression >

</logicalExpression>

</verify>

</businessRule>

### Example 3: constant values (for each value/combination of values, another value/combination of values should be the same in all related records)

<businessRule>

<businessRuleCode>example3</businessRuleCode>

<description>The value in the data element ‘Limit for the result evaluation’ (evalLowLimit) in which the

cut-off value is reported, must be the same (constant) for each combination of values in the

data elements ‘Isolate’ (isolParamCode.base) and ‘Coded description of the parameter’

(paramCode.base) which is used to report the substance tested (per zoonosis/substance only one cut-off value is possible);</description>

<infoMessage>evalLowLimit is not the same (constant) for each combination of values in isolParamCode and paramCode;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>evalLowLimit</dataElement>

<dataElement>isolParamCode.base</dataElement>

<dataElement>paramCode.base</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysOrganisation</systemVariable>

</appliesTo>

<ignoreNull>no</ignoreNull>

<forEach>

<dataElement>isolParamCode.base</dataElement>

<dataElement>paramCode.base</dataElement>

</forEach>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>evalLowLimit</dataElement>

</operand>

<operator>is</operator>

<operand>

<constant>constant</constant>

</operand>

</simpleExpression>

</logicalExpression>

</verify>

</businessRule>

### Example 4: uniqueness of values for a data provider and previous data collections

<businessRule>

<businessRuleCode>example4</businessRuleCode>

<description>If the value in the data element ‘Coded description of the parameter’ (paramCode.base) is different from ‘Not in list’ (RF-XXXX-XXX-XXX), then the combination

of values in the data elements ‘Coded description of the parameter’ (paramCode.base), ‘Sample taken identification code’ (sampId), and ‘Sample analysed portion sequence’ (anPortSeq) must be unique for a data provider (sysOrganisation), also with respect to previous pesticide data collections, i.e only one result per parameter for each

sample analysed portion per sample;</description>

<infoMessage>More than one paramCode per sample analysed portion for one sample is reported, though paramCode.base is different from

‘Not in list’ (RF-XXXX-XXX-XXX);</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>paramCode.base</dataElement>

<dataElement>sampId</dataElement>

<dataElement>anPortSeq</dataElement>

</checkedDataElements>

<includes>

<include>

<databaseTable>FINAL\_MOPER\_11</databaseTable>

</include>

<include>

<databaseTable>MOPER</databaseTable>

<dataCollection>MOPER.PPP\_2012\_FINAL</dataCollection>

<dataCollection>MOPER.PPP\_2013\_FINAL</dataCollection>

</include>

</includes>

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysOrganisation</systemVariable>

</appliesTo>

<ignoreNull>no</ignoreNull>

<condition>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>paramCode.base</dataElement>

</operand>

<operator>isNotEqualTo</operator>

<operand>

<value>RF-XXXX-XXX-XXX</value>

</operand>

</simpleExpression>

</logicalExpression>

</condition>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>paramCode.base</dataElement>

<dataElement>sampId</dataElement>

<dataElement>anPortSeq</dataElement>

</operand>

<operator>is</operator>

<operand>

<constant>unique</constant>

</operand>

</simpleExpression>

</logicalExpression>

</verify>

</businessRule>

### Example 5: aggregate function: sum of values sharing the same context

<businessRule>

<businessRuleCode>example5</businessRuleCode>

<description>The value in 'Total units positive' (totUnitsPositive) must be less than or equal to the sum of values in 'Number of units positive' (unitsPositive) of the same context. The data elements of the context are: matrix, zoonosis.base at level 1, sampStage, sampOrig, sampType, sampContext, sampler, progSampStrategy, sampDetails, sampArea, sampUnit;</description>

<infoMessage>totUnitsPositive is greater than the sum of unitsPositive of the same context;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>matrix</dataElement>

<dataElement>zoonosis.base</dataElement>

<dataElement>sampStage</dataElement>

<dataElement>sampOrig</dataElement>

<dataElement>sampType</dataElement>

<dataElement>sampContext</dataElement>

<dataElement>sampler</dataElement>

<dataElement>progSampStrategy</dataElement>

<dataElement>sampDetails</dataElement>

<dataElement>sampArea</dataElement>

<dataElement>sampUnit</dataElement>

<dataElement>totUnitsPositive</dataElement>

<dataElement>unitsPositive</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysOrganisation</systemVariable>

</appliesTo>

<ignoreNull>yes</ignoreNull>

<forEach>

<dataElement>matrix</dataElement>

<catalogueAncestor>

<dataElement>zoonosis.base</dataElement>

<ancestorLevel>1</ancestorLevel>

<hierarchyCode>microParam</hierarchyCode>

</catalogueAncestor>

<dataElement>sampStage</dataElement>

<dataElement>sampOrig</dataElement>

<dataElement>sampType</dataElement>

<dataElement>sampContext</dataElement>

<dataElement>sampler</dataElement>

<dataElement>progSampStrategy</dataElement>

<dataElement>sampDetails</dataElement>

<dataElement>sampArea</dataElement>

<dataElement>sampUnit</dataElement>

</forEach>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>totUnitsPositive</dataElement>

</operand>

<operator>isLessThanOrEqualTo</operator>

<operand>

<aggregation> <aggregateFunction>sum</aggregateFunction> <dataElement>unitsPositive</dataElement>

</aggregation>

</operand>

</simpleExpression>

</logicalExpression>

<verify>

</businessRule>

### Example 6: Transposition of values

<businessRule>

<businessRuleCode>example6</businessRuleCode>

<description>The number of animals must be less than the number of herds/flocks of the same animal species. For each value in the data element ‘Matrix’ (matrix), the number in the data element ‘Population’ (population) corresponding to the data element ‘Unit’ (unit) with the value ‘animal’ (G199A) must be less than the number in the data element ‘Population’ (population) corresponding to data element ‘Unit’ (unit) with the value ‘herd/flock’ (G202A);</description>

<infoMessage>The number of animals is not less than the number of herds/flocks of the same animal species;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>population</dataElement>

<dataElement>unit</dataElement>

<dataElement>matrix</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysDataCollection</systemVariable>

<systemVariable>sysCountry</systemVariable>

</appliesTo>

<ignoreNull>no</ignoreNull>

<forEach>

<dataElement>matrix</dataElement>

</forEach>

<transformation>

<transformationType>transposition</transformationType>

<input>

<transformationRole>transposedDataElement</transformationRole>

<transformationOperand>

<dataElement>unit</dataElement>

</transformationOperand>

</input>

<input>

<transformationRole>transposedValueDataElement</transformationRole>

<transformationOperand>

<dataElement>population</dataElement>

</transformationOperand>

</input>

</transformation>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>G199A</dataElement>

</operand>

<operator>isLessThan</operator>

<operand>

<dataElement>G202A</dataElement>

</operand>

</ simpleExpression >

<logicalExpression>

</verify>

</businessRule>

### Example 7: pattern matching with LIKE

<businessRule>

<businessRuleCode>example7</businessRuleCode>

<description>Warning: If the attribute ‘Comment’ of the data element ‘Additional sampling program information’ (progInfo.com) contains the text ‘HACCP’, then it is likely that the value in the data element ‘Sampler’ (sampler) should be ‘HACCP and own check’ (CX04A);</description>

<infoMessage>sampler is not ‘HACCP and own check’ (CX04A), though progInfo.com contains the text ‘HACCP’;</infoMessage>

<infoType>warning</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>sampler</dataElement>

<dataElement>progInfo.com</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<condition>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>progInfo.com</dataElement>

</operand>

<operator>isLike</operator>

<operand>

<value>%HACCP%</value>

</operand>

</simpleExpression>

</logicalExpression>

</condition>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>sampler</dataElement>

</operand>

<operator>isEqualTo</operator>

<operand>

<value>CX04A</value>

</operand>

<simpleExpression>

</logicalExpression>

</verify>

</businessRule>

### Example 8: a value matches a certain structural pattern (matchesRegEx)

<businessRule>

<businessRuleCode>example8</businessRuleCode>

<description>The number in the attribute ‘Percentage of alcohol’ of the data element

‘Expression of result percentage’ (exprResPerc.alcoholPerc) must be a decimal with a precision of 2 (exactly two digits after the decimal point);</description>

<infoMessage>exprResPerc.alcoholPerc does not have exactly two digits after the

decimal point;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>exprResPerc.alcoholPerc</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<dataElement>exprResPerc.alcoholPerc</dataElement>

</operand>

<operator>matchesRegEx</operator>

<operand>

<value>\d+\.\d{2}</value>

</operand>

</simpleExpression>

<logicalExpression>

</verify>

</businessRule>

### Example 9: comparison of dates

<businessRule>

<businessRuleCode>example9</businessRuleCode>

<description>The date when the analysis was done, reported in the data elements

‘Day of analysis’ (analysisD), ‘Month of analysis’ (analysisM), and

‘Year of analysis’ (analysisY), must be less than or equal to the current date;</description>

<infoMessage>The date in analysisD, analysisM, and analysisY is not less than

the current date;</infoMessage>

<infoType>error</infoType>

<status>active</status>

<lastUpdate>2014-07-23</lastUpdate>

<checkedDataElements>

<dataElement>analysisD</dataElement>

<dataElement>analysisM</dataElement>

<dataElement>analysisY</dataElement>

</checkedDataElements>

<appliesTo>

<systemVariable>sysRecord</systemVariable>

</appliesTo>

<verify>

<logicalExpression>

<simpleExpression>

<operand>

<date>

<day>analysisD</day>

<month>analysisM</month>

<year>analysisY</year>

</date>

</operand>

<operator>isLessThanOrEqualTo</operator>

<operand>

<constant>currentDate</constant>

</operand>

</simpleExpression>

<logicalExpression>

</verify>

</businessRule>

## General business rules

A general business rule is a business rule that checks an inherent feature of a data element or of a combination of data elements. For example, it is reasonable to limit generally acceptable numbers to a range between 0 and 100 when a data element is designated for reporting a percentage. In a specific data collection, it might be appropriate to further reduce this range. This is feasible through the creation of a specific business rule applicable to a data collection which would overrule any general business rule.

The specific business rules are defined by EFSA’s data managers who are in charge of the data collections in conjunction with relevant stakeholders. They are published with the other information regarding the data collection.

The general business rules of this section (Table 74) refer to the SSD2 data model. Please note that the provision of a value in data elements that are defined as mandatory in the data model is not checked by business rules but by the file format structure validation. The mandatory data elements in the SSD2 data model are the following: resId, analysisY, paramCode, paramType, sampId, sampCountry, sampMatCode, sampY and sampMatType. Similarly, the checking of the validity of values, i.e. checking transmitted values against controlled catalogues, is performed in the data collection metadata validation.

General business rules for the SSD2 data model

| Business  RuleCode | description | infoMessage | infoType | checkedData  Elements | forEach | appliesTo |
| --- | --- | --- | --- | --- | --- | --- |
| GBR001 | The ‘Result identification code’ (resId) must be unique (the value may occur only once); | resId is not unique (duplicates exist); | error | resId |  | sysDataCollection sysOrganisation |
| GBR002 | The descriptors for the sample taken and the matrix sampled (sections D, E) and the ‘Sampling event identification code’ (sampEventId) must be constant (the same) for all records with the same ‘Sample taken identification code’ (sampId). The data elements in the sections D and E are: sampId, repCountry, sampCountry, sampArea, repYear, sampY, sampM, sampD, sampSize, sampSizeUnit, sampInfo, sampMatType, sampMatCode, sampMatText, origCountry, origArea, origFishAreaCode, origFishAreaText, procCountry, procArea, sampMatInfo; | The sample taken or matrix sampled descriptors (sections D, E) or sampEventId are not constant for all records with the same sampId; | error | sampEventId sampId repCountry sampCountry sampArea repYear sampY sampM sampD sampSize sampSizeUnit sampInfo sampMatType sampMatCode sampMatText origCountry origArea origFishAreaCode origFishAreaText procCountry procArea sampMatInfo  progId  progLegalRef  progType  progInfo sampMethod  sampUnitType sampUnitSize sampUnitSizeUnit sampUnitIds | sampId | sysDataCollection sysOrganisation |
| GBR003 | The descriptors for the sample analysed and the matrix analysed (sections F, G) and the ‘Sample taken identification code’ (sampId) must be constant (the same) for all records with the same ‘Sample analysed identification code’ (sampAnId). The data elements in the sections F and G are: sampAnId, sampAnRefTime, analysisY, analysisM, analysisD, sampAnInfo, anMatCode, anMatText, anMatInfo; | The sample analysed or matrix analysed descriptors (sections F, G) or sampId are not constant for all records with the same sampAnId; | error | sampId sampAnId sampAnRefTime analysisY analysisM analysisD sampAnInfo anMatCode anMatText anMatInfo | sampAnId | sysDataCollection sysOrganisation |
| GBR004 | The descriptors for the sample analysed portion (section H) and the ‘Sample analysed identification code’ (sampAnId) must be constant (the same) for all records with the same ‘Sample analysed portion sequence’ (anPortSeq). The data elements in the section H are: anPortSeq, anPortSize, anPortSizeUnit, anPortInfo; | The sample analysed portion descriptors (section H) or sampAnId are not constant for all records with the same anPortSeq; | error | sampAnId anPortSeq anPortSize anPortSizeUnit anPortInfo | anPortSeq | sysDataCollection sysOrganisation |
| GBR005 | The descriptors for the isolate (section I) and the ‘Sample taken identification code’ (sampId) must be constant (the same) for all records with the same ‘Isolate identification’ (isolId). The data elements in the section I are: isolId, isolParamCode, isolParamText, isolInfo; | The isolate descriptors (section I) or sampId are not constant for all records with the same isolId; | error | sampId isolId isolParamCode isolParamText isolInfo | isolId | sysDataCollection sysOrganisation |
| GBR006 | The ‘Local organisation country’ (localOrgCountry) must be constant (the same) for all records with the same ‘Local organisation identification code’ (localOrgId); | localOrgCountry is not constant for all records with the same localOrgId; | error | localOrgId localOrgCountry | localOrgId | sysDataCollection sysOrganisation |
| GBR007 | The ‘Coded description of the isolate’ (isolParamCode ), the ‘Text description of the isolate’ (isolParamText) and the ‘Additional information on the isolate’ (isolInfo) must be constant (the same) for all records with the same ‘Isolate identification’ (isolId); | isolParamCode, isolParamText or isolInfo is not constant for all records with the same isolId; | error | isolId isolParamCode isolParamText isolInfo | isolId | sysDataCollection sysOrganisation |
| GBR008 | The ‘Laboratory country’ (labCountry) must be constant (the same) for all records with the same ‘Laboratory identification code’ (labId); | labCountry is not constant for all records with the same labId; | error | labId labCountry | labId | sysDataCollection sysOrganisation |
| GBR009 | The ‘Analytical method reference code’ (anMethRefCode), the ‘Analytical method type’ (anMethType), the ‘Analytical method code’ (anMethCode), the ‘Analytical method text’ (anMethText), and the ‘Additional information on the analytical method’ (anMethInfo) must be constant (the same) for all records with the same ‘Analytical method identification’ (anMethRefId); | anMethRefCode, anMethType, anMethCode, anMethText or anMethInfo is not constant for all records with the same anMethRefId; | error | anMethRefId anMethRefCode anMethType anMethCode anMethText anMethInfo | anMethRefId | sysDataCollection sysOrganisation |
| GBR010 | The ‘Area of sampling’ (sampArea) must be within the ‘Country of sampling’ (sampCountry); | sampArea is not within sampCountry; | error | sampArea Country |  | sysRecord |
| GBR011 | The ‘Area of origin of the sample taken’ (origArea) must be within the ‘Country of origin of the sample taken’ (origCountry); | origArea is not within origCountry; | error | origArea origCountry |  | sysRecord |
| GBR012 | The ‘Area of processing of the sample taken’ (procArea) must be within the ‘Country of processing of the sample taken’ (procCountry); | procArea is not within procCountry; | error | procArea procCountry |  | sysRecord |
| GBR013 | If in the ‘Coded description of the matrix of the sample taken’ the ‘generic-term facet’ (sampMatCode.gen) is reported with the descriptor ‘Other’ (A07XE), then a text must be reported in the ‘Text description of the matrix of the sample taken’ (sampMatText); | sampMatText is missing, though mandatory if sampMatCode.gen is ‘Other’ (A07XE); | error | sampMatCode.gen sampMatText |  | sysRecord |
| GBR014 | If in the ‘Coded description of the analysed matrix’ the ‘generic-term facet’ (anMatCode.gen) is reported with the descriptor ‘Other’ (A07XE), then a text must be reported in the ‘Text description of the matrix analysed’ (anMatText); | anMatText is missing, though mandatory if anMatCode.gen is ‘Other’ (A07XE); | error | anMatCode.gen anMatText |  | sysRecord |
| GBR015 | If the reported value in the ‘Coded description of the parameter’ (paramCode.param) is ‘Not in list’ (RF-XXXX-XXX-XXX), then a text must be reported in the ‘Parameter text’ (paramText); | paramText is missing, though mandatory if paramCode.param is ‘Not in list’ (RF-XXXX-XXX-XXX); | error | paramCode.param paramText |  | sysRecord |
| GBR016 | Status pending: attribute not defined for anMethCode If the reported value in the ‘Analytical method code’ (anMethCode.meth) is ‘Classification not possible’ (F001A), then a text must be reported in the ‘Analytical method text’ (anMethText); | anMethText is missing, though mandatory if anMethCode.meth is ‘Classification not possible’ (F001A); | error | anMethCode.meth anMethText |  | sysRecord |
| GBR017 | The value in the data element ‘Percentage of fat’ (exprResPerc.fatPerc) must be expressed as a percentage and so be between ‘0’ and ‘100’ (e.g. ‘40’ must be reported for 40%); | exprResPerc.fatPerc is not between ‘0’ and ‘100’; | error | exprResPerc.fatPerc |  | sysRecord |
| GBR018 | The value in the data element ‘Percentage of moisture ‘ (exprResPerc.moistPerc) must be expressed as a percentage and so be between ‘0’ and ‘100’ (e.g. ‘40’ must be reported for 40%); | exprResPerc.moistPerc is not between ‘0’ and ‘100’; | error | exprResPerc.moistPerc |  | sysRecord |
| GBR019 | The value in the data element ‘Percentage of alcohol’ (exprResPerc.alcoholPerc) must be expressed as a percentage and so be between ‘0’ and ‘100’ (e.g. ‘40’ must be reported for 40%); | exprResPerc.alcoholPerc is not between ‘0’ and ‘100’; | error | exprResPerc.alcoholPerc |  | sysRecord |
| GBR020 | If the value in the ‘Expression of result type’ (exprResType) is ‘Fat weight’ (B003A), then a value must be reported in the ‘Percentage of fat’ (exprResPerc.fatPerc); | exprResPerc.fatPerc is missing, though mandatory if exprResType is ‘Fat weight’ (B003A); | warning | exprResPerc.fatPerc exprResType |  | sysRecord |
| GBR021 | If the value in the ‘Expression of result type’ (exprResType) is ‘Dry matter’ (B002A), then a value must be reported in the ‘Percentage of moisture ‘ (exprResPerc.moistPerc); | exprResPerc.moistPerc is missing, though mandatory if exprResType is ‘Dry matter’ (B002A); | warning | exprResPerc.moistPerc exprResType |  | sysRecord |
| GBR022 | If a ‘Sampling unit size’ (sampUnitSize) is reported, then a ‘Sampling unit size unit’ (sampUnitSizeUnit) must be reported; | sampUnitSizeUnit is missing, though sampUnitSize is reported; | error | sampUnitSize sampUnitSizeUnit |  | sysRecord |
| GBR023 | If a ‘Sample taken size’ (sampSize) is reported, then a ‘Sample taken size unit’ (sampSizeUnit) must be reported; | sampSizeUnit is missing, though sampSize is reported; | error | sampSize sampSizeUnit |  | sysRecord |
| GBR024 | If a ‘Sample analysed portion size’ (anPortSize) is reported, then a ‘Sample analysed portion size unit’ (anPortSizeUnit) must be reported; | anPortSizeUnit is missing, though anPortSize is reported; | error | anPortSize anPortSizeUnit |  | sysRecord |
| GBR025 | If the value reported in ‘Type of result’ (resType) is ‘Qualitative Value (Binary)’ (BIN) (i.e. a qualitative value), then a ‘Result qualitative value’ (resQualValue) must be reported; | resQualValue is missing, though resType is ‘Qualitative Value (Binary)’ (BIN); | error | resQualValue resType |  | sysRecord |
| GBR026 | If a value is reported in at least one of the following data elements: ‘Result LOD’ (resLOD), ‘Result LOQ’ (resLOQ), ‘Result lower limit of the working range’ (resLLWR), ‘Result upper limit of the working range’ (resULWR), ‘CC alpha’ (CCalpha), ‘CC, beta’ (CCbeta), ‘Result value’ (resVal), ‘Result value uncertainty’ (resValUncert), ‘Result value uncertainty Standard deviation’ (resValUncertSD), ‘Limit for the result, evaluation ‘ (evalLowLimit), ‘Limit for the result evaluation (High limit)’ (evalHighLimit), then a ‘Result unit’ (resUnit) must be reported; | resUnit is missing, though at least one numeric data element (e.g. resLOD, resLOQ, resLLWR, resULWR, CCalpha, CCbeta, resVal, resValUncert, resValUncertSD, evalLowLimit or evalHighLimit) is reported; | error | resLOD resLOQ resLLWR resULWR CCalpha CCbeta resVal resValUncert resValUncertSD evalLowLimit evalHighLimit  resUnit |  | sysRecord |
| GBR027 | If a value is reported in ‘Limit for the result evaluation ‘ (evalLowLimit), then a ‘Type of limit for the result evaluation’ (evalLimitType) must be reported; | evalLimitType is missing, though evalLowLimit is reported; | error | evalLimitType evalLowLimit |  | sysRecord |
| GBR028 | If a value is reported in ‘Limit for the result evaluation (High limit)’ (evalHighLimit), then a ‘Limit for the result evaluation ‘ (evalLowLimit) must be reported; | evalLowLimit is missing, though evalHighLimit is reported; | error | evalLowLimit evalHighLimit |  | sysRecord |
| GBR029 | The value reported in ‘Limit for the result evaluation (High limit)’ (evalHighLimit) must be greater than the value reported in ‘Limit for the result evaluation ‘ (evalLowLimit); | evalHighLimit is not greater than evalLowLimit; | error | evalLowLimit evalHighLimit |  | sysRecord |
| GBR030 | If ‘Result value’ (resVal) is greater than ‘Limit for the result evaluation ‘ (evalLowLimit), then the value in ‘Evaluation of the result’ (evalCode) must be different from ‘below or equal to maximum permissible quantities’ (J002A); | evalCode is ‘below or equal to maximum permissible quantities’ (J002A), though resVal is greater than evalLowLimit; | error | evalCode resVal evalLowLimit |  | sysRecord |
| GBR031 | If ‘Evaluation of the result’ (evalCode) is ‘below or equal to maximum permissible quantities’ (J002A), then ‘Result value’ (resVal) must be less than or equal to ‘Limit for the result evaluation ‘ (evalLowLimit); | resVal is greater than evalLowLimit, though evalCode is ‘below or equal to maximum permissible quantities’ (J002A); | error | evalCode resVal evalLowLimit |  | sysRecord |
| GBR032 | If ‘Evaluation of the result’ (evalCode) is either ‘above maximum permissible quantities’ (J003A) or ‘Compliant due to measurement uncertainty’ (J031A), then ‘Result value’ (resVal) must be greater than ‘Limit for the result evaluation ‘ (evalLowLimit); | resVal is lower than evalLowLimit, though evalCode is either ‘above maximum permissible quantities’ (J003A) or ‘Compliant due to measurement uncertainty’ (J031A); | error | evalCode resVal evalLowLimit |  | sysRecord |
| GBR033 | If the value in the data element ‘Type of result’ (resType) is ‘Non Detected Value (below LOD)’ (LOD), then a value must be reported in the data element ‘Result LOD’ (resLOD); | resLOD is missing, though resType is ‘Non Detected Value (below LOD)’ (LOD); | error | resLOD resType |  | sysRecord |
| GBR034 | The value in ‘Result LOD’ (resLOD) must be less than or equal to the value in ‘Result LOQ’ (resLOQ); | resLOD is not less than or equal to resLOQ; | error | resLOD resLOQ |  | sysRecord |
| GBR035 | The value in ‘Result LOD’ (resLOD) must be greater than ‘0’; | resLOD is not greater than ‘0’; | error | resLOD |  | sysRecord |
| GBR036 | If the value in the data element ‘Type of result’ (resType) is ‘Non Quantified Value (below LOQ)’ (LOQ), then a value must be reported in the data element ‘Result LOQ’ (resLOQ); | resLOQ is missing, though resType is ‘Non Quantified Value (below LOQ)’ (LOQ); | error | resLOQ resType |  | sysRecord |
| GBR037 | The value in ‘Result LOQ’ (resLOQ) must be greater than ‘0’; | resLOQ is not greater than ‘0’; | error | resLOQ |  | sysRecord |
| GBR038 | If the value in the data element ‘Type of result’ (resType) is ‘Value below CCalpha (below CCα)’ (CCA), then a value must be reported in the data element ‘CC alpha’ (CCalpha); | CCalpha is missing, though resType is ‘Value below CCalpha (below CCα)’ (CCA); | error | CCalpha resType |  | sysRecord |
| GBR039 | The value in ‘CC alpha’ (CCalpha) must be less than the value in ‘CC beta’ (CCbeta); | CCalpha is not less than CCbeta; | error | CCalpha CCbeta |  | sysRecord |
| GBR040 | The value in ‘CC alpha’ (CCalpha) must be greater than ‘0’; | CCalpha is not greater than ‘0’; | error | CCalpha |  | sysRecord |
| GBR041 | If the value in the data element ‘Type of result’ (resType) is ‘Value below CCbeta (below CCβ)’ (CCB), then a value must be reported in the data element ‘CC beta’ (CCbeta); | CCbeta is missing, though resType is ‘Value below CCbeta (below CCβ)’ (CCB); | error | CCbeta resType |  | sysRecord |
| GBR042 | The value in ‘CC beta’ (CCbeta) must be greater than ‘0’; | CCbeta is not greater than ‘0’; | error | CCbeta |  | sysRecord |
| GBR043 | If the value in the data element ‘Type of result’ (resType) is ‘Numerical Value’ (VAL), then a value must be reported in the data element ‘Result value’ (resVal); | resVal is missing, though resType is ‘Numerical Value’ (VAL); | error | resVal resType |  | sysRecord |
| GBR044 | If the value in the data element ‘Type of result’ (resType) is ‘Non Detected Value (below LOD)’ (LOD), then the data element ‘Result value’ (resVal) must be empty; | resVal is reported, though resType is ‘Non Detected Value (below LOD)’ (LOD); | error | resVal resType |  | sysRecord |
| GBR045 | The value in ‘Result value’ (resVal) must be greater than ‘0’; | resVal is not greater than ‘0’; | error | resVal |  | sysRecord |
| GBR046 | The value in ‘Result value recovery rate’ (resValRec) must be greater than ‘0’; | resValRec is not greater than ‘0’; | error | resValRec |  | sysRecord |
| GBR047 | The value in ‘Result value uncertainty Standard deviation’ (resValUncertSD) must be greater than ‘0’; | resValUncertSD is not greater than ‘0’; | error | resValUncertSD |  | sysRecord |
| GBR048 | The value in ‘Result value uncertainty’ (resValUncert) must be greater than ‘0’; | resValUncert is not greater than ‘0’; | error | resValUncert |  | sysRecord |
| GBR049 | The value in ‘Result value uncertainty’ (resValUncert) must be greater than or equal to resValUncertSD. | resValUncert is less than or greater than resValUncertSD | error | resValUncert  resValUncertSD |  | sysRecord |
| GBR050 | The reporting year, reported in ‘Reporting year’ (repYear), must be a valid year; | The reporting year, reported in repYear, is not a valid year; | error | repYear |  | sysRecord |
| GBR051 | The date of the slaughtering, reported in ‘Day of slaughtering’ (sampEventInfo.slaughterD), ‘Month of slaughtering’ (sampEventInfo.slaughterM), and ‘Year of slaughtering’ (sampEventInfo.slaughterY), must be a valid date; | The combination of values in sampEventInfo.slaughterD, sampEventInfo.slaughterM, and sampEventInfo.slaughterY is not a valid date; | error | sampEventInfo.slaughterD sampEventInfo.slaughterM sampEventInfo.slaughterY |  | sysRecord |
| GBR052 | The date of the sampling, reported in ‘Day of sampling’ (sampD), ‘Month of sampling’ (sampM), and ‘Year of sampling’ (sampY), must be a valid date; | The combination of values in sampD, sampM, and sampY is not a valid date; | error | sampD sampM sampY |  | sysRecord |
| GBR053 | The date of the arrival in the laboratory, reported in ‘Arrival Day’ (sampInfo.arrivalD), ‘Arrival Month’ (sampInfo.arrivalM), and ‘Arrival Year’ (sampInfo.arrivalY), must be a valid date; | The combination of values in sampInfo.arrivalD, sampInfo.arrivalM, and sampInfo.arrivalY is not a valid date; | error | sampInfo.arrivalD sampInfo.arrivalM sampInfo.arrivalY |  | sysRecord |
| GBR054 | The date of the production, reported in ‘Day of production’ (sampMatInfo.prodD), ‘Month of production’ (sampMatInfo.prodM), and ‘Year of production’ (sampMatInfo.prodY), must be a valid date; | The combination of values in sampMatInfo.prodD, sampMatInfo.prodM, and sampMatInfo.prodY is not a valid date; | error | sampMatInfo.prodD sampMatInfo.prodM sampMatInfo.prodY |  | sysRecord |
| GBR055 | The date of the expiry, reported in ‘Day of expiry’ (sampMatInfo.expiryD), ‘Month of expiry’ (sampMatInfo.expiryM), and ‘Year of expiry’ (sampMatInfo.expiryY), must be a valid date; | The combination of values in sampMatInfo.expiryD, sampMatInfo.expiryM, and sampMatInfo.expiryY is not a valid date; | error | sampMatInfo.expiryD sampMatInfo.expiryM sampMatInfo.expiryY |  | sysRecord |
| GBR056 | The date of the analysis, reported in ‘Day of analysis’ (analysisD), ‘Month of analysis’ (analysisM), and ‘Year of analysis’ (analysisY), must be a valid date; | The combination of values in analysisD, analysisM, and analysisY is not a valid date; | error | analysisD analysisM analysisY |  | sysRecord |
| GBR057 | The date of the completion of the analysis, reported in ‘Completion day of analysis’ (sampAnInfo.compD), ‘Completion month of analysis’ (sampAnInfo.compM), and ‘Completion year of analysis’ (sampAnInfo.compY), must be a valid date; | The combination of values in sampAnInfo.compD, sampAnInfo.compM, and sampAnInfo.compY is not a valid date; | error | sampAnInfo.compD sampAnInfo.compM sampAnInfo.compY |  | sysRecord |
| GBR058 | The date of the isolation, reported in ‘Isolation day’ (isolInfo.isolD), ‘Isolation month’ (isolInfo.isolM), and ‘Isolation year’ (isolInfo.isolY), must be a valid date; | The combination of values in isolInfo.isolD, isolInfo.isolM, and isolInfo.isolY is not a valid date; | error | isolInfo.isolD isolInfo.isolM isolInfo.isolY |  | sysRecord |
| GBR059 | The reporting year, reported in ‘Reporting year’ (repYear), cannot be later than the current year; | The reporting year, reported in repYear, is later than the current year; | error | repYear |  | sysRecord |
| GBR060 | The date of the slaughtering, reported in ‘Day of slaughtering’ (sampEventInfo.slaughterD), ‘Month of slaughtering’ (sampEventInfo.slaughterM), and ‘Year of slaughtering’ (sampEventInfo.slaughterY), cannot be later than the current date; | The date of the slaughtering, reported in sampEventInfo.slaughterD, sampEventInfo.slaughterM, and sampEventInfo.slaughterY, is later than the current date; | error | sampEventInfo.slaughterD sampEventInfo.slaughterM sampEventInfo.slaughterY |  | sysRecord |
| GBR061 | The date of the sampling, reported in ‘Day of sampling’ (sampD), ‘Month of sampling’ (sampM), and ‘Year of sampling’ (sampY), cannot be later than the current date; | The date of the sampling, reported in sampD, sampM, and sampY, is later than the current date; | error | sampD sampM sampY |  | sysRecord |
| GBR062 | The date of the arrival in the laboratory, reported in ‘Arrival Day’ (sampInfo.arrivalD), ‘Arrival Month’ (sampInfo.arrivalM), and ‘Arrival Year’ (sampInfo.arrivalY), cannot be later than the current date; | The date of the arrival in the laboratory, reported in sampInfo.arrivalD, sampInfo.arrivalM, and sampInfo.arrivalY, is later than the current date; | error | sampInfo.arrivalD sampInfo.arrivalM sampInfo.arrivalY |  | sysRecord |
| GBR063 | The date of the production, reported in ‘Day of production’ (sampMatInfo.prodD), ‘Month of production’ (sampMatInfo.prodM), and ‘Year of production’ (sampMatInfo.prodY), cannot be later than the current date; | The date of the production, reported in sampMatInfo.prodD, sampMatInfo.prodM, and sampMatInfo.prodY, is later than the current date; | error | sampMatInfo.prodD sampMatInfo.prodM sampMatInfo.prodY |  | sysRecord |
| GBR064 | The date of the analysis, reported in ‘Day of analysis’ (analysisD), ‘Month of analysis’ (analysisM), and ‘Year of analysis’ (analysisY), cannot be later than the current date; | The date of the analysis, reported in analysisD, analysisM, and analysisY, is later than the current date; | error | analysisD analysisM analysisY |  | sysRecord |
| GBR065 | The date of the completion of the analysis, reported in ‘Completion day of analysis’ (sampAnInfo.compD), ‘Completion month of analysis’ (sampAnInfo.compM), and ‘Completion year of analysis’ (sampAnInfo.compY), cannot be later than the current date; | The date of the completion of the analysis, reported in sampAnInfo.compD, sampAnInfo.compM, and sampAnInfo.compY, is later than the current date; | error | sampAnInfo.compD sampAnInfo.compM sampAnInfo.compY |  | sysRecord |
| GBR066 | The date of the isolation, reported in ‘Isolation day’ (isolInfo.isolD), ‘Isolation month’ (isolInfo.isolM), and ‘Isolation year’ (isolInfo.isolY), cannot be later than the current date; | The date of the isolation, reported in isolInfo.isolD, isolInfo.isolM, and isolInfo.isolY, is later than the current date; | error | isolInfo.isolD isolInfo.isolM isolInfo.isolY |  | sysRecord |
| GBR067 | The ‘Day of slaughtering’ (sampEventInfo.slaughterD) must be between 1 and 31; | sampEventInfo.slaughterD is not between 1 and 31; | error | sampEventInfo.slaughterD |  | sysRecord |
| GBR068 | The ‘Day of sampling’ (sampD) must be between 1 and 31; | sampD is not between 1 and 31; | error | sampD |  | sysRecord |
| GBR069 | The ‘Arrival Day’ (sampInfo.arrivalD) must be between 1 and 31; | sampInfo.arrivalD is not between 1 and 31; | error | sampInfo.arrivalD |  | sysRecord |
| GBR070 | The ‘Day of production’ (sampMatInfo.prodD) must be between 1 and 31; | sampMatInfo.prodD is not between 1 and 31; | error | sampMatInfo.prodD |  | sysRecord |
| GBR071 | The ‘Day of expiry’ (sampMatInfo.expiryD) must be between 1 and 31; | sampMatInfo.expiryD is not between 1 and 31; | error | sampMatInfo.expiryD |  | sysRecord |
| GBR072 | The ‘Day of analysis’ (analysisD) must be between 1 and 31; | analysisD is not between 1 and 31; | error | analysisD |  | sysRecord |
| GBR073 | The ‘Completion day of analysis’ (sampAnInfo.compD) must be between 1 and 31; | sampAnInfo.compD is not between 1 and 31; | error | sampAnInfo.compD |  | sysRecord |
| GBR074 | The ‘Isolation day’ (IsolInfo.isolD) must be between 1 and 31; | isolInfo.isolD is not between 1 and 31; | error | isolInfo.isolD |  | sysRecord |
| GBR075 | The ‘Month of slaughtering’ (sampEventInfo.slaughterM) must be between 1 and 12; | sampEventInfo.slaughterM is not between 1 and 12; | error | sampEventInfo.slaughterM |  | sysRecord |
| GBR076 | The ‘Month of sampling’ (sampM) must be between 1 and 12; | sampM is not between 1 and 12; | error | sampM |  | sysRecord |
| GBR077 | The ‘Arrival Month’ (sampInfo.arrivalM) must be between 1 and 12; | sampInfo.arrivalM is not between 1 and 12; | error | sampInfo.arrivalM |  | sysRecord |
| GBR078 | The ‘Month of production’ (sampMatInfo.prodM) must be between 1 and 12; | sampMatInfo.prodM is not between 1 and 12; | error | sampMatInfo.prodM |  | sysRecord |
| GBR079 | The ‘Month of expiry’ (sampMatInfo.expiryM) must be between 1 and 12; | sampMatInfo.expiryM is not between 1 and 12; | error | sampMatInfo.expiryM |  | sysRecord |
| GBR080 | The ‘Month of analysis’ (analysisM) must be between 1 and 12; | analysisM is not between 1 and 12; | error | analysisM |  | sysRecord |
| GBR081 | The ‘Completion month of analysis’ (sampAnInfo.compM) must be between 1 and 12; | sampAnInfo.compM is not between 1 and 12; | error | sampAnInfo.compM |  | sysRecord |
| GBR082 | The ‘Isolation month’ (isolInfo.isolM) must be between 1 and 12; | isolInfo.isolM is not between 1 and 12; | error | isolInfo.isolM |  | sysRecord |
| GBR083 | The date of the production, reported in ‘Day of production’ (sampMatInfo.prodD), ‘Month of production’ (sampMatInfo.prodM), and ‘Year of production’ (sampMatInfo.prodY), cannot be later thanthe date of the expiry, reported in ‘Day of expiry’ (sampMatInfo.expiryD), ‘Month of expiry’ (sampMatInfo.expiryM), and ‘Year of expiry’ (sampMatInfo.expiryY); | The date of the production, reported in sampMatInfo.prodD, sampMatInfo.prodM, and sampMatInfo.prodY, is later thanthe date of the expiry, reported in sampMatInfo.expiryD, sampMatInfo.expiryM, and sampMatInfo.expiryY; | error | sampMatInfo.prodD sampMatInfo.prodM sampMatInfo.prodY sampMatInfo.expiryD sampMatInfo.expiryM sampMatInfo.expiryY |  | sysRecord |
| GBR084 | The date of the production, reported in ‘Day of production’ (sampMatInfo.prodD), ‘Month of production’ (sampMatInfo.prodM), and ‘Year of production’ (sampMatInfo.prodY), cannot be later than the date of the sampling, reported in ‘Day of sampling’ (sampD), ‘Month of sampling’ (sampM), and ‘Year of sampling’ (sampY); | The date of the production, reported in sampMatInfo.prodD, sampMatInfo.prodM, and sampMatInfo.prodY, is later than the date of the sampling, reported in sampD, sampM, and sampY; | error | sampMatInfo.prodD sampMatInfo.prodM sampMatInfo.prodY sampD sampM sampY |  | sysRecord |
| GBR085 | The date of the production, reported in ‘Day of production’ (sampMatInfo.prodD), ‘Month of production’ (sampMatInfo.prodM), and ‘Year of production’ (sampMatInfo.prodY), cannot be later than the date of the analysis, reported in ‘Day of analysis’ (analysisD), ‘Month of analysis’ (analysisM), and ‘Year of analysis’ (analysisY); | The date of the production, reported in sampMatInfo.prodD, sampMatInfo.prodM, and sampMatInfo.prodY, is later than the date of the analysis, reported in analysisD, analysisM, and analysisY; | error | sampMatInfo.prodD sampMatInfo.prodM sampMatInfo.prodY analysisD analysisM analysisY |  | sysRecord |
| GBR086 | The date of the sampling, reported in ‘Day of sampling’ (sampD), ‘Month of sampling’ (sampM), and ‘Year of sampling’ (sampY), cannot be later than the analysis, reported in ‘Day of analysis’ (analysisD), ‘Month of analysis’ (analysisM), and ‘Year of analysis’ (analysisY); | The date of the sampling, reported in sampD, sampM, and sampY, is later than the date of the analysis, reported in analysisD, analysisM, and analysisY; | error | sampD sampM sampY analysisD analysisM analysisY |  | sysRecord |
| GBR087 | The date of the arrival in the laboratory, reported in ‘Arrival Day’ (sampInfo.arrivalD), ‘Arrival Month’ (sampInfo.arrivalM), and ‘Arrival Year’ (sampInfo.arrivalY), cannot be later than the date of the completion of the analysis, reported in ‘Completion day of analysis’ (sampAnInfo.compD), ‘Completion month of analysis’ (sampAnInfo.compM), and ‘Completion year of analysis’ (sampAnInfo.compY); | The date of the arrival in the laboratory is later than the date of the completion of the analysis, reported in sampAnInfo.compD, sampAnInfo.compM, and sampAnInfo.compY; | error | sampInfo.arrivalD sampInfo.arrivalM sampInfo.arrivalY sampAnInfo.compD sampAnInfo.compM sampAnInfo.compY |  | sysRecord |
| GBR088 | The date of the sampling, reported in ‘Day of sampling’ (sampD), ‘Month of sampling’ (sampM), and ‘Year of sampling’ (sampY), cannot be later than the date of the isolation, reported in ‘Isolation day’ (isolInfo.isolD), ‘Isolation month’ (isolInfo.isolM), and ‘Isolation year’ (isolInfo.isolY); | The date of the sampling, reported in sampD, sampM, and sampY, is later than the date of the isolation, reported in isolInfo.isolD, isolInfo.isolM, and isolInfo.isolY; | error | sampD sampM sampY isolInfo.isolD isolInfo.isolM isolInfo.isolY |  | sysRecord |
| GBR89 | The date of the slaughtering, reported in ‘Day of slaughtering’ (sampEventInfo.slaughterD), ‘Month of slaughtering’ (sampEventInfo.slaughterM), cannot be later than the date of the sampling, reported in ‘Day of sampling’ (sampD), ‘Month of sampling’ (sampM), and ‘Year of sampling’ (sampY); | The date of the slaughtering, reported in sampEventInfo.slaughterD, sampEventInfo.slaughterM, and sampEventInfo.slaughterY, is later than the date of the sampling, reported in sampD, sampM, and sampY; | error | sampEventInfo.slaughterD sampEventInfo.slaughterM sampEventInfo.slaughterY sampD sampM sampY |  | sysRecord |
| GBR090 | If the value in the data element ‘Coded description of the parameter’ (paramCode.param) is different from ‘Not in list’ (RF-XXXX-XXX-XXX), then the combination of values in the data elements ‘Coded description of the parameter’ (paramCode.param), ‘Sample taken identification code’ (sampId), and ‘Sample analysed portion sequence’ (anPortSeq) must be unique, i.e. only one result per parameter for each sample analysed portion per sample; | More than one paramCode.param per sample analysed portion for one sample is reported, though paramCode.param is different from ‘Not in list’ (RF-XXXX-XXX-XXX); | error | paramCode.param sampId anPortSeq |  | sysDataCollection sysOrganisation |

Syntax for business rules : quick reference

The section that follows shows each business rule statement and its syntax. Refer to Section 10.4, “Syntax of business rules” for the specification of each statement.

BUSINESS RULE CODE (10.4.3)



BUSINESS RULE TYPE (10.4.10)



DESCRIPTION (10.4.4)



INFORMATIVE MESSAGE (10.4.5)



INFORMATIVE TYPE (10.4.6)



STATUS (10.4.7)



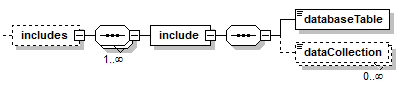
LAST UPDATE (10.4.8)



CHECKED DATA ELEMENTS (10.4.9)



INCLUDE (10.4.12)



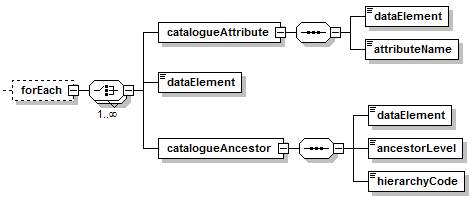
APPLY TO (10.4.13)



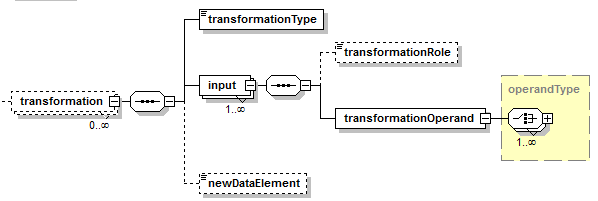
IGNORE NULL VALUES (10.4.19)



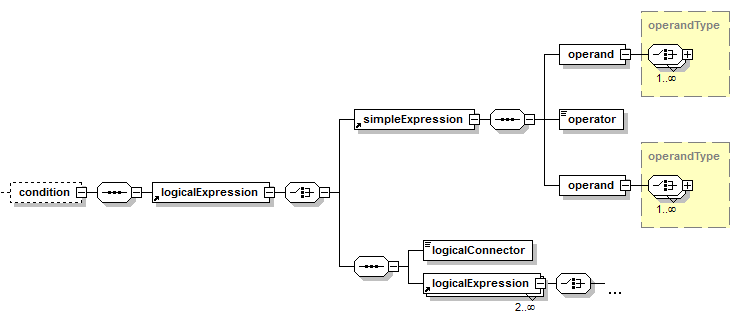
FOR EACH (10.4.14)



TRANSFORMATION (10.4.15)



CONDITION (10.4.16)



VERIFY (10.4.16)

