White Paper

ECCAIRS Data Bridge

Version 2.0 June 2015

ABSTRACT

The ECCAIRS[1] **Data Bridge** is an XML based, and XSD compliant, way to feed data into an ECCAIRS system. The transfer format consists of one or more XML files, each containing the data of a single occurrence in a specific format defined by the current taxonomy. These XML files are validated against an XSD schema that describes the involved taxonomy structure and the related data. The XML files are packaged into a compressed file (with the E5X extension), which is sent to the ECCAIRS system. The ECCAIRS software converts the compressed file into the E5F format so that the occurrence(s) can be read in the native containers of the ECCAIRS system (file or database).

This White Paper describes the XML format and explains how to create the package. Purpose of this document is to provide 3rd parties with a W3C compliant specification so that these parties can produce and verify correctness of occurrence data. The related XSD schema is provided with each release of an ECCAIRS extension.

This White paper is targeted at developers interested in creating XML files, packaged in an E5X package, to be fed into an ECCAIRS instance. A good knowledge of the XML language and the XSD Schema Definitions is required.

Contents

[1] In this White Paper ECCAIRS will refer to ECCAIRS 5. Previous versions of ECCAIRS do not support the Data Bridge functionality.

1. INTRODUCTION

The ECCAIRS Data Bridge is a part of the ECCAIRS software able to accept occurrence data provided in an XML based, and W3C compliant, format. This XML data is converted into the standard ECCAIRS E5F format[2].

Before the data is accepted by an ECCAIRS system, it must be successfully validated with respect to the taxonomy deployed in the current ECCAIRS system. This validation process verifies the semantics of the XML file, the existence of attributes and entities and the correctness of the provided attribute values following the definitions given in an XSD schema pertaining to the taxonomy.

The XSD schema, which is made available also independently from the ECCAIRS software, can be used as a specification of the supported input format. Since the supported XML files and the related XSD schema are W3C compliant, occurrence data can be produced without the need to install the ECCAIRS software. The XML files can be generated using standard tools, technologies on various software and operating system platforms.

By verifying the XML files against the XSD schema, a Data-Provider can be sure that the data provided to an ECCAIRS system deploying the same taxonomy as the taxonomy from which the XSD schema was derived will be accepted by a data integrator. This verification process can take place at the site of the Data-Provider and does not require any specific ECCAIRS software.

In the ECCAIRS system, the acceptance of the Data Bridge format is implemented as an elementary function of the software. For this reason it can be used in various modules. In the first releases reading of the Data Bridge data will be possible in the Data Manager, the Windows 3rd party API and the Webservices API.

In Figure 1 the ECCAIRS architecture is depicted. The ECCAIRS Common framework is the standard software used in any ECCAIRS instance. The standard ECCAIRS Extension (e.g. the Aviation Extension) provides 4 features: The taxonomy, the user interface, if required some extra functions and the XSD schema derived from the taxonomy.

The native storage format (E5F) is supported in the database and in the E5F files. These formats can be read as well as written by the ECCAIRS software. The Data Bridge storage format is supported in the E5X file which can only be read. E5X files are validated by the ECCAIRS Common Framework using the XSD while the XSD can also be used in 3rd party software environments to validate E5X files produced.

Customisation of the deployed system can be done by customising the taxonomy (if this supports custom attributes) and/or the User Interface. In addition, via appropriate API's and Add-Ins, the ECCAIRS functionality can be extended or ECCAIRS can be integrated in other environments. These customisations are out of the scope of this document but it should be noted that the Data Bridge only supports the standard taxonomy elements, not the customised taxonomy.

[2] See the White Paper: ECF E5F FILE FORMAT (Version 2.0 February 2013)

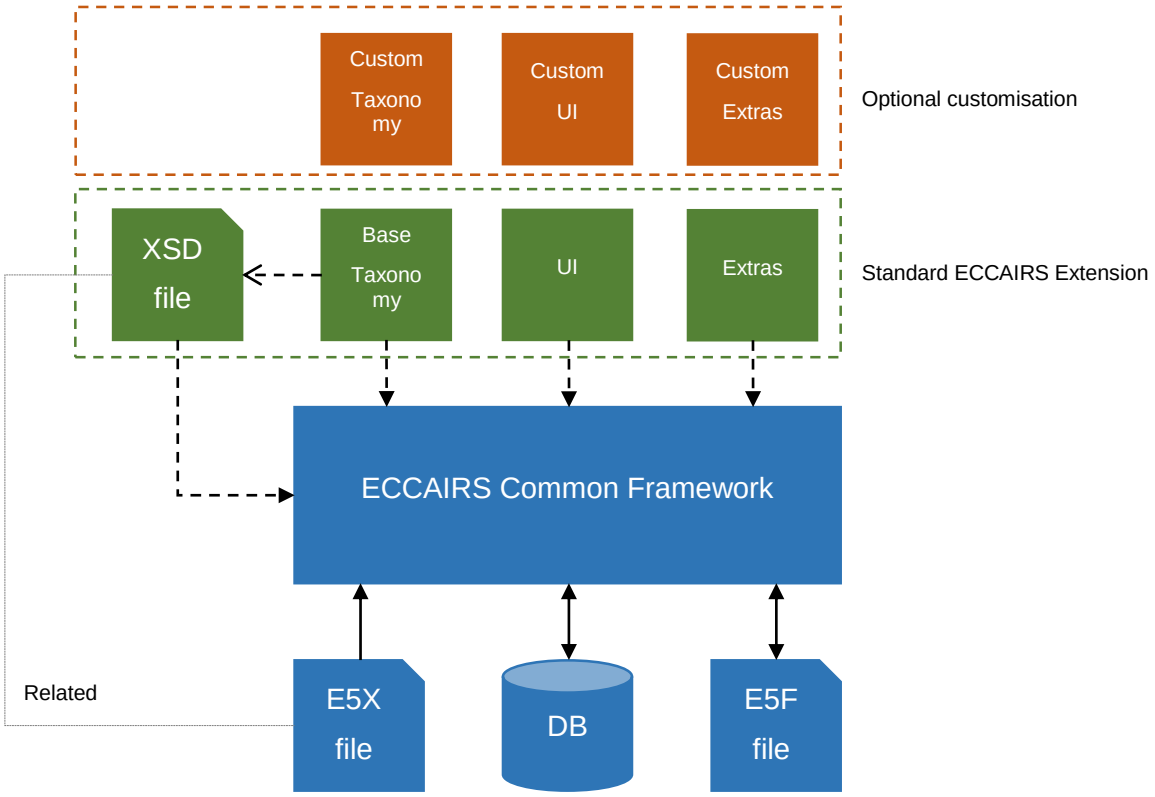

Figure 1: The ECCAIRS architecture

Both the XML format and the XSD structure follows the standards defined by W3C, which can be found here: http://www.w3.org/2001/XMLSchema

In this White Paper, the examples presented, both for the XSD as well as for the XML, are derived from the ECCAIRS Aviation Taxonomy. This is not to say that other taxonomies are not supported. Since the Data Bridge is part of the ECCAIRS Common Framework, the functionality is available to any deployment type.

Also note that the examples in this White Paper should not be taken as a specification. In the end it is the XSD which provides the full specification of the format of the data to be provided.

To clearly distinct the XML examples from the XSD definitions, in this White Paper the XSD definitions are coloured blue and the XML examples are coloured brown.

2. THE DATA BRIDGE

2.1. Technologies and standards

The XML format supported by the ECCAIRS Data Bridge follows the W3C standard for XML files (http://www.w3.org/2001/XMLSchema).

The XSD (XML Schema Definition language) format supported by the ECCAIRS Data Bridge follows the W3C standard for XSD files (http://www.w3.org/TR/xmlschema11-1/).

The XML file can be generated and validated inside different software environments. The following environments have been used for compliance testing:

- XML Spy
- .Net framework with C# language
- Eclipse

2.2. The package

An ECCAIRS Data Bridge file (with the extension .E5X) consists of a zipped[3] file in which all ECCAIRS occurrences are stored in separate XML files. For this reason, each XML Occurrence file should have a unique filename. As long as these XML filenames are unique within the .E5X package, the actual names of these files are irrelevant. The XML and XSD specifications apply for the individual XML Occurrence files, not for the packaging file.

The contents of the simplest ECCAIRS Data Bridge file (.E5X) containing just one occurrence could thus be:

Simple.E5X (zipfile)

- Occ1.xml (file)

An ECCAIRS Data Bridge file (.E5X) containing multiple occurrences would thus be:

Multiple.E5X (zipfile)

- Occ1.xml (file)
- Occ2.xml (file)
- …
- Occn.xml (file)

In principle, the ECCAIRS Data Bridge files support also attachments to ECCAIRS occurrences[4]. The simplest ECCAIRS Data Bridge file (.E5X) containing just one occurrence with one attachment (pdf file for example) would be packaged as this:

SimpleWithAtt.E5X (zipfile)

- Occ1.xml (file)
- Occ1 (folder)
    - Att1.pdf

The attachment is linked to one of the attributes in the occurrence of type Blob, which must point to the filename of the attachment (Att1.pdf in this case).

[3] The chosen compression format is the zip format since it is a standard supported in most IT platforms.

[4] It could be that attachment support will not be available in the first releases of ECCAIRS supporting ECCAIRS Data Bridge files.

An ECCAIRS Data Bridge file (.E5X) containing multiple occurrences with multiple attachments would thus be (for example):

MultipleWithAtt.E5X (zipfile)

- Occ1.xml (file)
- Occ1 (folder)
    - Att1.pdf (file)
    - Att2.pdf (file)
- Occ2.xml (file)
- Occ2 (folder)
    - Att3.pdf (file)
    - Att1.pdf (file)
    - Att4.pdf (file)
- …
- Occn.xml (file)

In this example, the first Occurrence has two attachments, the second occurrence has three attachments and the last occurrence has no attachments at all. Note that while the names of the XML Occurrence files must be unique within the package, the names of the attachments must be unique only within the scope of the occurrence (i.e. within the folder they are stored).

ECCAIRS Data Bridge files can be created and opened using the standard ZIP compression functions available in all IT platforms. A simple test to see if the packaging has been done correctly would be to rename the .E5X file into a .ZIP fiel and open this file using the standard Windows explorer.

2.3. The XSD files

Each ECCAIRS extension will provide an XML Schema Definition (XSD), which is used by the ECCAIRS software to validate the XML files inside an .E5X Data Bridge package. The same XSD can be used by Data-Providers to validate their output before sending it to their Data-Integrator. It is important to be aware that:

- The provided XSD schema is made up of several files:
    - Schema.xsd
      the master schema containing also references to the other XSD schemas. This XSD file is the one to be used when validating an XML occurrence file.
    - ECCAIRS_datatypes.xsd
      contains the general definitions for the XML structures used to store attribute values.
    - Several XSD files
      containing the list of the admitted values for attributes with a ‘predefined value list’.

- The XSD files contain ‘annotations’ that describe the meaning of each element. These annotations only serve to offer better comprehensibility, they are not used for verification and/or validation.

- The XSD Schemas are generated from an ECCAIRS specific taxonomy. This guarantees that all the validated XML occurrences can be converted into an ECCAIRS Occurrence related to the same taxonomy version.

- XML occurrences will also be accepted if the ECCAIRS target system uses a newer version of the same taxonomy (backwards compatibility)[5].

[5] Note that backwards compatibility requires a correct configuration of the target repository; in particular the conversion rules need to be added.

2.4. The XML Occurrence file

How exactly the XML occurrence files are created by the Data-Provider is out of the scope of this White Paper. Typically, we assume that the XML files are created from a proprietary reporting environment[6].

The basic structure of the XML file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<SET>
    <OCCURRENCE>
    </OCCURRENCE>
</SET>
```

The XSD is encoded using utf8 encoding so that it can support a large character set. Also the XML file that contains the occurrence has to be encoded using utf8. This encoding type has to be specified in the XML file prologue (see above).

The XML uses several tags. Some of these are taxonomy independent and are used to identify specific information or to group homogeneous elements. These are:

- SET
- OCCURRENCE[7]
    - ENTITIES
    - ATTRIBUTES
    - LINKS

The other tags are taxonomy dependent and they are generated from the ECCAIRS entities and attributes that are defined in a specific ECCAIRS taxonomy.

[6] It should be noted that Data-Providers using ECCAIRS as their reporting system can provide their data in ECCAIRS' native E5F format.

[7] The OCCURRENCE tag could be named different since it is depending on the taxonomy. For the Aviation taxonomy the root entity to store all data is the OCCURRENCE. For other taxonomies it could be different.

2.5.SET

The SET tag is the root node of the XML Occurrence file. The SET tag contains several attributes defining:

- Applicable namespace and schema attributes
  - xmlns (Data Bridge main namespace)
  - xmlns:xsi (XML Schema Instance namespace)
  - xmlns:dt (ECCAIRS data types definition namespace)
  - xsi:schemaLocation (Schema location)

- ECCAIRS system, taxonomy and domain attributes.
  - Version (the XSD structure version having the fixed value 1.0.0.0 for the current version)
  - TaxonomyName (the name of current taxonomy)
  - TaxonomyVersion (the version of current taxonomy)
  - Domain (the named subset of taxonomy attributes for the XML Data Bridge)

The values for the above attributes is defined in the XSD file provided and might change during time.

In addition the SET tag contains an element:

- OCCURRENCE: this tag name corresponds to the current taxonomy root entity (OCCURRENCE in the case of the Aviation taxonomy but it can be different for other taxonomies) and it is the starting point of the data structure containing all the data.

The XSD block that defines the SET complexType is:

```
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:sequence>
		<xs:element name="Occurrence" type="db:Occurrence" minOccurs="1" maxOccurs="1" />
	</xs:sequence>
	<xs:attribute name="TaxonomyName" type="xs:string" use="required" fixed="ECCAIRS Aviation" />
	<xs:attribute name="TaxonomyVersion" type="xs:string" use="required" fixed="2.8.0.0" />
	<xs:attribute name="Domain" type="xs:string" use="required" fixed="RIT" />
	<xs:attribute name="Version" type="xs:string" use="required" fixed="1.0.0.0" />
</xs:complexType>
```

An example of a simple occurrence in the XML structure is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2015 sp2 (http://www.altova.com)-->
<SET xmlns="http://eccairsportal.jrc.ec.europa.eu/ECCAIRS5_dataBridge.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:dt="http://eccairsportal.jrc.ec.europa.eu/ECCAIRS5_dataTypes.xsd"
        Version="1.0.0.0"
        TaxonomyName="ECCAIRS Aviation"
        TaxonomyVersion="2.8.0.0"
        Domain="RIT"
        xsi:schemaLocation="http://eccairsportal.jrc.ec.europa.eu/ECCAIRS5_dataBridge.xsd">
        <Occurrence entityId="24">
                <ATTRIBUTES>
                        <Local_Date attributeId="433">2014-08-13</Local_Date>
                </ATTRIBUTES>
                <ENTITIES>
                        <Aircraft ID="ID4" entityId="4">
                                <ATTRIBUTES>
                                        <Manufacturer_Model attributeId="21" AdditionalText=""
                                           AdditionalTextEncoding="xs:string">18370</Manufacturer_Model>
                                        <A_C_Flight_Level>150</A_C_Flight_Level>
                                        <Total_Cycles_A_C attributeId="33">10</Total_Cycles_A_C>
                                        <Call_Sign attributeId="54">ACR500</Call_Sign>
                                        <Serial_Number attributeId="254">EPO879VF</Serial_Number>
                                </ATTRIBUTES>
                        </Aircraft>
                </ENTITIES>
        </Occurrence>
</SET>
```

2.6.OCCURRENCE

The OCCURRENCE node[8] is the container for the real data stored in the ECCAIRS Occurrence. The OCCURRENCE is nothing else than a set of logically structured and interrelated Entities, Attributes and Links (using the ENTITIES, ATTRIBUTES and LINKS tags) following the definition of the current taxonomy.

The root element corresponds to the taxonomy's root entity and is represented by the OCCURRENCE tag. For each entity defined in the taxonomy an XSD complex type has been created inside the XSD schema. The entity type name corresponds to the entity synonym defined in the ECCAIRS taxonomy.

Each physical entity (not linked) type is composed by three main nodes:

- ATTRIBUTES
  containing the definition of the attributes that belong to the current entity (if the current entity doesn't have any attributes this node doesn't exist)
- ENTITIES
  containing the child entities that belong to the current entity (if the current entity doesn't have any child entities this node doesn't exist)
- LINKS
  containing the linked child entities belonging to the current entity (if the current entity doesn't have any linked child entities this node doesn't exist)

For each entity the identifier in the ECCAIRS taxonomy is specified in the XSD schema using a xs:attribute as follows:

```
<xs:attribute name="entityId" type="xs:string" fixed="4" />
```

This attribute is not mandatory, it can be omitted.

If an entity is linked somewhere in the taxonomy structure its XSD definition also contains an attribute called ID and defined as follows:

```
<xs:attribute name="ID" type="xs:ID" />
```

The type xs:ID is used for identifiers and has the following requirements:

- It uniquely identifies an element in an XML document and thus it's value must be unique within an XML instance
- Its value must be an NCName meaning that it must start with a letter or underscore, and can only contain letters, digits, underscores, hyphens, and periods

We suggest using positive numbers to identify each entity. An example of a valid XML for the entity Aircraft is:

```
<Aircraft ID="ID4" entityId="4">
        <ATTRIBUTES>
                <Current_Traffic_Type attributeId="29">2</Current_Traffic_Type>
                <Aircraft_Category attributeId="32">7</Aircraft_Category>
                <Total_Cycles_A_C attributeId="33">1000</Total_Cycles_A_C>
                <ATS_Route_Name>AArrtsfajj</ATS_Route_Name>
                ….
        </ATTRIBUTES>
        <ENTITIES> ….</ENTITIES>
        <LINKS>….</LINKS>
</Aircraft>
```

In the next sections, the ATTRIBUTES, ENTITIES and LINKS nodes are explained in more detail.

[8] OCCURRENCE is the name in the Aviation Taxonomy, in other taxonomies this tag could be named differently.

2.6.1. Attribute Types

The node ATTRIBUTES inside an entity contains all the available attributes under that entity. According to its XSD definition, it is possible to insert defined attributes (it is not necessary to insert all the defined attributes) and they must follow the attribute declaration order of the XSD (since they are contained inside a XSD sequence node). According to the XSD generation algorithm this order stands for the attribute identifier ascending order.

Each attribute definition in the XSD schema specifies the name, cardinality, type and ECCAIRS Identifier of the attribute. All those properties are derived from the attribute definition in the taxonomy. The ECCAIRS Identifier has to be included in the XML with its fixed value as defined in the XSD.

The available ECCAIRS data types are:

Type	Description
Alphanumeric	Alphanumeric strings with a maximum length of 255 characters
Blob	A file
Code	A value which represents a single entry of a predefined list
Code and additional text	A value which represents a single entry of a predefined list plus possibly an additional text
Code or alternative text	A value which represents a single entry of a predefined list or an alternative text
Date	Date
DateTime	Date and Time
Decimal	Decimal numer
ECCAIRSDataLink	Reference to an instance out of a defined ECCAIRS repository
Latitude	Latitude
Longitude	Longitude
Number	Numeric Integer values
OccurrenceBlob	A structure containing a single element from an ECCAIRS repository (comparable to a ‘record’ from a database)
Time	Time
Text	Text with a possible length of more than 255 characters

In the following paragraphs, the XSD structure and an example of corresponding valid XML will be explained for each attribute data type using some attributes from the Aviation taxonomy.

The attribute ‘attributeId’ that is contained inside each attribute definition has a fixed value that corresponds to the attribute identifier inside the taxonomy. It is not mandatory and it can be omitted.

2.6.1.1. Alphanumeric

The attribute Headline (601) is Alphanumeric with a maximum size set to 255; it is defined in the XSD as follows:

```
<xs:element name="Headline" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="dt:string255max">
                                <xs:attribute name="attributeId" type="xs:string" fixed="601" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

The type ‘string255max’ is defined inside ECCAIRS5_dataTypes.xsd since it is a general type used in several ECCAIRS taxonomies. The prefix dt is defined inside the Schema root node and stands for objects defined inside ECCAIRS5_dataTypes.xsd.

The attribute definition is based on the following general type:

```
<xs:simpleType name="string255max">
        <xs:restriction base="xs:string">
                <xs:maxLength value="255"/>
        </xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<Headline attributeId="601" >Helicopter crashed during emergency landing</Headline>
```

The attribute Airspace name (14) is also Alphanumeric but it has a different maximum size limit (20), so it is defined as follows using an ad-hoc created type:

```
<xs:element name="Airspace_Name" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="db:Base_Airspace_Name">
                                <xs:attribute name="attributeId" type="xs:string" fixed="14" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

The attribute definition is based on the following base type:

```
<xs:simpleType name="Base_Airspace_Name" xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:restriction base="xs:string">
                <xs:maxLength value="20" />
        </xs:restriction>
</xs:simpleType>
```

The base type name is ‘Base_Airspace_Name’. Base type definitions are grouped after the entity structure definition.

An example of valid XML is:

```
<Airspace_Name attributeId="14" >KOTVRDOVICE</Airspace_Name>
```

If an attribute value has to be uppercase, this constraint is specified in the XSD as in the following example for attribute Runway_Identifier (499):

```
<xs:element name="Runway_Identifier" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="db:Base_Runway_Identifier">
				<xs:attribute name="attributeId" type="xs:string" fixed="499" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following base type:

```
<xs:simpleType name="Base_Runway_Identifier" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:restriction base="xs:string">
		<xs:maxLength value="3" />
		<xs:pattern value="([A-Z0-9])*" />
	</xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<Runway_Identifier attributeId="499">03L</ Runway_Identifier>
```

2.6.1.2. Blob

The attribute Attachments (793) has type ECCAIRS Resource Locator and is defined in the XSD using the ResourceLocator type (defined in ECCAIRS5_dataTypes.xsd) as follows:

```
<xs:element name="Attachments" minOccurs="0" maxOccurs="unbounded"
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:complexContent>
			<xs:extension base="dt:ResourceLocator">
				<xs:attribute name="attributeId" type="xs:string" fixed="793" />
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:complexType name="ResourceLocator">
	<xs:sequence>
		<xs:element name="FileName" type="xs:string" minOccurs="1" maxOccurs="1"/>
		<xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
	</xs:sequence>
</xs:complexType>
```

An example of valid XML is:

```
<Attachments attributeId="793">
	<dt:FileName>PI EC_AS532AL 2800 Eurocopter 25_07_2012.pdf</dt:FileName>
	<dt:Description>TC holder report</dt:Description>
</Attachments>
```

2.6.1.3. Code

The attribute Light conditions(168) has type Code, meaning it is associated to a predefined value list. It is defined in the XSD as follows:

```
<xs:element name="Light_Conditions" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="db:VL168_5_0_1_1">
                                <xs:attribute name="attributeId" type="xs:string" fixed="168" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

The type name contains the associated value list identifier. The value list is defined in a separated file called VL168_5_0_1_1.xsd that is included in the XSD through the xs:include statement. The db prefix is defined inside the Schema root node: it indicates the standard namespace that is the main namespace also for the value lists definition.

An example of valid XML is:

```
<Light_Conditions attributeId="168">2</Light_Conditions>
```

The only allowed values for Code attributes are the ones defined in the associated value list. Each value list contains the value identifiers as defined in the taxonomy. Through the documented schema it is easy to see the corresponding description, detailed description and explanation for each value contained in a value list.

The naming convention of the value list XSD files is as follows (e.g. VL168_5_0_1_1.xsd):

The name consists of 5 parts separated by the '_' character.

1st part (VL168): "VL" followed by an identification number, making the filename unique

2nd part (5) : Data type of attribute(s) using the value list, the three possible values are:
- 5 Code
- 12 Code and Additional Text
- 13 Code or Alternative Text

3rd part (0): 0 means the complete value list of the attribute is included
nn means the sub branch of the value list is included starting with value nn

4th part (1): 1 means all value levels are included
0 means only a subset of levels has been included indicated by the 5th part

5th part (1): the number of levels included

Note that these names are only used for assisting developers to understand the origin of the Value List with respect to the original ECCAIRS taxonomy. At the level of the XSD the file names are arbitrary apart from that they must of course be unique.

2.6.1.4. Code and Additional Text

The attribute State/area of occ (454) has type Code and Additional Text and is associated to a predefined value list, just like Code. It is defined in the XSD as follows:

```
<xs:element name="State_Area_Of_Occ" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="db:VL1090_12_0_1_3">
                                <xs:attribute name="attributeId" type="xs:string" fixed="454" />
                                <xs:attribute name="AdditionalText" />
                                <xs:attribute name="AdditionalTextEncoding">
                                        <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                        <xs:enumeration value="xs:string" />
                                                        <xs:enumeration value="xs:base64Binary" />
                                                </xs:restriction>
                                        </xs:simpleType>
                                </xs:attribute>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

Like Code, it based on a value list so that the allowed values are those defined in the corresponding value list. In addition, it is possible to add some text inside a related attribute called AdditionalText. The text can be either a standard xs:string or it can be encoded as base64 [9]. The text format has to be specified inside the AdditionalTextEncoding.

An example of valid XML using base64 encoding for the additional text is:

```
<State_Area_Of_Occ attributeId="454" AdditionalText="OECAHS9993827"
AdditionalTextEncoding="xs:base64Binary">1</State_Area_Of_Occ>
```

An example of valid XML using string for the additional text is:

```
<State_Area_Of_Occ attributeId="454" AdditionalText="Close to Paris"
AdditionalTextEncoding="xs:string">1</State_Area_Of_Occ>
```

[9] The conversion of a string in format base64 should be done in the following way:

- Conversion of the string in a byte array using UTF8 encoding
- Conversione of the byte array in base64 using base64 transfer encoding for MIME (RFC 2045)

2.6.1.5. Code or Alternative Text

The attribute FIR UIR Name (16) has type Code or Additional Text and is associated to a predefined value list, just like type Code and Code and Additional Text. It is defined in the XSD as follows:

```
<xs:element name="FIR_UIR_Name" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="db:VL1084_13_0_1_2">
                                <xs:attribute name="attributeId" type="xs:string" fixed="16" />
                                <xs:attribute name="AlternativeText" />
                                <xs:attribute name="AlternativeTextEncoding">
                                        <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                        <xs:enumeration value="xs:string" />
                                                        <xs:enumeration value="xs:base64Binary" />
                                                </xs:restriction>
                                        </xs:simpleType>
                                </xs:attribute>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

It is possible to add an alternative text through the related AlternativeText attribute and specify the text format inside the AlternativeTextEncoding attribute, just as explained for AdditionalTextEncoding attribute.

Example of valid XML where there is only an alternative text is:

```
< FIR_UIR_Name attributeId="16"  AlternativeText="A kind of FIR" AlternativeTextEncoding="xs:string"></ FIR_UIR_Name >
```

Example of valid XML where there is only an alternative text is:

```
< FIR_UIR_Name attributeId="16"  AlternativeText="" AlternativeTextEncoding="xs:string">2</ FIR_UIR_Name >
```

If an alternative text is added, an eventual coded value inserted will be ignored (as in the example below where the value 2 is not relevant).

```
< FIR_UIR_Name attributeId="16"  AlternativeText="A kind of FIR" AlternativeTextEncoding="xs:string">2</ FIR_UIR_Name >
```

2.6.1.6. Date

The attribute UTC Date (477) has type Date and is defined inside the XSD file with the standard type xs:date, as follows:

```
<xs:element name="UTC_Date" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="xs:date">
                                <xs:attribute name="attributeId" type="xs:string" fixed="477" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

An example of valid XML is:

```
<UTC_Date attributeId="477">2007-08-13</ UTC_Date>
```

The date has to be inserted in the standard XML format, which is: YYYY-MM-DD

2.6.1.7. Datetime

The attribute Modification date (422) has type DateTime and is defined inside the XSD file as follows:

```
<xs:element name="Modification _Date" minOccurs="0" maxOccurs="1">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="dt:DateTime">
                                <xs:attribute name="attributeId" type="xs:string" fixed="422" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

The attribute definition is based on the following base type:

```
<xs:simpleType name="DateTime">
        <xs:restriction base="xs:dateTime">
                <xs:pattern value=".+T[^Z+-]+"/>
        </xs:restriction>
 </xs:simpleType>
```

An example of valid XML is:

```
<Modification_Date attributeId="422">2001-12-17T09:30:47</Modification_Date>
```

The datetime has to be inserted in the standard XML format that is: YYYY-MM-DDTHH-MM-SS.

2.6.1.8. Decimal

Attributes of Decimal type can have a measurement unit or not. In addition, they can have a specific lower limit, upper limit and maximum decimal digits.

Attribute Risk Level (940) is of a Decimal type, has specific limits but no unit of measurement. It is defined in the XSD file as follows:

```
<xs:element name="Risk_Level" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="db:Base_Risk_Level">
				<xs:attribute name="attributeId" type="xs:string" fixed="940" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following base type:

```
<xs:simpleType name="Base_Risk_Level" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:restriction base="xs:decimal">
		<xs:minInclusive value="0" />
		<xs:maxInclusive value="100" />
		<xs:fractionDigits value="50" />
	</xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<Risk_Level attributeId=”940”>11.53</Risk_Level>
```

The XML standard decimal separator is the point character.

Xs:fractionDigits value=”50” indicates that the maximum number of digits that can be used is 50.

Attribute Dew point (85) is also of a Decimal type, with specific limits and measurement units. A base type is created to define limits, while unit of measurement is specified in attribute definition. The following XSD block defines the Dew point attribute:

```
<xs:element name="Dew_Point" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="db:Base_Dew_Point">
				<xs:attribute name="Unit" use="required" fixed ="C" />
				<xs:attribute name="attributeId" type="xs:string" fixed="85" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following base type:

```
<xs:simpleType name="Base_Dew_Point" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:restriction base="xs:decimal">
		<xs:minInclusive value="-100" />
		<xs:maxInclusive value="100" />
		<xs:fractionDigits value="50" />
	</xs:restriction>
</xs:simpleType>
```

An attribute called “Unit” is defined to set the attribute unit of measurement. Its fixed value corresponds to attribute default unit. An example of valid XML is:

```
<Dew_Point attributeId=”85” Unit="C">-8.453</Dew_Point>
```

2.6.1.9. EccairsDataLink

The attribute Recommendation link (788) has type Eccairs Data Link and is defined in the XSD using the ECCAIRS DataLink type (defined in ECCAIRS5_dataTypes.xsd) as follows:

```
<xs:element name=" Recommendation_Link " minOccurs="0" maxOccurs="1"
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:complexContent>
                        <xs:extension base="dt:DataLink ">
                                <xs:attribute name="attributeId" type="xs:string" fixed="788" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:complexType name="DataLink">
        <xs:sequence>
                <xs:element name="EccairsNumber" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="ResponsibleEntity" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="OccurrenceKey" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="CreationDate" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
                <xs:element name="LastModificationDate" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
                <xs:element name="Repository" type="xs:string" minOccurs="0" maxOccurs="1"/>
                <xs:element name="TaxonomyKey" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="TaxonomyName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="TaxonomyVersion" type="xs:string" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
</xs:complexType>
```

An example of valid XML is:

```
<Recommendation_Link attributeId=”788”>
        <ec:EccairsNumber>128/2014</ec:EccairsNumber>
        <ec:ResponsibleEntity>1234</ec:ResponsibleEntity>
        <ec:OccurrenceKey>FG56LCDE0000000001234FGH00000000</ec:OccurrenceKey>
        <ec:CreationDate>2001-12-17T09:30:47</ec:CreationDate>
        <ec:LastModificationDate>2001-12-17T09:30:47</ec:LastModificationDate>
        <ec:Repository>SRIS</ec:Repository>
        <ec:TaxonomyKey>AG87ASDG4920UI6500DFJKUTF5629481</ec:TaxonomyKey>
        <ec:TaxonomyName>SRIS</ec:TaxonomyName>
        <ec:TaxonomyVersion>5.6.1.1</ec:TaxonomyVersion>
</Recommendation_Link>
```

2.6.1.10. Latitude

The attribute Latitude of occ (439) is of type Latitude and is defined inside the XSD file using the ECCAIRS LatitudeDecimal type (defined in ECCAIRS5_dataTypes.xsd), as follows:

```
<xs:element name="Latitude_Of_Occ" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="dt:LatitudeDecimal">
				<xs:attribute name="attributeId" type="xs:string" fixed="439" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:simpleType name="LatitudeDecimal">
	<xs:restriction base="xs:double">
		<xs:minInclusive value="-90.0"/>
		<xs:maxInclusive value="90.0"/>
	</xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<Latitude_Of_Occ attributeId="439"> 51.511035 </Latitude_Of_Occ>
```

The latitude has to be inserted in the decimal format, identical to how it is being done by Google Maps.

The XML standard decimal separator is the point character.

The latitude in the above example corresponds to London UK.

2.6.1.11. Longitude

The attribute Longitude of occ (444) is of type Longitude and it is defined inside the XSD file using the ECCAIRS LongitudeDecimal type (defined in ECCAIRS5_dataTypes.xsd), as follows:

```
<xs:element name="Longitude_Of_Occ" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="dt:LongitudeDecimal">
				<xs:attribute name="attributeId" type="xs:string" fixed="444" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:simpleType name="LongitudeDecimal">
	<xs:restriction base="xs:double">
		<xs:minInclusive value="-180.0"/>
		<xs:maxInclusive value="180.0"/>
	</xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<Longitude_Of_Occ attributeId="444"> -0.161931 </Longitude_Of_Occ>
```

The longitude has to be inserted in the decimal format, identical to how it is being done by Google Maps.

The XML standard decimal separator is the point character.

The longitude in the above example corresponds to London UK.

2.6.1.12. Number

The attribute Total fatalities aircraft (459) is of type Number and has specific lower and upper limit values. In the XSD it is described using an ad-hoc created base type as follows:

```
<xs:element name="Total_Fatalities_Aircraft" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="db:Base_Total_Fatalities_Aircraft">
				<xs:attribute name="attributeId" type="xs:string" fixed="459" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following base type:

```
<xs:simpleType name="Base_Total_Fatalities_Aircraft" xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:restriction base="xs:int">
		<xs:minInclusive value="0" />
		<xs:maxInclusive value="9999" />
	</xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<Total_Fatalities_Aircraft attributeId="459">27</Total_Fatalities_Aircraft>
```

2.6.1.13. OccurrenceBlob

The attribute PiggyBack Data (838) is of type Eccairs Embedded Data and is defined in the XSD using the ECCAIRS EmbeddedData type (defined in ECCAIRS5_dataTypes.xsd) as follows:

```
<xs:element name=" Piggyback_Data " minOccurs="0" maxOccurs="1"
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:complexContent>
                        <xs:extension base="dt:EmbeddedData ">
                                <xs:attribute name="attributeId" type="xs:string" fixed="838" />
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:complexType name="EmbeddedData">
        <xs:sequence>
                <xs:element name="FileName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
                <xs:element name="EccairsNumber" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="ResponsibleEntity" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="OccurrenceKey" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="CreationDate" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
                <xs:element name="LastModificationDate" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
                <xs:element name="Repository" type="xs:string" minOccurs="0" maxOccurs="1"/>
                <xs:element name="TaxonomyKey" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="TaxonomyName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="TaxonomyVersion" type="xs:string" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
</xs:complexType>
```

An example of valid XML is:

```
<Piggyback_Data attributeId=”838”>
        <ec:FileName>PiggyBack_1.e5f</ec:FileName>
        <ec:Description>Occurrence piggy back</ec:Description>
        <ec:EccairsNumber>123/2015</ec:EccairsNumber>
        <ec:ResponsibleEntity>Luxembourg - CAA</ec:ResponsibleEntity>
        <ec:OccurrenceKey>000ABCDE0000000001234FGH00000000</ec:OccurrenceKey>
        <ec:CreationDate>2001-12-17T09:30:47Z</ec:CreationDate>
        <ec:LastModificationDate>2001-12-17T09:30:47Z</ec:LastModificationDate>
        <ec:Repository>AVIATION</ec:Repository>
        <ec:TaxonomyKey>3516ASDG4920470000DFJKUTF5629481</ec:TaxonomyKey>
        <ec:TaxonomyName>AVIATION</ec:TaxonomyName>
        <ec:TaxonomyVersion>2.4.0.0</ec:TaxonomyVersion>
</Piggyback_Data>
```

2.6.1.14. Time

The attribute UTC time (478) is of type Time and is defined in the XSD using the ECCAIRS EmbeddedData type (defined in ECCAIRS5_dataTypes.xsd) as follows:

```
<xs:element name="UTC_Time" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:simpleContent>
			<xs:extension base="dt:Time">
				<xs:attribute name="attributeId" type="xs:string" fixed="478" />
			</xs:extension>
		</xs:simpleContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:simpleType name="Time">
	<xs:restriction base="xs:time">
	  <xs:pattern value=".[^Z+-]+"/>
	</xs:restriction>
</xs:simpleType>
```

An example of valid XML is:

```
<UTC_Time attributeId="478">09:30:47</UTC_Time>
```

The time has to be inserted in the standard XML format, which is: hh:mm:ss

2.6.1.15. Text

The attribute Narrative text (425) is of type Text and is defined in the XSD using the ECCAIRS Text type (defined in defined in ECCAIRS5_dataTypes.xsd) as follows:

```
<xs:element name="Narrative_Text" minOccurs="0" maxOccurs="1">
	<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
		<xs:complexContent>
			<xs:extension base="dt:Text">
				<xs:attribute name="attributeId" type="xs:string" fixed="425" />
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
</xs:element>
```

The attribute definition is based on the following general type:

```
<xs:complexType name="Text">
	<xs:choice minOccurs="1" maxOccurs="1">
		<xs:element name="PlainText" type="xs:string"/>
		<xs:element name="EncodedText" type="xs:base64Binary"/>
	</xs:choice>
</xs:complexType>
```

According to Text definition it is possible to insert plain text or base 64 codified text.

Examples of valid XML are:

```
<Narrative_Text attributeId=”425”>
	<dt:PlainText>This is a plain text narrative</dt:PlainText>
</Narrative_Text>
```

or

```
<Narrative_Text attributeId=”425”>
	<dt:EncodedText>UjBsR09EbGhjZ0dTQUxNQUFBUUNBRU1tQ1p0dU1GU</dt:EncodedText>
</Narrative_Text>
```

It is not allowed to insert both PlainText and EncodedText within the same node.

2.6.2. Multivalues attributes

For attributes that support multiple values, no matter what the data type is, the cardinality is set through the MaxOccurs attribute in their XSD definition. For example as follows for Occurrence category (430):

```
<xs:element name="Occurrence_Category" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <xs:simpleContent>
                        <xs:extension base="db:VL430_5_0_1_1">
                                <xs:attribute name="attributeId" type="xs:string" fixed="430" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

Unbounded means that there are no limits in multiple values insertion.

So in the XML it is possible to add the Occurrence_category node several times, according to what specified in maxOccurs, as follows:

```
<Occurrence_Category attributeId=”430”>25</Occurrence_Category >
<Occurrence_Category attributeId=”430”>29</Occurrence_Category>
<Occurrence_Category attributeId=”430”>26</Occurrence_Category>
<Occurrence_Category attributeId=”430”>104</Occurrence_Category>
```

All the nodes that contain multiple values related to the same attribute have to be grouped together inside ATTRIBUTES node to respect the attribute order.

2.6.3. Entities

The node ENTITIES contains all the available child entities of the current entity. In the XML it is possible to insert only some entities (it is not necessary to insert all the defined entities). The entities have to be inserted in the order they are defined in the XSD schema since they are contained inside a XSD sequence node. According to the XSD generation algorithm this order stands for the entity identifier ascending order.

For each child entity is specified:

- Name, corresponding to the entity synonym defined in taxonomy
- Type, containing the type name that defines the entity inside the XSD file entity structure. Each child entity is defined as XSD complexType inside entity structure
- Cardinality

An example of XSD child entity definition inside the parent entity is:

```
<xs:element name="Aerodrome_General" type="db:Aerodrome_General" minOccurs="0" maxOccurs="unbounded" />
```

If maxOccurs >1 the entity node can be repeated several times inside the XML ENTITIES up to maximum limit. The entity nodes referred to the same entity have be grouped together inside the ENTITIES node.

An example of a valid XML is:

```
…
<ENTITIES>
        <Aerodrome_General entityId="1">
                <ATTRIBUTES>
                        <Aerodrome_latitude attributeId="1">-90</Aerodrome_latitude>
                        <Aerodrome_longitude attributeId="2">-180</Aerodrome_longitude>
                …
        </ Aerodrome_General>
        …
</ENTITIES>
…
```

The attribute ‘entityId that is contained inside each entity definition has a fixed value that corresponds to the entity identifier inside the taxonomy. It is not mandatory and it can be omitted.

2.6.4. Links

The node LINKS contains all the available linked child entities of the current entity. In the XML it is possible to insert only some entities (it is not necessary to insert all the defined entities). The order of the linked entities has to be the definition one.

For each child entity is specified:

- Name, corresponding to the entity synonym defined in taxonomy
- Type, containing the type name that defines the entity inside the XSD file entity structure. Each child entity is defined as XSD complexType inside entity structure
- Cardinality

An example of XSD linked child entity definition inside the parent entity is:

```
<xs:element name="Events" type="db:L-Events" minOccurs="0" maxOccurs="unbounded" />
```

Using the following ad-hoc type:

```
<xs:complexType name="L-Events">
        <xs:attribute name="REF" type="xs:IDREF" use="required"/>
</xs:complexType>
```

This complex type contains a mandatory attribute called REF with xs:IDREF type. This is used to refer to a physical entity and it has to contain the linked entity identifier that is contained in that entity ID attribute

If maxOccurs > 1 the linked entity node can be repeated several times inside the XML LINKS up to maximum limit. The entity nodes referring to the same linked entity have to be grouped together inside the LINKS node.

An example of a valid XML is:

```
<LINKS>
        <Events REF="ID7"/>
</LINKS>
```