

Aeronautical Information Exchange Model (AIXM)

AIXM UML to XML Schema Mapping

Copyright: 2007 - EUROCONTROL and Federal Aviation Administration

All rights reserved.

This document and/or its content can be download, printed and copied in whole or in part, provided that the above copyright notice and this condition is retained for each such copy.

For all inquiries, please contact:

Brett BRUNK - brett.brunk@faa.gov

Eduard POROSNICU - eduard.porosnicu@eurocontrol.int

Edition No.	Issue Date	Author	Reason for Change
0.1	2006/06/22	Brett Brunk	First Edition
	2006/08/25	Barb Cordell	Incorporate data modelling section
	2006/09/06	Vamshi Reddy	Incorporate Data type section
	2007/01/30	Barb Cordell	Update for Release Candidate 1
0.2	2007/07/31	Scott Wilson	Update for Release Candidate 2

CONTENTS

1	SCOPE	1
1.1	Introduction	1
1.2	References	1
2	AIXM UML MODELLING CONVENTIONS	2
2.1	Diagram types.....	2
2.2	Stereotypes	2
2.3	Abstract Classes.....	2
2.4	Features.....	2
2.5	Objects.....	3
2.6	Choice	4
2.7	Properties	4
2.7.1	Attributes	4
2.8	DataTypes	5
2.8.1	Relationships	5
2.8.1.1	Relationships to Objects	5
2.8.1.2	Relationships to Features	6
2.8.1.3	Association Classes.....	6
2.8.1.4	Inheritance	6
2.9	Naming	7
3	OTHER ASPECTS OF THE MODEL	8
3.1	The Abstract Model.....	8
3.2	NilReasonType.....	9
4	MAPPING TO THE AIXM XML SCHEMA	10
4.1	AIXM is GML.....	10
4.2	The GML Object-Property Model.....	10
4.3	Mapping Inheritance.....	11
4.4	Mapping Name of Classes.....	11
4.5	Mapping Features.....	11
4.5.1	An Example Mapping	11
4.5.1.1	RunwayPropertyGroup	11
4.5.1.2	RunwayTimeSliceType	13
4.5.1.3	RunwayTimeSlice.....	14
4.5.1.4	RunwayTimeSlicePropertyType.....	15
4.5.1.5	RunwayType.....	15
4.5.1.6	Runway	15

4.5.1.7	RunwayExtension	16
4.6	Mapping Objects	17
4.6.1	An Example Mapping	17
4.6.1.1	AbstractAddressExtension	17
4.6.1.2	AddressPropertyGroup	18
4.6.1.3	AddressType	18
4.6.1.4	Address	19
4.6.1.5	AddressPropertyType	20
4.7	Mapping Choices	20
4.8	Mapping Relationships to Objects	20
4.8.1	Mapping Associations with Association Classes	21
4.9	Mapping Relationships to Features	22
4.10	Mapping Data Types	23
4.10.1	<<datatype>>	23
4.10.2	<<enumeration>>	23
4.10.3	<<codelist>>	25
4.11	Mapping Unit of Measurement	26

1 Scope

1.1 Introduction

With version 5, AIXM moves from Entity-Relationship diagrams to the Unified Modeling Language (UML) diagrams. The AIXM Conceptual Model and data standard are maintained as a UML model. The AIXM exchange model is codified as a series of XML schemas. There is a direct link between the AIXM Conceptual Model and the AIXM XML Schema.

This document describes how the AIXM Conceptual Model is converted into the AIXM XML Schema. The conversion process is illustrated using a series of examples from the AIXM 5 XML schema.

1.2 References

1. Geographic Information – Spatial Schema. ISO 19107. First Edition, 2003-05-01
2. Geography Markup Language (GML). ISO/TC 211/WG 4/PT 19136 OGC GML RWG. Committee Draft. 2004-02-07.
3. UML 2.0 In a Nutshell. Dan Pilone. O'Reilly Media Inc. 2005.
4. AIXM 5 Modelling Conventions.
5. AIXM 5 Proposal
6. AIXM 5 Profile of GML
7. Galdos
8. Rational Edge UML Basics

2 AIXM UML Modelling Conventions

2.1 Diagram types

Two types of diagrams are used in the model:

- Class diagrams – Used to represent the features, properties, relationships and inheritance between features;
- Package diagrams – Used to split the model into modules and identify dependencies among sets of classes.

2.2 Stereotypes

The classes are distinguished by their stereotypes. Stereotypes are used to further define and extend standard UML concepts. The main stereotype are <<feature>>, <<object>>, <<choice>>, <<datatype>>, <<enumeration>> and <<odelist>>.

2.3 Abstract Classes

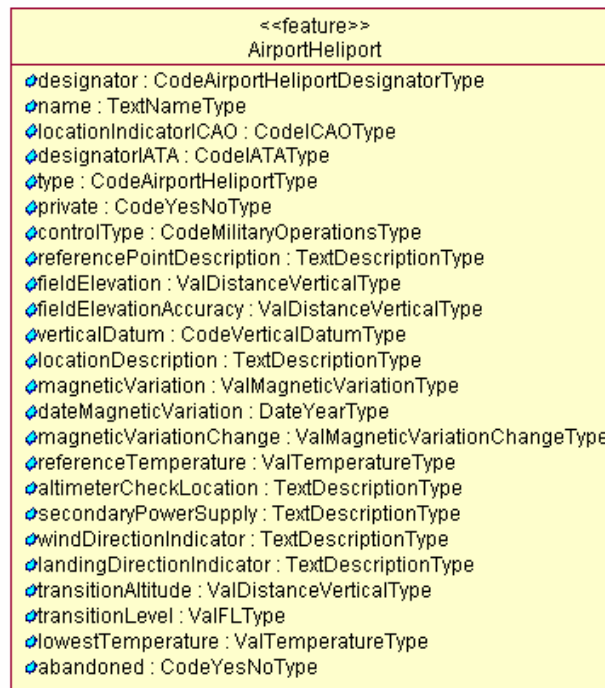
In addition, some classes are abstract. Abstract classes are designated by putting the class name in *italics*. An abstract class cannot be realised in an implementation such as an XML document. Abstract classes are used as base classes in an inheritance hierarchy. For example, the *AIXMFeature* abstract class describes the basic properties of an AIXM Feature. Every specific AIXM Feature, such as Runway, inherits¹ from the abstract *AIXMFeature* class.

2.4 Features

Features describe real world entities and are fundamental in AIXM. AIXM features can be concrete and tangible, or abstract and conceptual and can change in time [7]. Features are represented as classes with a stereotype <<feature>>. Examples include Runway and AirportHeliport.

AIXM features are dynamic features. Timeslice objects are used to describe the changes that affect the AIXM feature over time. Timeslice objects and temporality are discussed extensively in a separate AIXM Temporality document.

¹ Please see section 3.1 The Abstract Model, which explains why this inheritance is not visible in UML

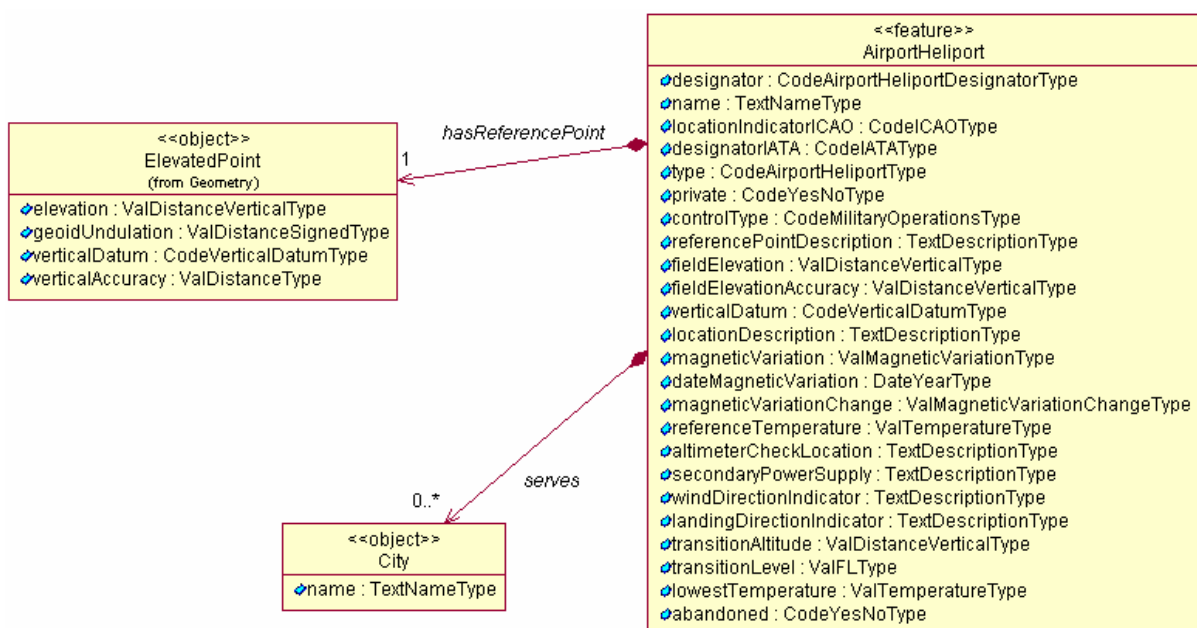


2.5 Objects

Objects are abstractions of real world entities or, more frequently, of properties of these entities, which do not exist outside of a feature. An object is created for two reasons in AIXM:

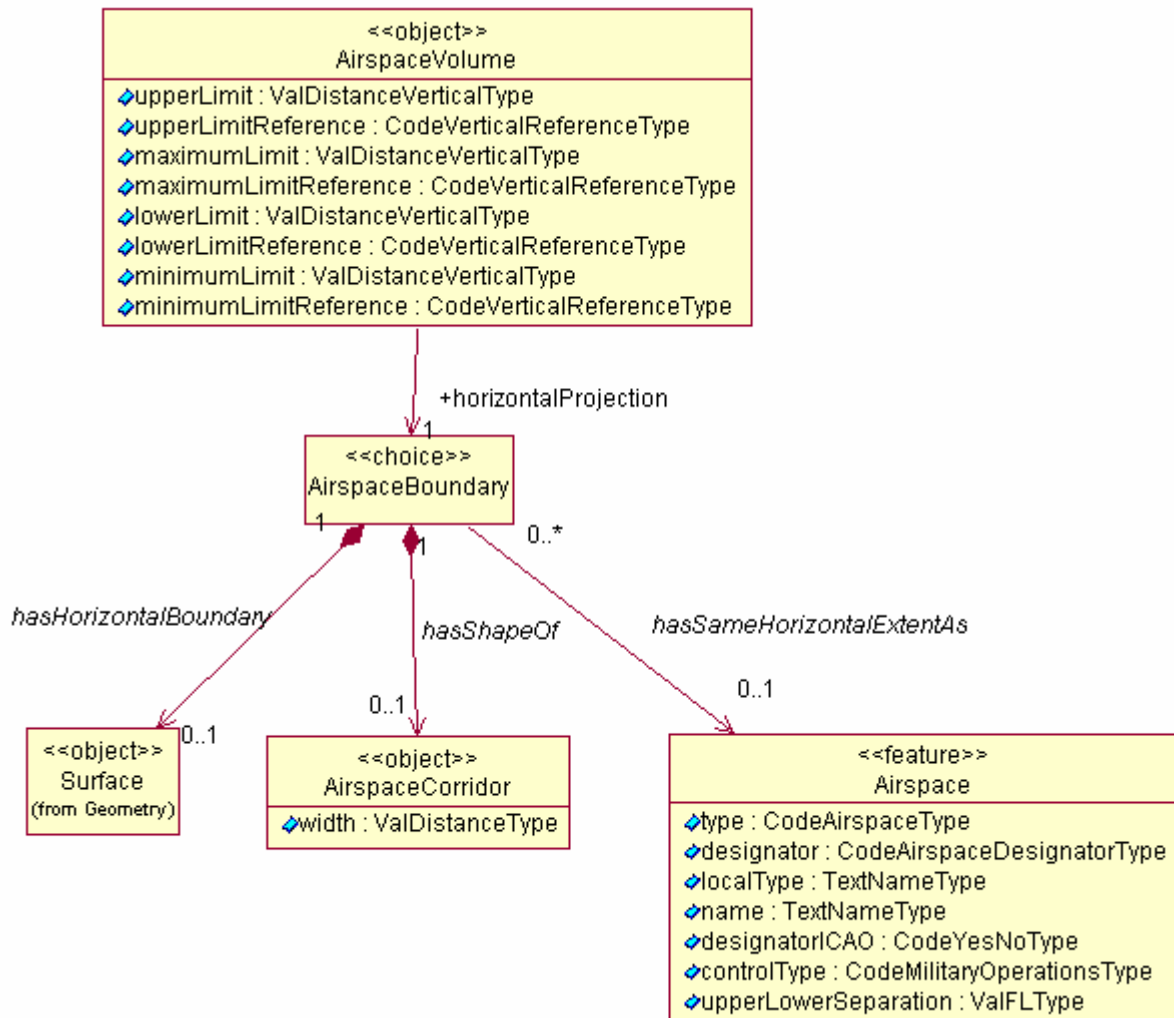
- When a property has a multiplicity greater than one (such as the city served by an AirportHeliport), or
- The object has its own attributes that are reused throughout the model, such as ElevatedPoint.

In both cases, the property is represented as an object with the proper UML composition relationship as shown below.



2.6 Choice

Some classes are marked as `<<choice>>`. These are used to model XOR relationships. For example, an AirspaceVolume's horizontal projection can be a Surface, an AirspaceCorridor or the same shape as for another Airspace.



2.7 Properties

Properties are the attributes and relationships that characterise a feature or object. In the UML:

- Attributes are used to describe simple properties of a feature or object;
- Relationships are used to describe associations to features or objects. Whenever a property has a multiplicity greater than one, it is described using a UML relationship with cardinality.

2.7.1 Attributes

Simple properties of cardinality one are shown as attributes in the UML diagram.

An attribute has the following format:

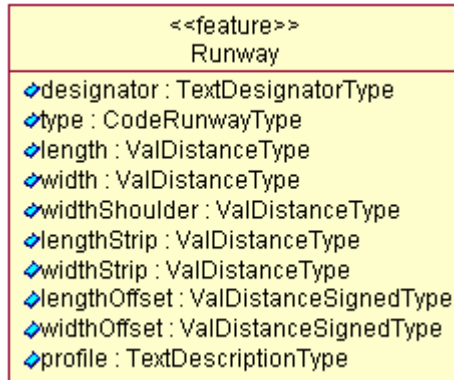
Visibility / stereotype name : type multiplicity

For AIXM 5 the following values are used:

- Visibility – Public
- / - not used

- Stereotype – not used
- Name – name of the property
- Type – property type
- Multiplicity – usually not specified; for reasons related to the AIXM Temporality model, an implementation should assume that all properties are optional, [0..1]

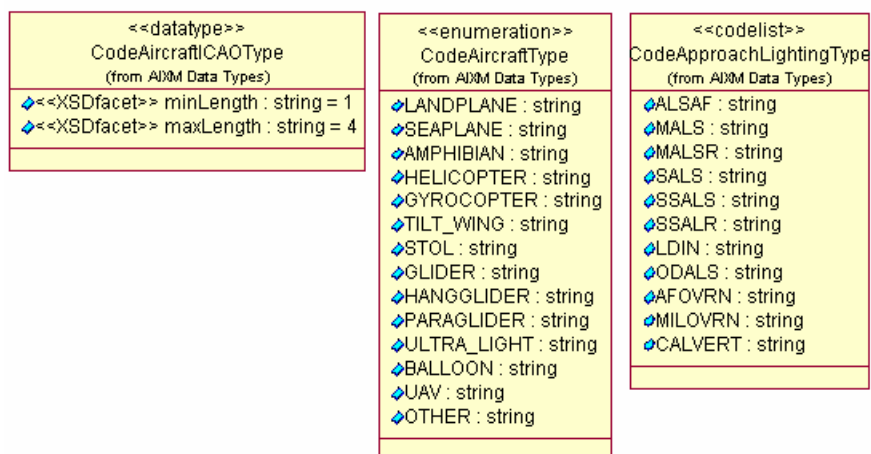
To illustrate, the Runway feature has several simple properties e.g. designator and type. These properties are assigned a datatype; for example, the designator attribute is of type TextDesignatorType.



2.8 DataTypes

The UML model lists the datatypes that are used throughout the model. These are given one of three stereotypes:

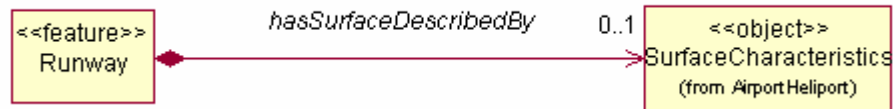
- <<datatype>> – This is basic data type that specifies a pattern to use.
- <<enumeration>> – This codes a fixed list of values. The OTHER value is important for mapping future changes to the list. If a value is added in the future and is not recognised by the current enumeration, it is mapped to the OTHER value.
- <<odelist>> – This is similar to an enumeration in that it is used to indicate a list of possible values. However, the list can be expanded, which means that the codelist is a union between the explicitly enumerated values and free text.



2.8.1 Relationships

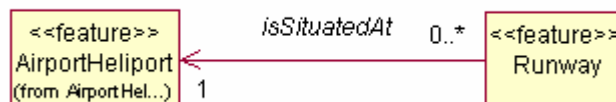
2.8.1.1 Relationships to Objects

Relationships to objects are depicted by the standard UML composition (*aggregation by value*) association. Composition is a form of aggregation with strong ownership and coincident lifetime of the parts by the whole. The part is removed when the whole is removed.



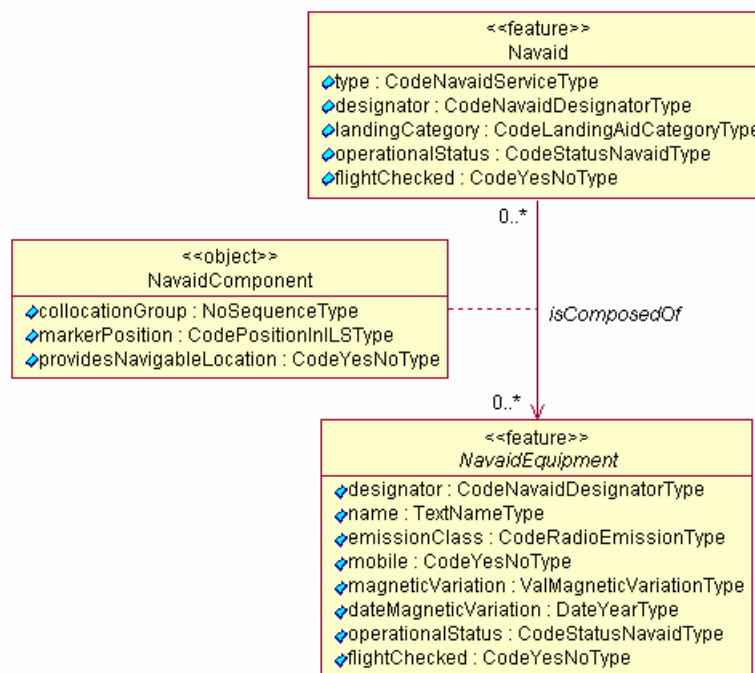
2.8.1.2 Relationships to Features

Relationships to features are described with a standard UML association. All of the associations are navigable in only one direction. This shows that the two classes are related but only one class knows that the relationship exists [8]. In the example below the Runway feature knows about the AirportHeliport but the AirportHeliport does not know about the Runway.



2.8.1.3 Association Classes

When information about a relationship is required, a UML association class is used. The association class is attached to the relationship with a dotted line.

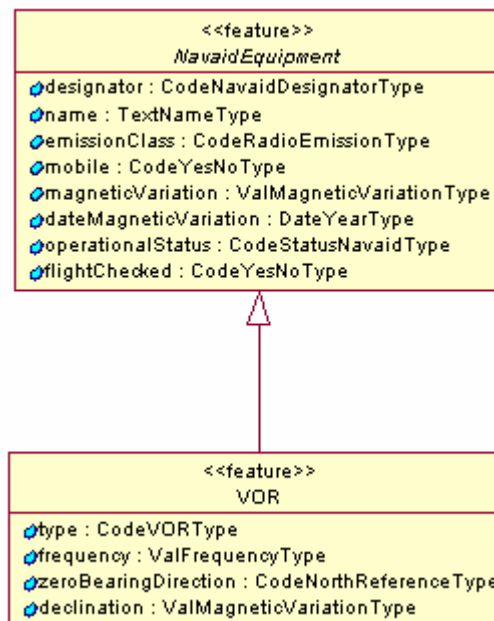


It is normally good practice in UML to make the name of the association class and the association the same. In the example, the NavaidComponent class should be called isComposedOf. However, AIXM 5 does not follow this. This departure was done so that AIXM conforms to the GML object/property model, where association names become properties (lowerCamelCase) and class names become objects (UpperCamelCase). To enforce the UML practice would have meant breaking the object/property model and result in “ugly” XML.

2.8.1.4 Inheritance

Inheritance refers to the ability of one class (the specialized or child class) to inherit the properties of another class (the generalized or parent class), and then add new properties of its own. In AIXM, Features must only inherit from other Features and Objects must only inherit from other Objects. Multiple inheritance is not allowed.

In the example below the VOR is a kind of NavaidEquipment.



2.9 Naming

Feature, Object and Choice names are written in UpperCamelCase e.g. NavaidEquipment.

Simple property names (i.e. attributes) are written in lowerCamelCase e.g. widthShoulder.
 Relationship names are written in lowerCamelCase but as present tense verbs e.g. isSituatingAt.

Datatype names are written in UpperCamelCase and end with 'Type' e.g. CodeAircraftType.

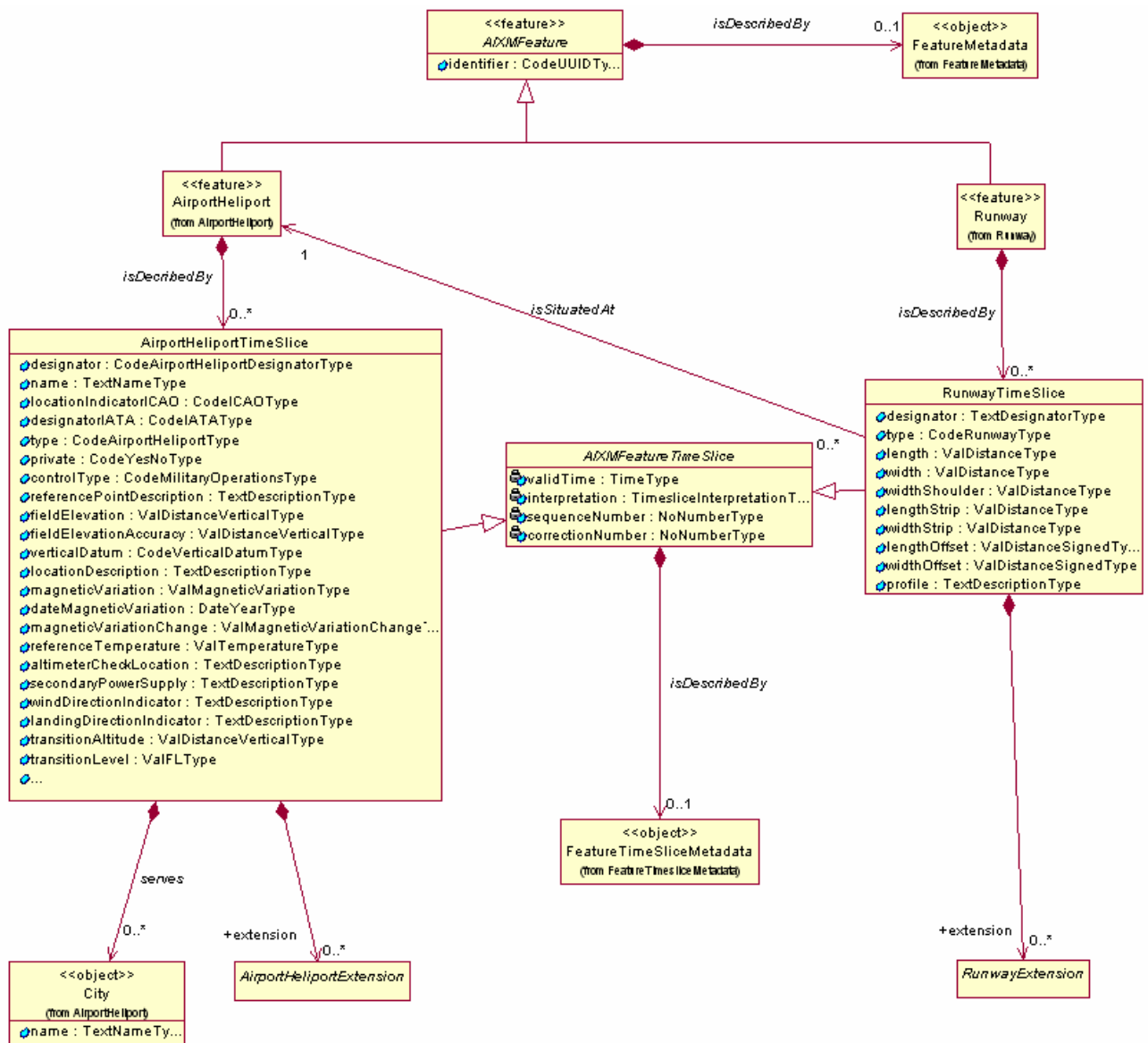
3 Other Aspects of the Model

To simplify the UML model some convenience steps have been taken. Some elements are not shown on every diagram and some relationships are ‘assumed’.

3.1 The Abstract Model

The model should contain a set of abstract AIXM classes that are used as the building blocks for the AIXM XML Schema. However, for simplicity, these relationships are not shown on any diagram and do not really exist in the UML. They are just assumed to exist, when converting from the UML model to the XML Schema of AIXM.

The UML below shows how each and every <<feature>> inherits from the abstract AIXMFeature class. The concrete features are described by TimeSlices which are composed of properties. The TimeSlice inherits from the abstract AIXMFeatureTimeSlice class.



The diagram also shows that each AIXM Feature is described by FeatureMetadata and each TimeSlice is described by FeatureTimeSliceMetadata.

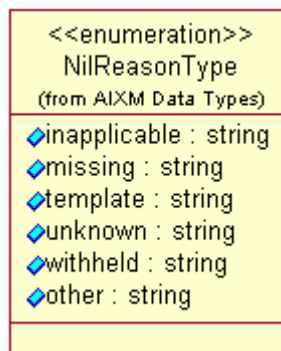
Finally, each TimeSlice may contain an Extension. The Extension mechanism allows each user of AIXM5 to define and use his own specific attributes and classes.

All of these relationships and classes must be mapped in the AIXM XML Schema.

The diagram above is quite complex. If applied to the whole set of AIXM classes, it might undermine the readability of the UML diagrams. Therefore, the Design Team has decided to provide a simplified UML model, without visible inheritance of all features from the abstract AIXMFeature and without visible *SomeFeatureTimeSlice* classes. If considered necessary by the AIXM user community, the final AIXM 5 Release could implement the complete class hierarchy and TimeSlice specialisations in the published UML model.

3.2 NilReasonType

The UML model contains a datatype called NilReasonType. This is not shown on the main diagrams to save clutter. However, the AIXM XML Schema has this attribute on all of the AIXM properties.



4 Mapping to the AIXM XML Schema

4.1 AIXM is GML

As discussed extensively in the AIXM 5 Proposal [6], the AIXM exchange model is an XML exchange standard based on a subset of GML. Essentially:

- AIXM Features are GML features;
- AIXM Objects are GML objects;
- AIXM follows the GML object-property concept.

4.2 The GML Object-Property Model

The GML object-property model explains some of the complexity of the mapping. It means that no GML object may appear as the immediate child of a GML object. Consequently, no element may be both a GML object and a GML property.

The object-property model prohibits the encoding of an object directly inside a feature, e.g.

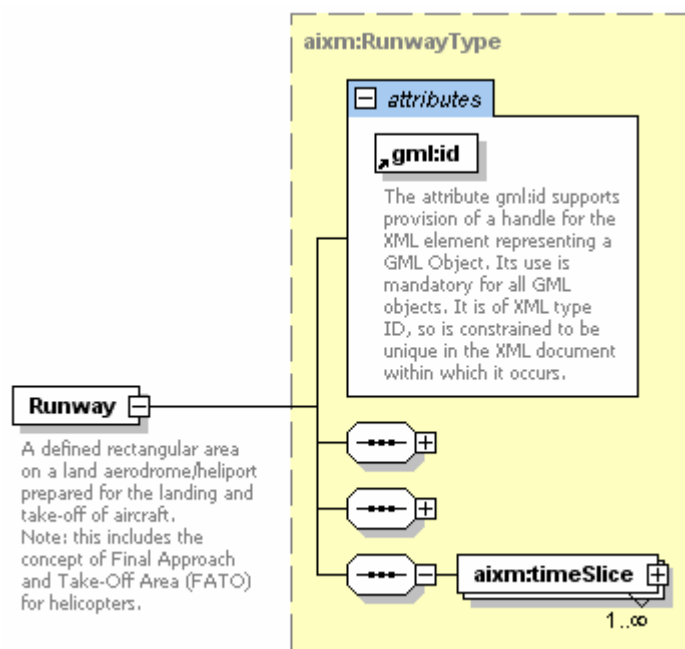
```
<AirportHeliport> <!-- feature -->
  <ElevatedPoint> <!-- object -->
```

Instead, in a compliant GML application schema, an association between two features (or a feature and an object) is implemented over a property of the feature, e.g.

```
<AirportHeliport> <!-- feature -->
  <hasReferencePoint> <!-- property -->
    <ElevatedPoint> <!-- object -->
```

The direction of the association arrow from the UML diagrams (the navigability) dictates which of the two association partners has the property that associates the other.

In the AIXM XML Schema, the object-property model is encoded by declaring a type and then assigning properties (attributes and relationships) to that type. The type defines the object.



4.3 Mapping Inheritance

Within the AIXM XML Schema, inheritance implies two characteristics:

1. Substitutability. The more general feature or object can be substituted by a specialization. In the XML schema this is supported using substitution groups.
2. Property inheritance. The specialized feature inherits all of the properties of the more general feature. In the XML schema including the properties of the general class into the specialized class supports this.

4.4 Mapping Name of Classes

The UML class name is used for the element names in the XML Schema.

4.5 Mapping Features

For each AIXM Feature in the UML, the following XML schema entities are created:

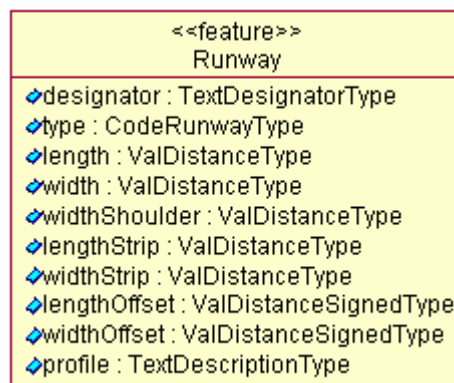
- *FeaturePropertyType*
- *Feature*
- *FeatureType*
- *FeatureTimeSlicePropertyType*
- *FeatureTimeSlice*
- *FeatureTimeSliceType*
- *FeaturePropertyGroup*
- *AbstractFeatureExtension*



The direction in which the different types and elements are used in the schema definition (e.g. Feature uses FeatureType)

4.5.1 An Example Mapping

The Runway feature (shown below) will be used to illustrate the mapping. The example will concentrate on the properties (shown as attributes).



4.5.1.1 RunwayPropertyGroup

An XML Schema (XSD) group is generated for each feature containing all of the properties (attributes and relationships) of the feature.

Below is an example of the RunwayPropertyGroup in graphic form and as an extract from the XSD. It shows clearly how the attributes are mapped from the UML to the XSD and how the relationship 'isSituatingAt_AirportHeliport' is created. It also illustrates the presence of the nilReason attribute.



```

<group name="RunwayPropertyGroup">
  <sequence>
    <element name="designator" nillable="true" minOccurs="0">
      <complexType>
        <simpleContent>
          <extension base="aixm:TextDesignatorType">
            <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    <element name="type" nillable="true" minOccurs="0">
      <complexType>
        <simpleContent>
          <extension base="aixm:CodeRunwayType">
            <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    <element name="length" nillable="true" minOccurs="0">
      <complexType>
        <simpleContent>
          <extension base="aixm:ValDistanceType">
            <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    ...
    <element name="hasSurfaceDescribedBy"
      type="aixm:SurfaceCharacteristicsPropertyType" minOccurs="0"/>
  </sequence>

```

```

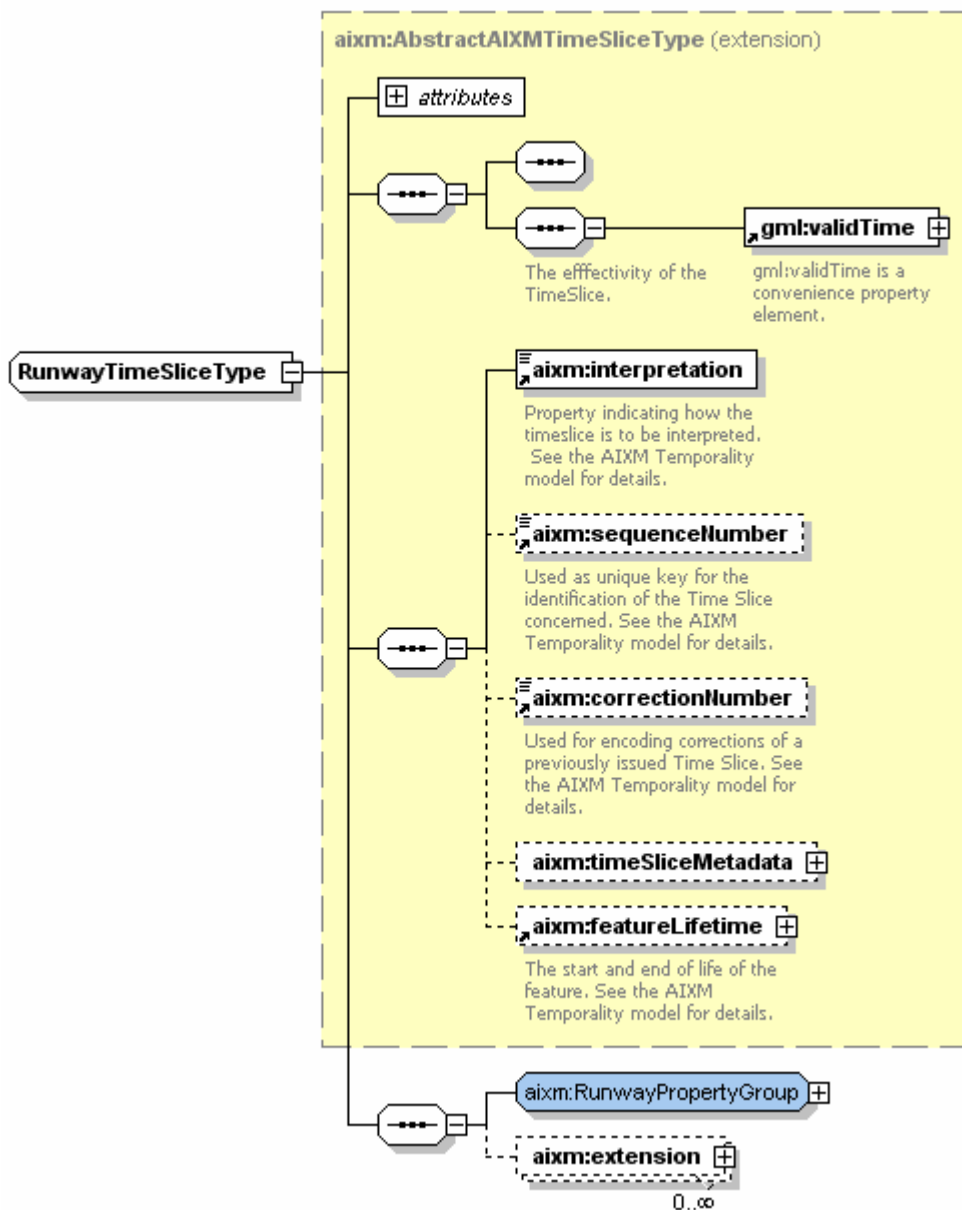
    <element name="isSituatingAt_AirportHeliport"
type="aixm:AirportHeliportPropertyType" minOccurs="0"/>
    ...
  </sequence>
</group>

```

4.5.1.2 RunwayTimeSliceType

The properties of a feature or the target of any feature relationship can change within the lifetime of the feature. This temporality can be expressed in GML by using dynamic features and feature collections. The TimeSlice property of a dynamic feature contains one or more Feature TimeSlices that capture the evolution of the feature over time. A gml:TimeSlice object contains the dynamic properties of the feature.

For each feature a TimeSlice property is created that contains an array of feature TimeSlice objects. This example shows the RunwayTimeSliceType encapsulating all of the Runway properties (RunwayPropertyGroup created above) that change over time.



```

<complexType name="RunwayTimeSliceType">
  <complexContent>
    <extension base="aixm:AbstractAIXMTimeSliceType">
      <sequence>
        <group ref="aixm:RunwayPropertyGroup" />

```

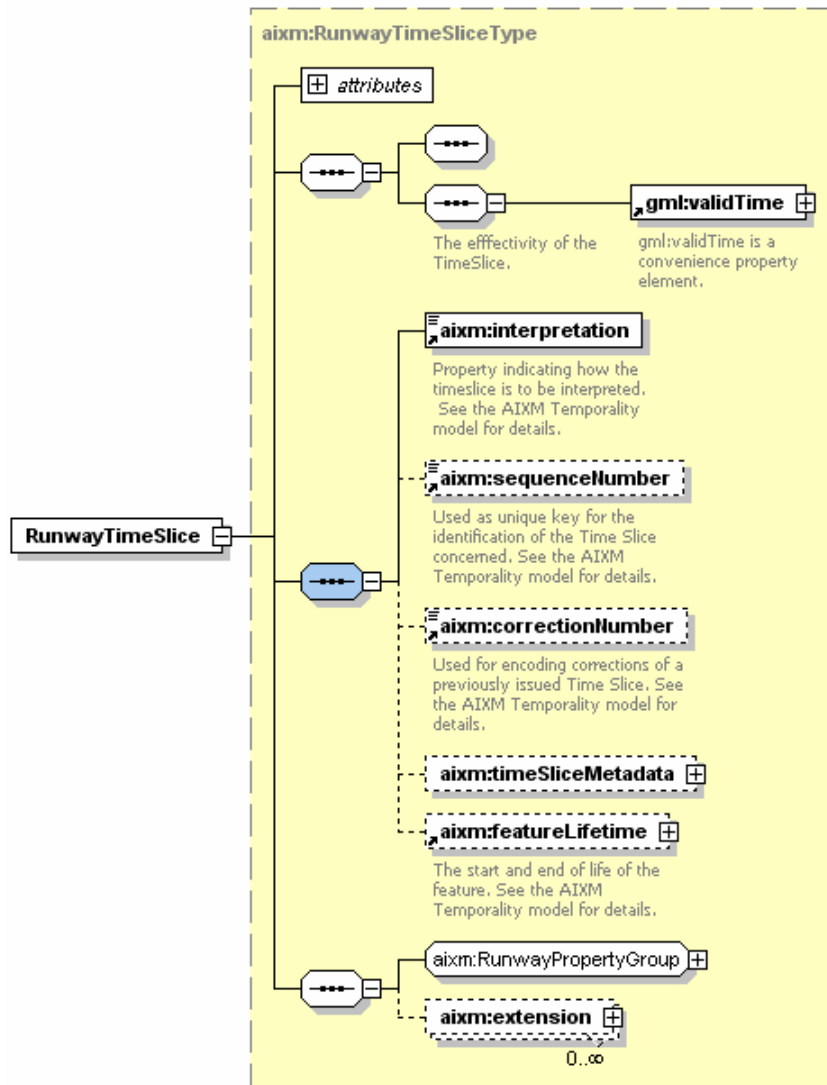
```

<element name="extension" minOccurs="0" maxOccurs="unbounded">
  <complexType>
    <sequence>
      <element ref="aixm:AbstractRunwayExtension"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
</element>
</sequence>
</extension>
</complexContent>
</complexType>

```

4.5.1.3 RunwayTimeSlice

The *FeatureTimeSlice* object is of type *FeatureTimeSliceType*. Continuing the example, the *RunwayTimeSlice* element is of type *RunwayTimeSliceType*.



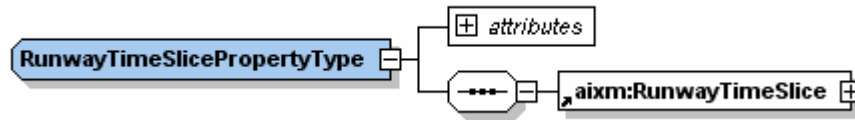
```

<element name="RunwayTimeSlice" type="aixm:RunwayTimeSliceType"
substitutionGroup="gml:AbstractTimeSlice"/>

```

4.5.1.4 RunwayTimeSlicePropertyType

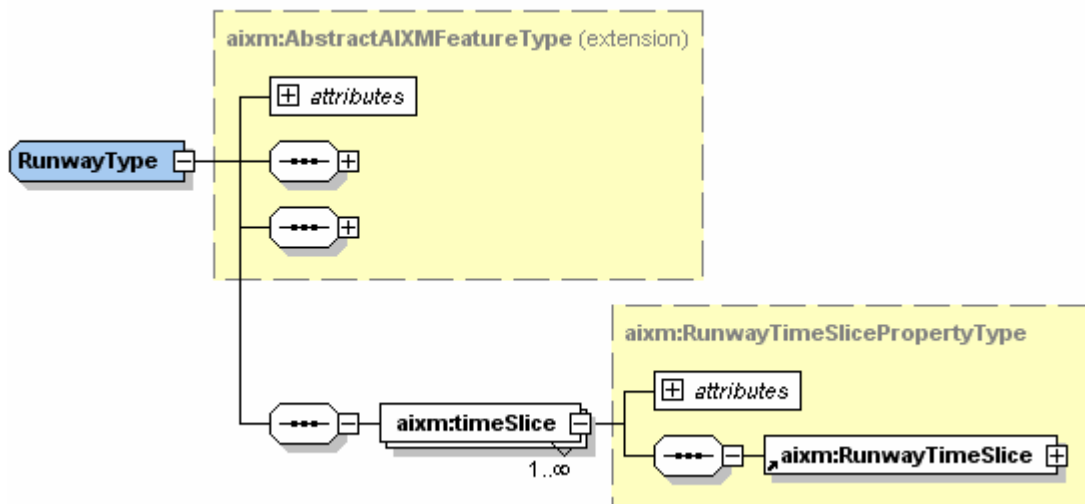
A GML property type containing a *FeatureTimeSlice* objects is created.



```
<complexType name="RunwayTimeSlicePropertyType">
  <sequence>
    <element ref="airm:RunwayTimeSlice"/>
  </sequence>
  <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

4.5.1.5 RunwayType

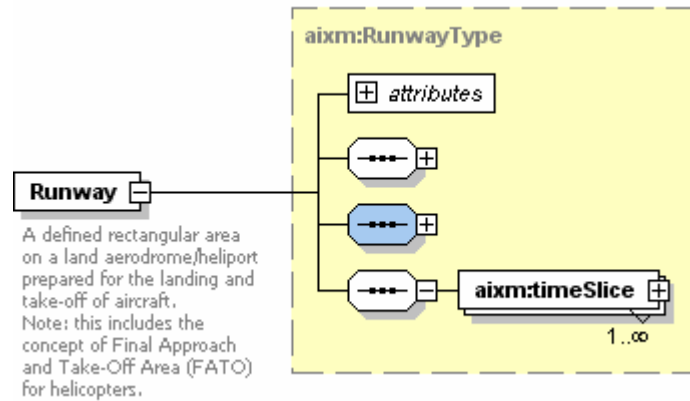
Continuing with the object-property model, the Runway feature type is created extending the `AbstractAIXMFeatureType` with the `RunwayTimeSlice` object created above.



```
<complexType name="RunwayType">
  <complexContent>
    <extension base="airm:AbstractAIXMFeatureType">
      <sequence>
        <element name="timeSlice"
          type="airm:RunwayTimeSlicePropertyType" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

4.5.1.6 Runway

The Runway feature is then defined by the `RunwayType`.

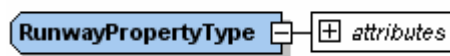


```
<element name="Runway" type="aixm:RunwayType"
substitutionGroup="aixm:AbstractAIXMFeature">
  <annotation>
    <appinfo>RWY</appinfo>
    <documentation>A defined rectangular area on a land
aerodrome/heliport prepared for the landing and take-off of
aircraft. Note: this includes the concept of Final Approach and
Take-Off Area (FATO) for helicopters.</documentation>
  </annotation>
</element>
```

4.5.1.6.1 RunwayPropertyType

When a property of a feature is a relationship, the relationship must be associated to another feature or object. This is done through the creation of the *FeaturePropertyType*, in this case, the *RunwayPropertyType*.

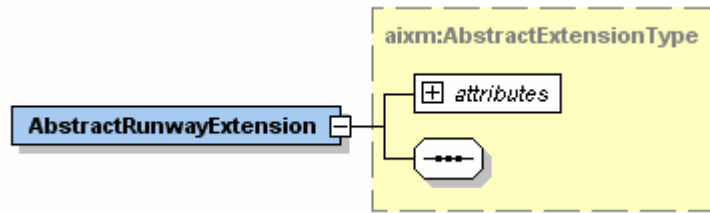
In AIXM, when the relationship or association points to another feature, the feature is referenced using the `xlink:href` attribute (it's always a "remote" encoding). When the relationship points to an object, the object is included inside the parent. (Objects cannot exist without the parent.) Since a Runway is a feature the *RunwayPropertyType* is created with the attribute `xlink:href`.



```
<complexType name="RunwayPropertyType">
  <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
</complexType>
```

4.5.1.7 RunwayExtension

All Features and Objects can be extended. A relationship is created with an abstract XML element that acts as the root for all extensions. Below is an example of the extension for the Runway feature. The *AbstractRunwayExtension* element uses the *AbstractExtensionType* as shown below.



```
<element name="AbstractRunwayExtension"
type="aixm:AbstractExtensionType" abstract="true"
substitutionGroup="aixm:AbstractExtension" />
```

4.6 Mapping Objects

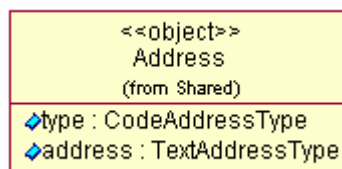
AIXM objects are encoded as GML objects. For the most part, the XML schema entities are created in the same way as for Features, following the object-property model. However it is important to remember that AIXM objects do not exist outside of a feature and are therefore part of the feature timeslice. TimeSlice types and elements are not created for objects.

For each AIXM Object the following XML schema entities are created:

- *ObjectPropertyGroup*
- *Object*
- *ObjectType*
- *ObjectPropertyType*
- *AbstractObjectExtension*

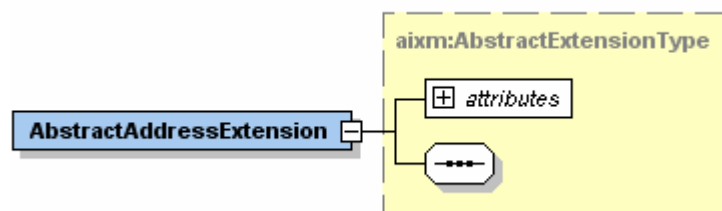
4.6.1 An Example Mapping

The example will use the Address object illustrated below. This is used in many places throughout the model; for example, an AirportHelicopter isContactedAt an Address.



4.6.1.1 AbstractAddressExtension

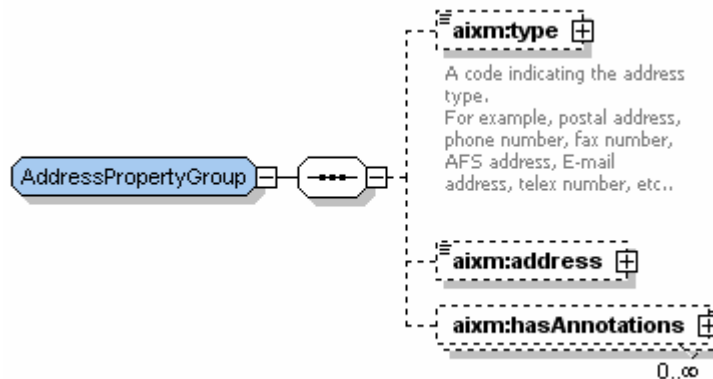
An abstract XML element acts as the root for all extension to theAddress object. Object extensions are defined in the same way as Feature extensions.



```
<element name="AbstractAddressExtension"
type="aixm:AbstractExtensionType" abstract="true"
substitutionGroup="aixm:AbstractExtension" />
```

4.6.1.2 AddressPropertyGroup

An XSD group containing the properties of the Address object is created, again similar to Features.



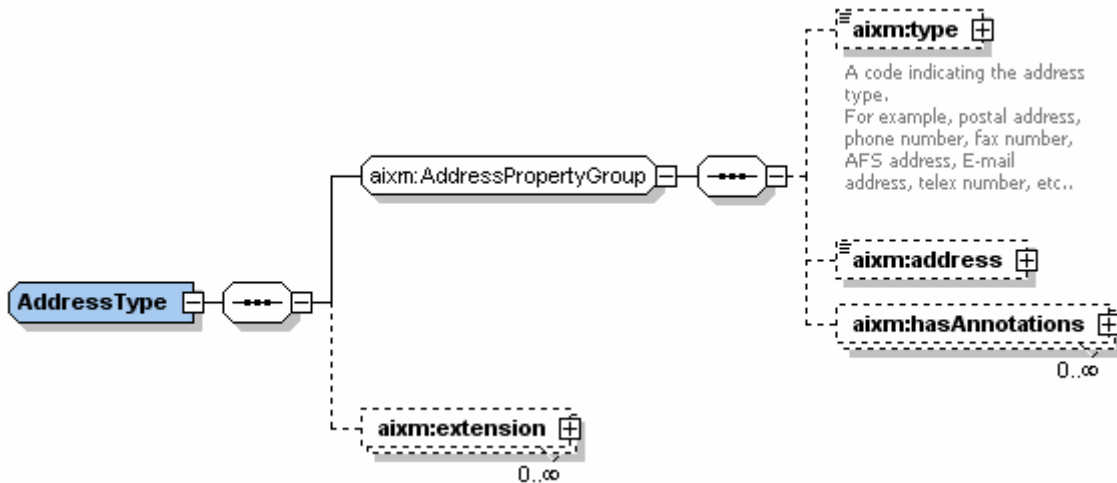
```

<group name="AddressPropertyGroup">
  <sequence>
    <element name="type" nillable="true" minOccurs="0">
      <annotation>
        <appinfo>CODE_TYPE</appinfo>
        <documentation>A code indicating the address type.
        For example, postal address, phone number, fax number, AFS address,
        E-mail address, telex number, etc..
        </documentation>
      </annotation>
      <complexType>
        <simpleContent>
          <extension base="aixm:CodeAddressType">
            <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    <element name="address" nillable="true" minOccurs="0">
      <complexType>
        <simpleContent>
          <extension base="aixm:TextAddressType">
            <attribute name="nilReason" type="gml:NilReasonEnumeration"/>
          </extension>
        </simpleContent>
      </complexType>
    </element>
    <element name="hasAnnotations" type="aixm:NotePropertyType"
    minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</group>

```

4.6.1.3 AddressType

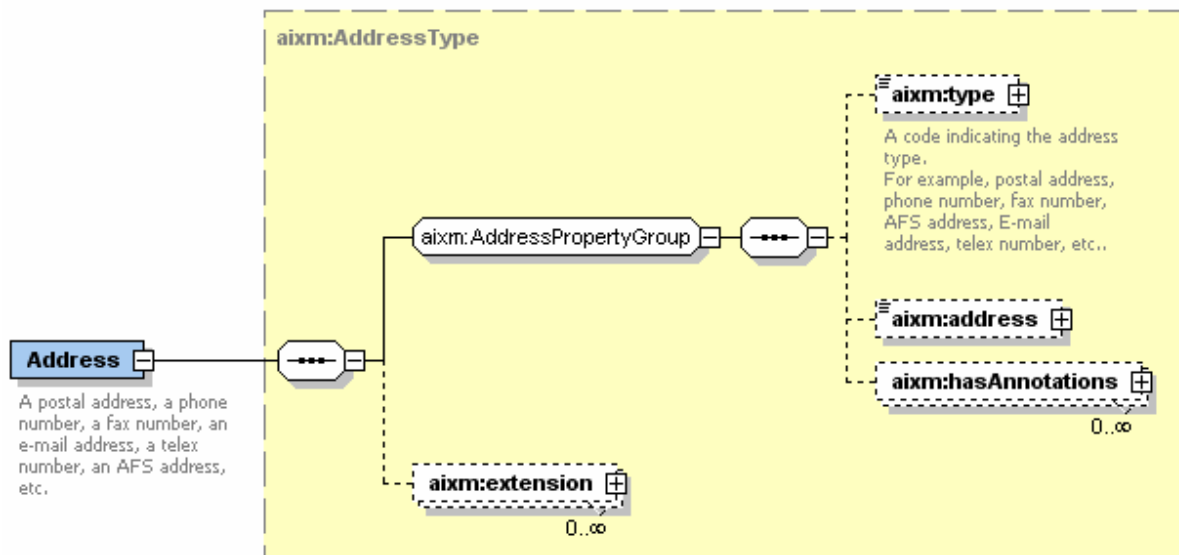
The AddressType definition uses the AddressPropertyGroup and the extension.



```
<complexType name="AddressType">
  <sequence>
    <group ref="aixm:AddressPropertyGroup"/>
    <element name="extension" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <sequence>
          <element ref="aixm:AbstractAddressExtension"/>
        </sequence>
      </complexType>
    </element>
  </sequence>
</complexType>
```

4.6.1.4 Address

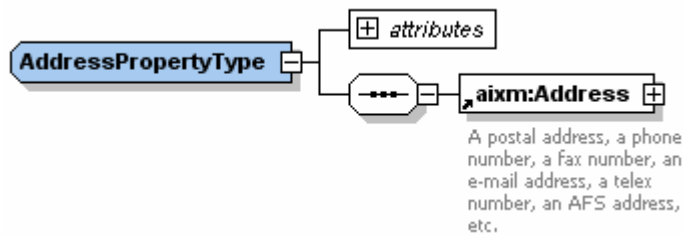
The Address object is then defined as an XSD element of type AddressType.



```
<element name="Address" type="aixm:AddressType">
  <annotation>
    <appinfo>ADDRESS</appinfo>
    <documentation>A postal address, a phone number, a fax number, an
    e-mail address, a telex number, an AFS address, etc.
    </documentation>
  </annotation>
</element>
```

4.6.1.5 AddressPropertyType

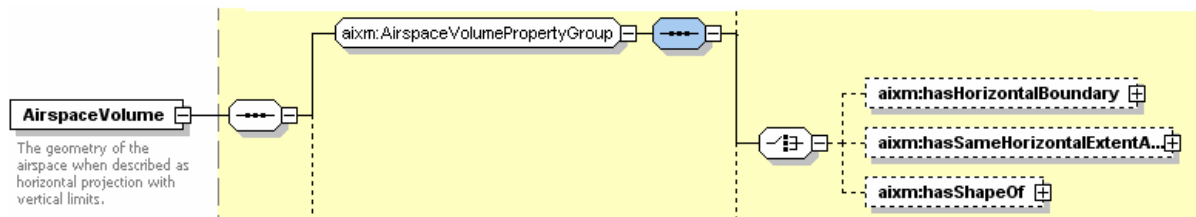
An XSD complex type representing a GML property type is created. A Feature uses this element to include the Address object rather than reference it (using `xlink:href`) because object does not exist without the parent.



```
<complexType name="AddressPropertyType">
  <sequence>
    <element ref="aixm:Address"/>
  </sequence>
  <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

4.7 Mapping Choices

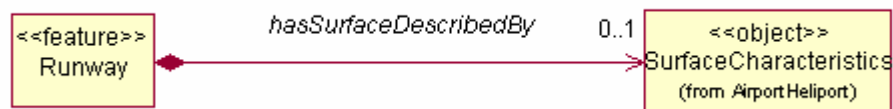
Classes marked with the stereotype `<<choice>>` do not appear in the XML Schema. Instead, the choice of elements is created.

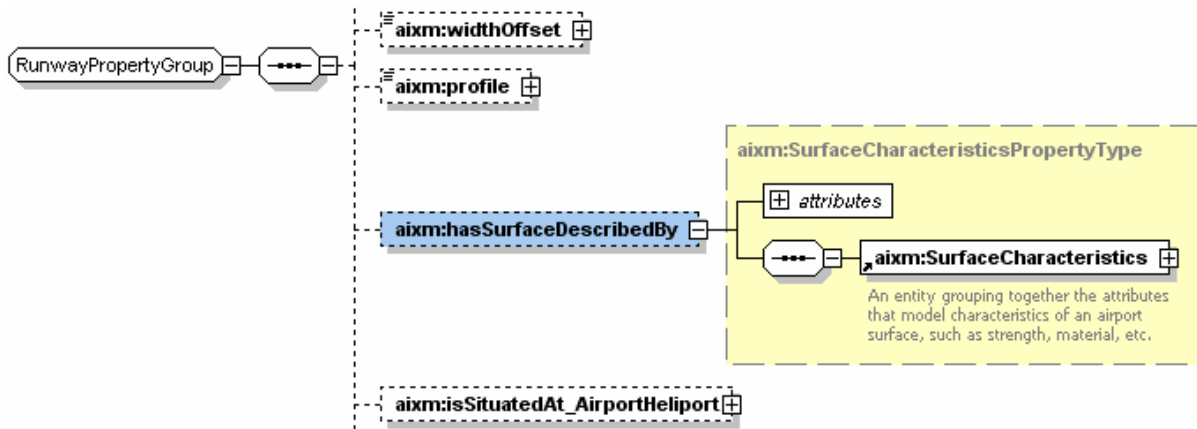


4.8 Mapping Relationships to Objects

Relationships are encoded by creating an XML element with the same name as the rolename on the UML model. It is of type *ObjectPropertyType*.

In this example, the *SurfaceCharacteristics* object is a property of the *Runway*. The **hasSurfaceDescribedBy** property of the *Runway* is defined as being of type *SurfaceCharacteristicsPropertyType*.

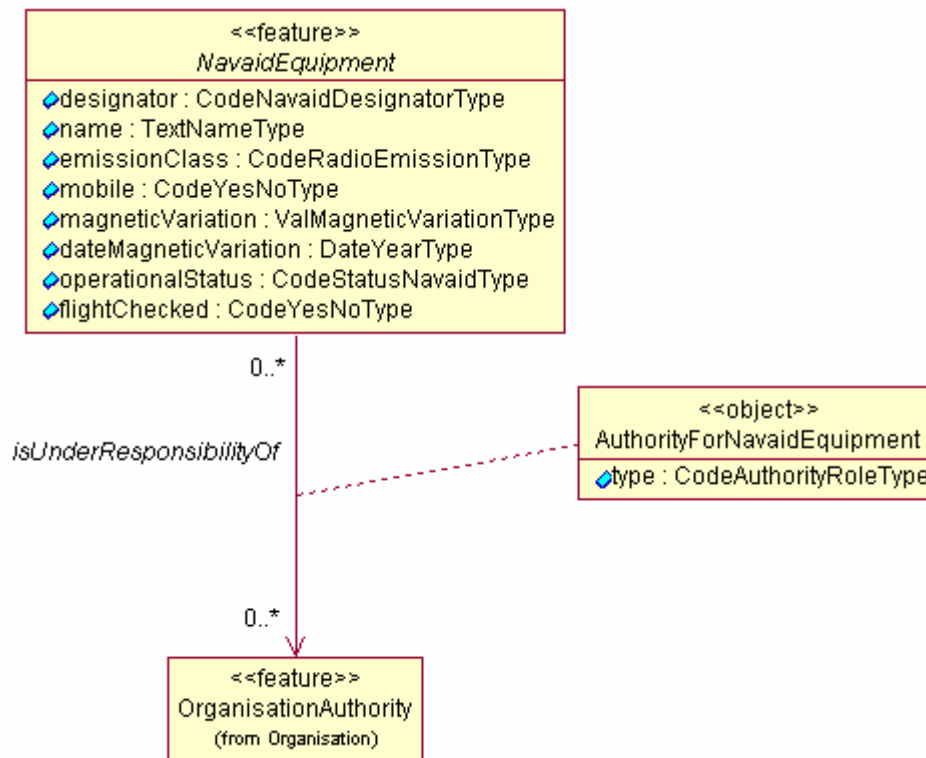




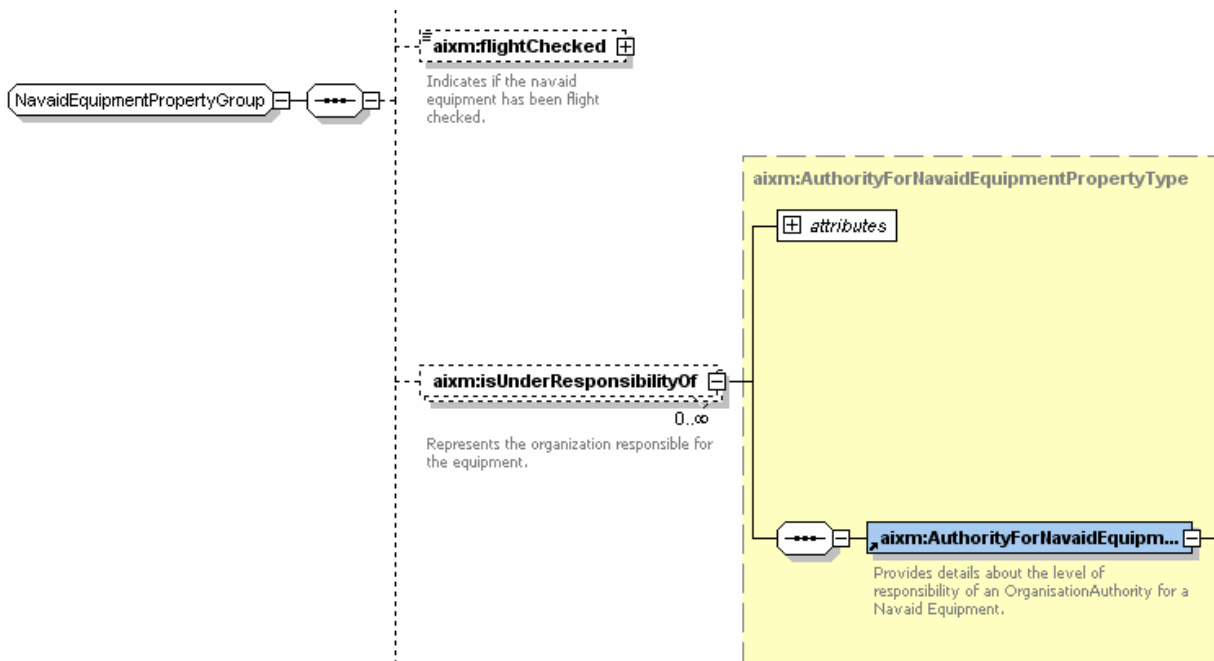
```
<element name="hasSurfaceDescribedBy"
type="aixm:SurfaceCharacteristicsPropertyType" minOccurs="0"/>
```

4.8.1 Mapping Associations with Association Classes

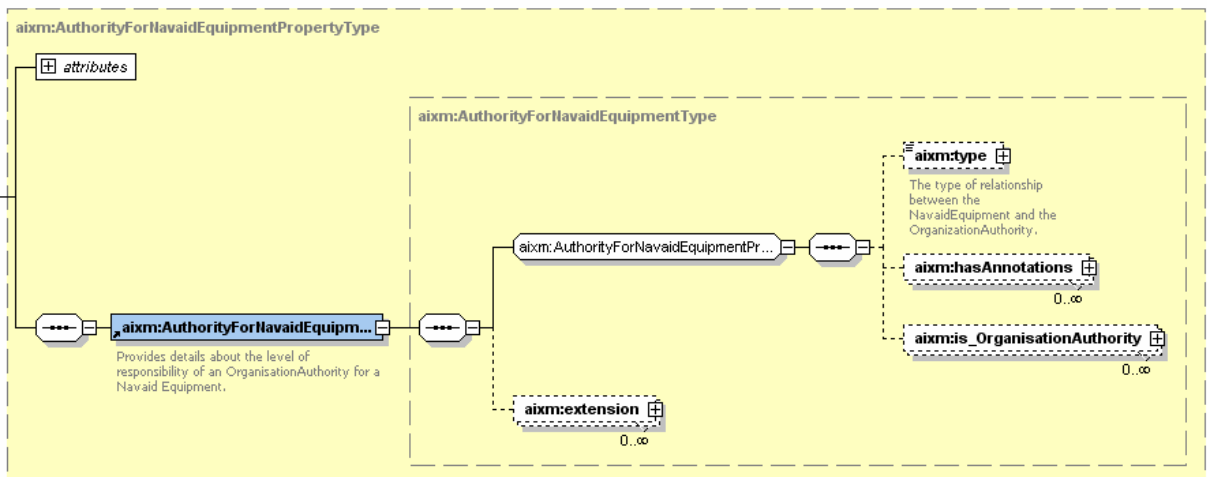
In the UML below, the NavaidEquipment feature has a relationship to the OrganisationAuthority feature. This relationship contains properties defined in the AuthorityForNavaidEquipment object.



Mapping this in XSD an 'isUnderResponsibilityOf' property is created in the NavaidEquipment feature as shown below. (The direction of the arrow is important. If the direction would have been to the NavaidEquipment, the property would have been created in the OrganisationAuthority feature.)

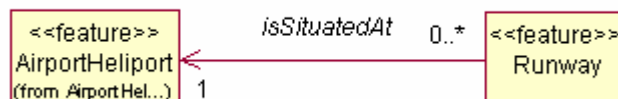


A second step is then required to complete the XSD. In this case an 'is_OrganisationAuthority' element is added.



4.9 Mapping Relationships to Features

In AIXM, Relationships to features are described by reference using `xlink:href`. The UML rolename is used for the XML element name and the XML element is of type `FeaturePropertyType`.



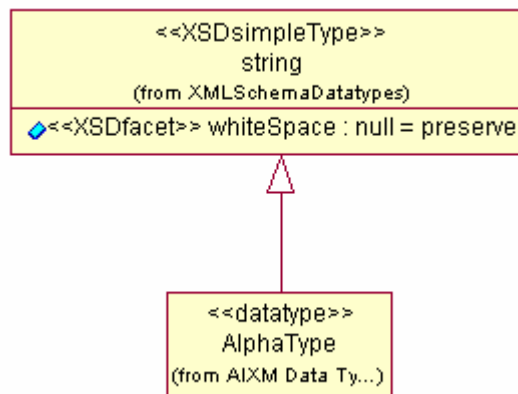


4.10 Mapping Data Types

4.10.1 <<datatype>>

The datatypes map directly to the built-in datatypes defined by the XML schema specification. The default datatypes are `string`, `float`, `double`, etc, which are considered `simpleTypes`.

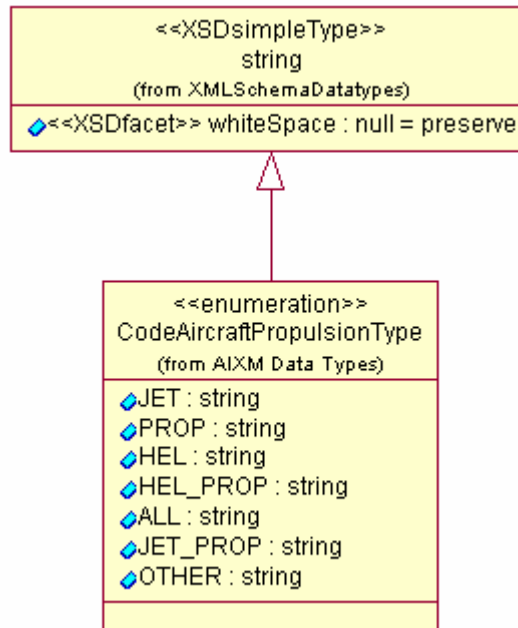
The `AlphaType` acts as a convenient example.



```
<simpleType name="AlphaType">
  <restriction base="xsd:string">
    <pattern value="[A-Z]*"/>
  </restriction>
</simpleType>
```

4.10.2 <<enumeration>>

Enumerations are given the `<<enumeration>>` stereotype e.g. `CodeAircraftPropulsionType`.

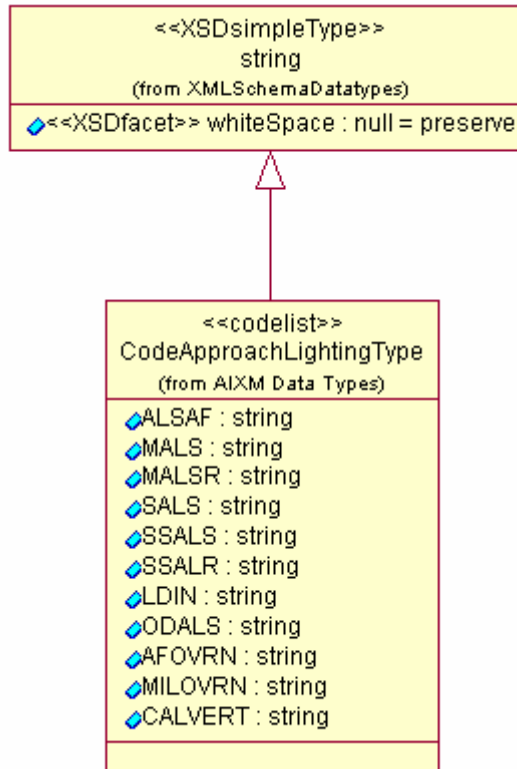


```

<simpleType name="CodeAircraftPropulsionType">
  <annotation>
    <documentation/>
  </annotation>
  <restriction base="xsd:string">
    <enumeration value="JET">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="PROP">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="HEL">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="HEL_PROP">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="ALL">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    ...
  </restriction>
</simpleType>
  
```

4.10.3 <<codelist>>

Codelists are given the stereotype <<codelist>>. An example:



```

<simpleType name="CodeApproachLightingType">
  <union memberTypes="aixm:CodeApproachLightingType_base
xsd:string"/>
</simpleType>
  
```

This is treated as a `simpleType` by combining two or more member classes using `xsd:union`. The `xsd:string` as a member specifies that a value can be one from the pre-specified list or a custom value.

```

<simpleType name="CodeApproachLightingType_base">
  <annotation>
    <documentation/>
  </annotation>
  <restriction base="xsd:string">
    <enumeration value="ALSAF">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="MALS">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="MALSR">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="SALS">
      <annotation>
  
```

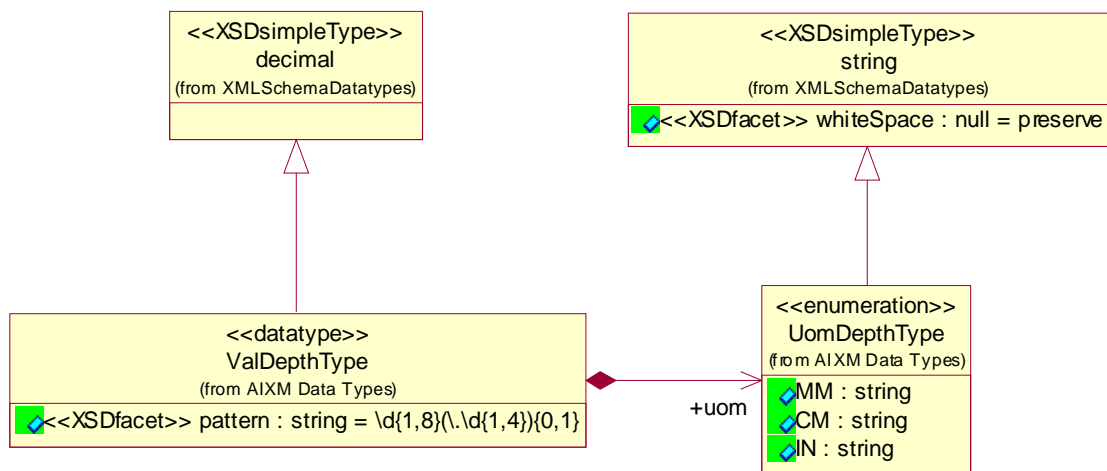
```

    <documentation/>
  </annotation>
</enumeration>
<enumeration value="SSALS">
  <annotation>
    <documentation/>
  </annotation>
</enumeration>
<enumeration value="SSALR">
  <annotation>
    <documentation/>
  </annotation>
</enumeration>
...
</restriction>
</simpleType>

```

4.11 Mapping Unit of Measurement

A Unit of measurement (UOM) exist for many data types that take numerical values. This has been illustrated as a containment relationship drawn from the value class to the UOM class.



In XSD, a simple type is created for the UOM class, as for any other <<enumeration>>.

```

<simpleType name="UomDepthType">
  <annotation>
    <documentation/>
  </annotation>
  <restriction base="xsd:string">
    <enumeration value="MM">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="CM">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
    <enumeration value="IN">
      <annotation>
        <documentation/>
      </annotation>
    </enumeration>
  </restriction>

```

```

    </enumeration>
  </restriction>
</simpleType>

```

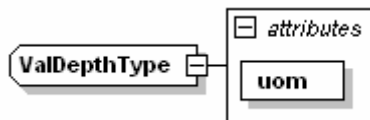
In a second step, the class having an associated UOM is generated first as a “base” simple type.

```

<simpleType name="ValDepthTypeBase">
  <annotation>
    <documentation/>
  </annotation>
  <restriction base="xsd:decimal">
    <pattern value="\d{1,8}(\.\d{1,4}){0,1}" />
  </restriction>
</simpleType>

```

Then, a complex type is created, implementing the UOM list of values as an attribute:



```

<complexType name="ValDepthType">
  <simpleContent>
    <annotation>
      <documentation/>
    </annotation>
    <extension base="aixm:ValDepthTypeBase">
      <attribute name="uom" type="aixm:UomDepthType" use="required" />
    </extension>
  </simpleContent>
</complexType>

```