

AIXM

AIXM Application Schema Generation

Aeronautical Information Exchange Model (AIXM)

Copyright: 2010 - 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	2007/12/20	Barb Cordell / Paul Heffley	First Edition
0.2	2008/01/08	Barb Cordell / Paul Heffley	Updated version
1.0	2008/03/10	Eddy Porosnicu	First public version Editorial modifications
1.1	2010/02/04	Hubert Lepori	Updated version - AIXM 5.1

CONTENTS

1	SCOPE	1
1.1	Introduction	1
1.2	Background	1
1.3	Objective.....	3
1.4	References	3
2	EXTENDING AIXM FEATURES / OBJECTS.....	4
2.1	UML Package for Extensions	4
2.1.1	Package Structure.....	4
2.1.2	Package Specifications and Namespaces	4
2.2	UML Extension Package.....	5
2.2.1	Overview	5
2.2.2	XML Schema Generation.....	6
2.2.2.1	Imported and Included Schemas.....	7
2.2.2.2	Executing the Script.....	8
2.2.2.3	XML Schema Output.....	9
2.3	Data Type Extension Package	11
2.3.1	Overview	11
2.3.2	XML Schema Generation.....	13
2.3.2.1	Imported and Included Schemas.....	13
2.3.2.2	Executing the Script.....	14
2.4	UML Message Package	15
2.4.1	Overview	15
2.4.2	XML Schema Generation.....	16
2.4.2.1	Imported and Included Schemas.....	16
2.4.2.2	Executing the Script.....	16
2.4.2.3	XML Schema Output.....	16

1 Scope

1.1 Introduction

The purpose of this document is to describe how the AIXM UML model can be extended to support the needs of a particular Community of Interest (COI):

- ✓ Define messages that are necessary and eventually restrict the content of these messages to a sub-set of the AIXM features;
- ✓ Extend existing AIXM features with new attributes or associations or define new features, which are only relevant for that community.

The UML to XML Schema conversion for extensions is illustrated using a series of examples from the AIXM 5 Application Schema extensions.

1.2 Background

The AIXM conceptual model and data standard are maintained as a UML model. AIXM was developed to be extendable allowing greater flexibility for international use. Each feature and codelist may be extended to meet individual needs of the AIXM Community of Interest (COI).

If you are not familiar with the AIXM UML to AIXM XSD mapping document please review before reading this document. A good understanding of the mapping document will help to understand AIXM extensions.

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.

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.

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.

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

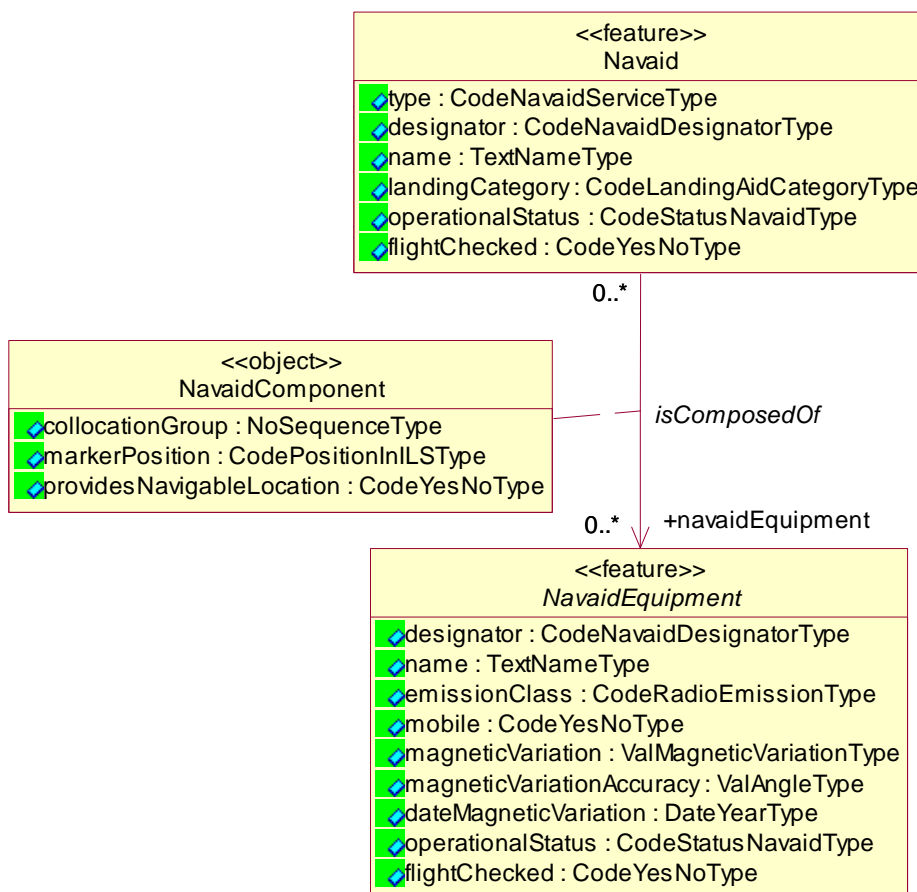
- Attributes are used to describe simple properties of a feature or object;
- Relationships are used to describe associations to features or objects. Whenever a property has a multiplicity greater than one, it is described using a UML relationship with cardinality.

- 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
- 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

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

- <<datatype>> - This is basic data type that specifies a pattern to use.
- <<odelist>> - This is a data type which codes a predefined list of values. The <<odelist>> includes the value OTHER which can be expanded with some free text in uppercase (“OTHER:MY_VALUE”).to support un-supported values.

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



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.

Important Note: Inheritance is supported only from “Abstract” classes. The UML to XSD scripts do not support the inheritance from non-Abstract classes. The “extension”

mechanism, explained in this document, gives the possibility to extend an existing AIXM class with new properties, with the advantage that the extended class remains valid against the core AIXM schema.

1.3 Objective

The core AIXM model provides the definition of standardised aeronautical information features. In order to use AIXM for a specific application, a Community of Interest (COI) will have to agree upon how instances of AIXM features are to be exchanged and communicated in the community. This can be accomplished by either using a pre-defined Web Service (such as WFS), which enables the direct provision of individual AIXM features or collections of AIXM features or by defining custom AIXM messages with specific properties and encompassing a selected list of AIXM features.

In the definition of the AIXM Application Schema, the COI might also want to extend the core AIXM with additional properties and features. Some principles that regulate such extensions include:

- An extension of an existing AIXM feature should remain valid against the definition of the core AIXM XSD element with the same name (for that purpose, the AbstractSomeFeatureExtension element is provided in the core AIXM XSD). A consequence is that it is not possible to extend <<datatype>> classes. Only <<codelist>> may be extended.
- An additional feature and objects shall follow the core AIXM modelling conventions (stereotypes, naming, data types, etc.)

Important Note: It is under the responsibility of the COI to ensure that the extensions do not duplicate features and properties that already exist in the core model. When such extensions are defined, the COI might want to share it with the global AIXM community. For this purpose, the application schema can be made available through www.aixm.aero.

1.4 References

1. Geographic Information – Spatial Schema. ISO 19107. First Edition, 2003-05-01
2. ISO 19136:2007 - Geographic information -- Geography Markup Language (GML)
3. UML 2.0 In a Nutshell. Dan Pilone. O'Reilly Media Inc. 2005.
4. AIXM UML to XML Schema Mapping, www.aixm.aero (see Downloads)
5. AIXM Temporality Model, www.aixm.aero (see Downloads)

2 Extending AIXM Features / Objects

2.1 UML Package for Extensions

To extend AIXM, a new package must be created under the AIXM Application Schemas package. This package will contain all the information you need for your extension.

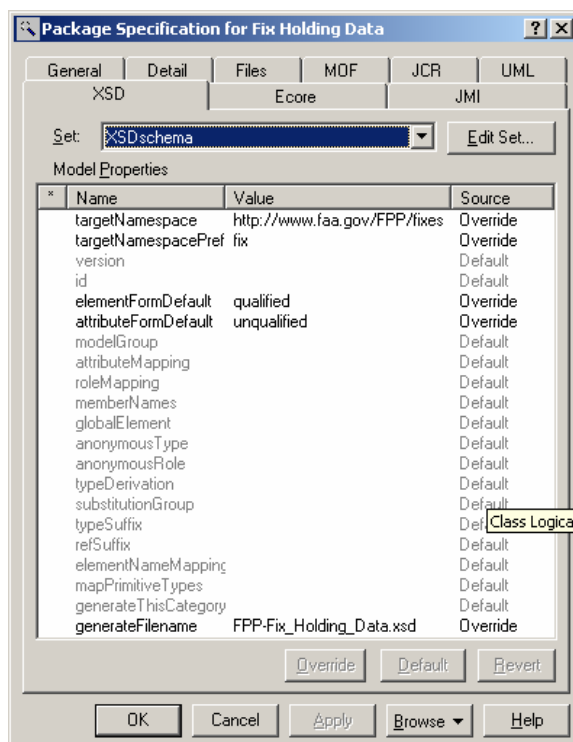
2.1.1 Package Structure

Different types of sub-packages are used to control the generation of appropriate XML schemas (XSD). The Extension sub-package contains the extensions to AIXM Core features and objects. If the extension requires new data types, then a second sub-package, the Extension Data Types, is created containing any new data types and codelists needed. The final sub-packages that are needed are the message packages. Multiple packages may be required based on the number of different message schemas needed. Most Application Schema Packages will have at least one request package and one response package.



2.1.2 Package Specifications and Namespaces

The extension package must have the appropriate XSD tool attributes set so the script can generate the namespaces correctly. Below is an example of how these attributes are set for the Fix Holding Data sub-package.



There are five properties that are needed for each new Application Schema package being used to generate XML Schemas. These properties are highlighted below with the Source as 'Override'.

To modify the value of these properties, open the Package Specification and navigate to the XSD tab. The targetNamespace and targetNamespacePrefix property values are determined by the COI and used in accordance with other related schemas and will determine if an external import is included or imported.

Additionally, denote the generateFilename property as applicable so the schema is named consistently each time it is generated with the UML to XSD scripts.

The following was generated for the Fix Holding Data Application Schema package.

```
<schema xmlns:fix="http://www.faa.gov/FPP/fixes"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:aixm="http://www.aixm.aero/schema/5.1"
xmlns:dpshare="http://www.faa.gov/avnis/shared"
targetNamespace="http://www.faa.gov/avnis/fixes"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

2.2 UML Extension Package

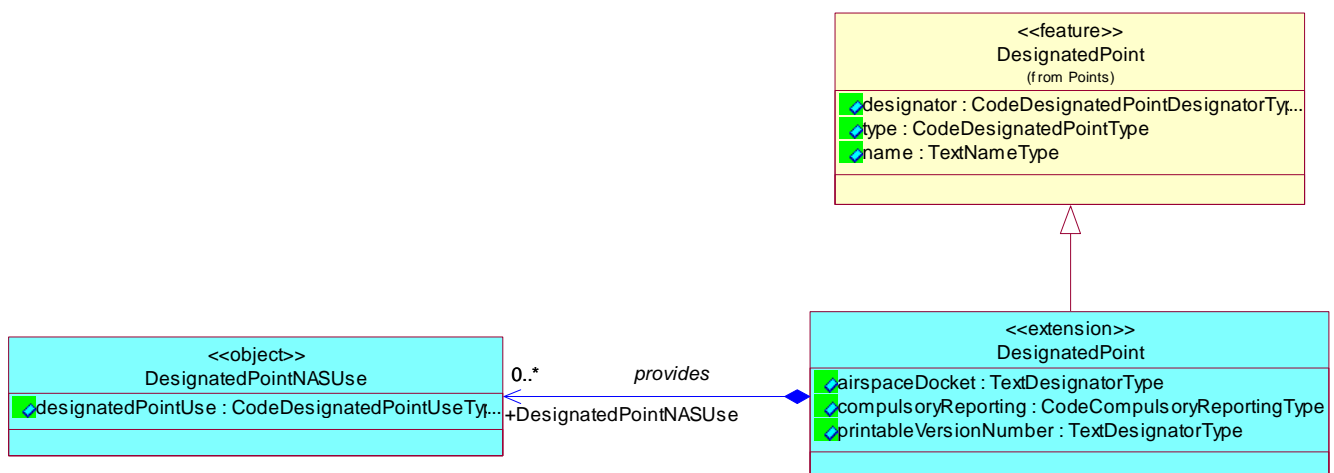
2.2.1 Overview

A feature or object may be extended by creating a class with the same name as the core AIXM feature and giving it a stereotype <<extension>>. This new class can contain:

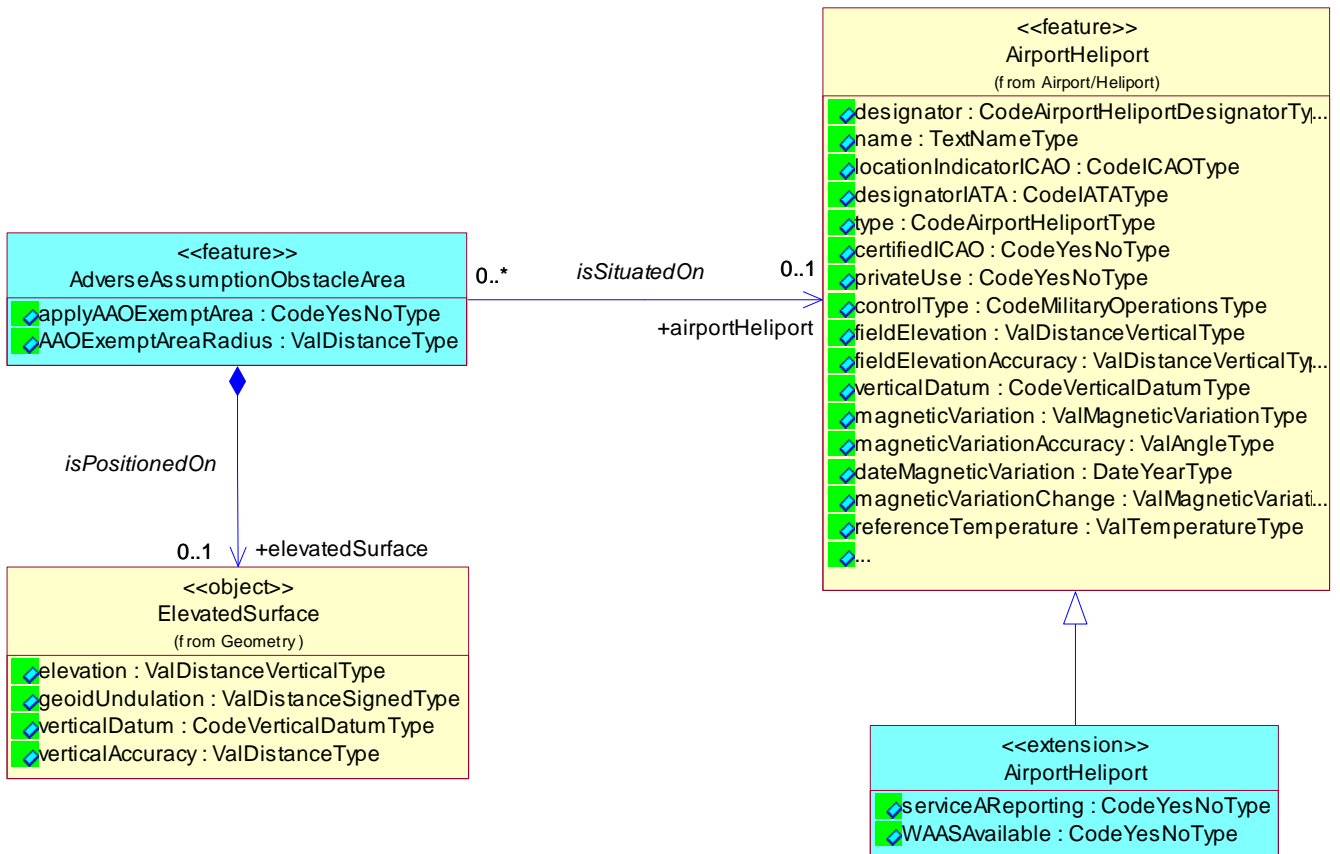
- ✓ New attributes
- ✓ New associations. ***Important Note:*** if a core AIXM class is involved, the navigability of the association should always be from the <<extension>> class towards the core AIXM class.

In addition, it is possible in extensions to create declare totally new classes (features and objects), that do not extend existing AIXM Core classes. The only rule is to follow the AIXM UML modelling conventions described at the beginning of the document. This will enable the script AIXM-FeatureGenerator.ebs to correctly generate the XML elements for these new classes. This situation is not described in detail in this document because it does not require any special action. Just follow the UML modeling conventions.

The example below shows the modelling convention used to extend the DesignatedPoint feature. The example adds a new attribute to DesignatedPoint and a new relationship to a new object called DesignatedPointNASUse.



Associations can also be created between new features or objects and AIXM Core features or objects as depicted below in the association between AdverseAssumptionObstacle and AirportHeliport. The new association should be created in the Application Schema package and towards the AIXM Core Feature rather than an extension (if present). This action ensures the relationship is represented properly in the XML Schema



Use the approved rules for AIXM Core elements to produce new features or objects. Some rules that apply to new *extension* classes are:

- The extension class stereotype must be <<extension>>.
- The extension class name for the extension must match the class you are extending. (When using Rational Rose, it is possible to do that if you create a new class in the navigation menu at the left and change the name of that class; only afterwards drag and drop the class on the diagram.)
- The extension class must be a specialized class extending the matching base class.
- The extension class attributes are added to the extension class the same as they would a regular AIXM class (Data Types are discussed later in this document).

2.2.2 XML Schema Generation

Use the AIXM-FeatureGenerator.ebs script to generate the extension XML Schema in which the script triggers the generation of an extension element by recognizing the <<extension>> stereotype. Generation of the extension follows the AIXM generation rules.

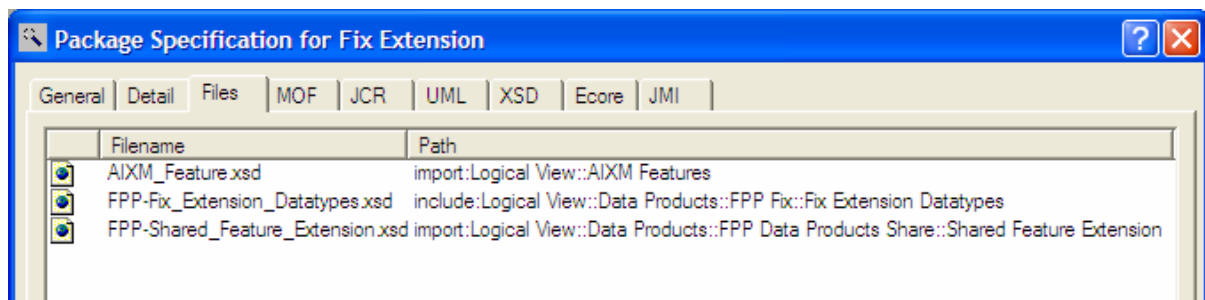
If new data types or codelists are introduced the script, AIXM-DataTypeGenerator.ebs, must be executed first on the associated Data Type Package.

2.2.2.1 Imported and Included Schemas

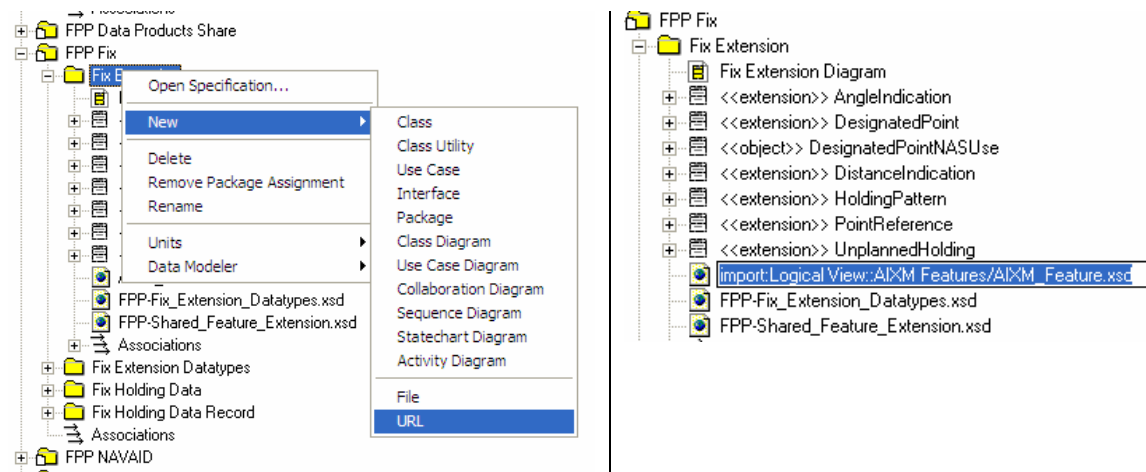
Every application schema sub-package must incorporate imported XML Schemas from the AIXM Core Schemas. Additionally, if new data types or codelists are introduced, the schema from that sub-package must be included. It is sometimes required to utilize common objects from a 'shared' package to increase object re-use. These elements should be incorporated as well by importing the schema.

It is not required for these schemas to be generated for the script to run in Rational Rose, but if they are not created and in the folder structure when the schema is opened it will have errors. The script, AIXM-DataTypeGenerator.ebs, is used to create the schema on the associated Data Type sub-package.

These linked schemas, essentially URL's, can be incorporated by entering them on the Files tab of the Package Specification.



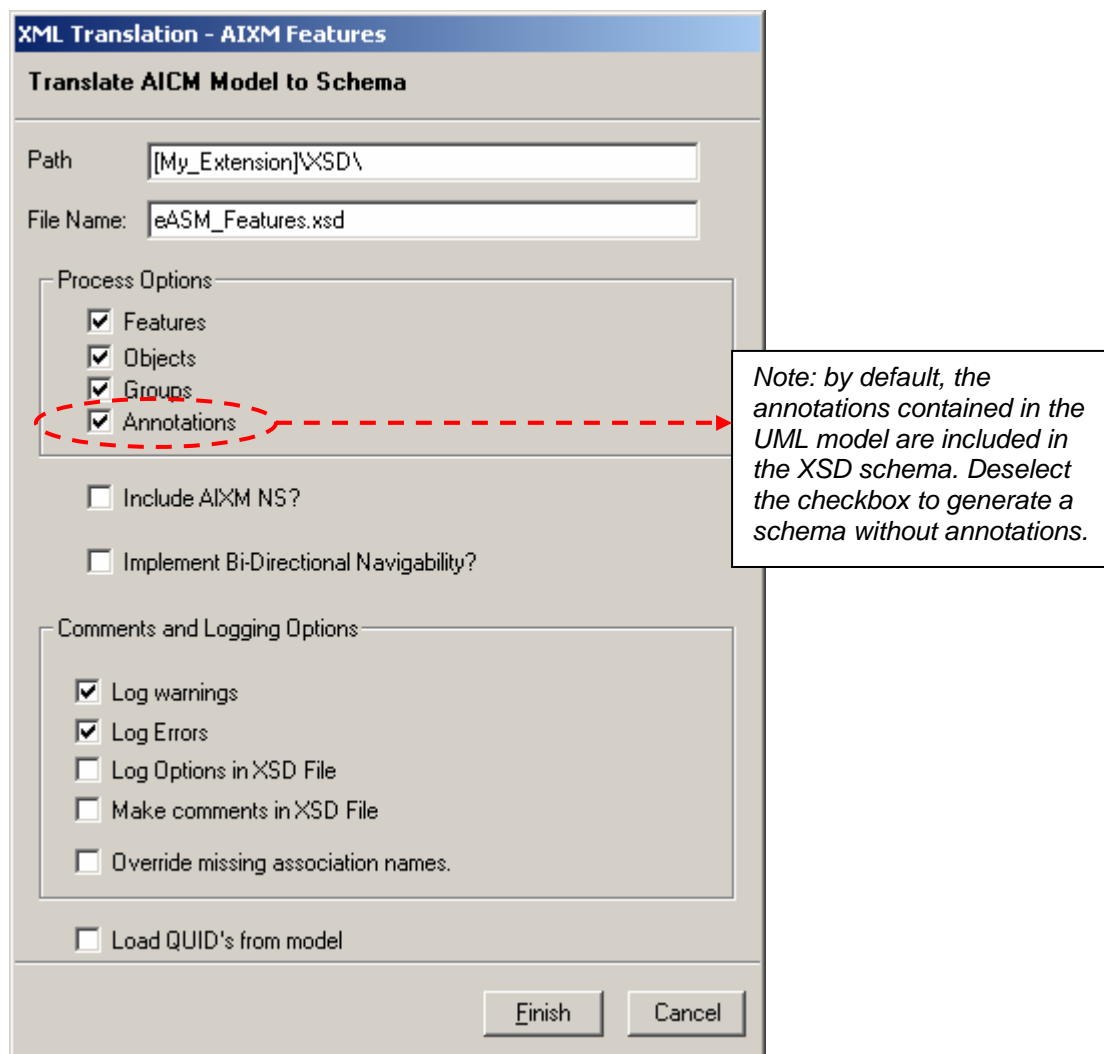
They can be added also be added in the navigation window by entering the full path of the schema location within the model.



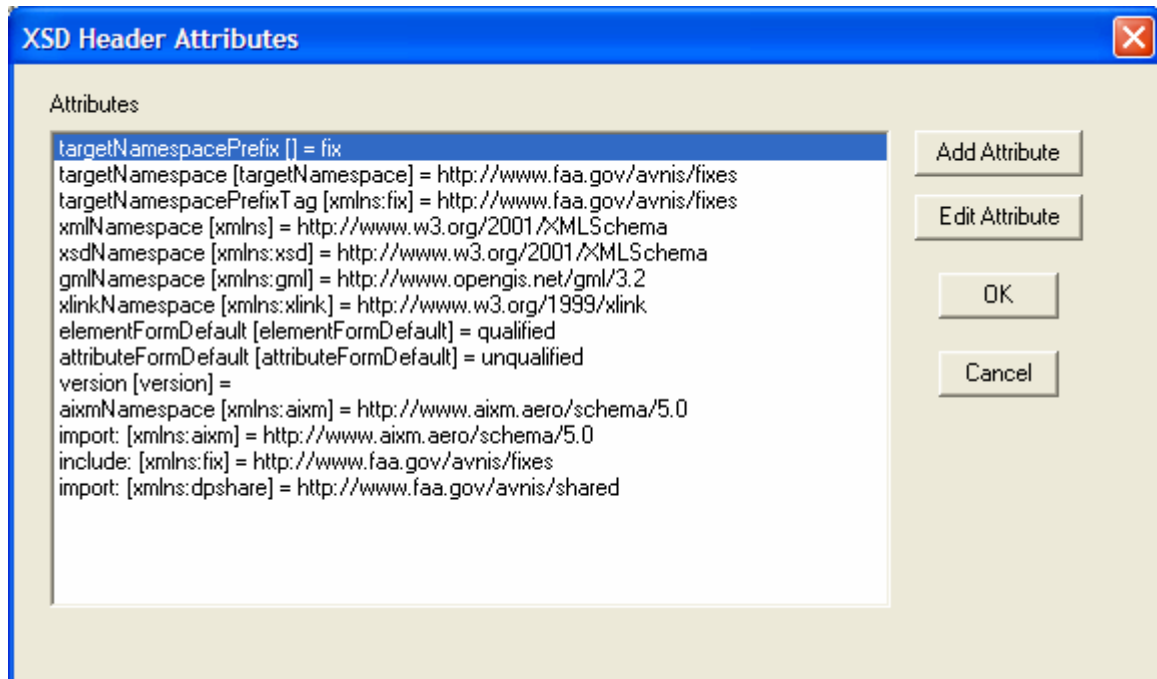
2.2.2.2 Executing the Script

There are some specific options that need to be set when executing the script in Rational Rose, but most of the options will be defaulted. In the image below, notice the File Name is pulled from the generateFilename property set in the package specification earlier.

The schema generation scripts are used for both AIXM Core and Application Schemas. The checkbox for 'Include AIXM NS' is selected when executing scripts for packages that are not part of the AIXM Core set since the AIXM Namespace is automatically included for those schemas. Furthermore, select the checkbox for 'Load QUID's from model' when new classes have been added to the model since the script was last run, which ensures the element identifiers are correctly recognized.



After selecting 'Finish' and regenerating the QUID's (if needed), a dialogue will open to allow the addition or editing of the XSD Header attributes. It should not be necessary to make any changes, but notice the targetNamespace attributes generated from the package specifications set earlier. After selecting 'OK', the XML schema file will be generated and can be found in the directory in which the script was run.

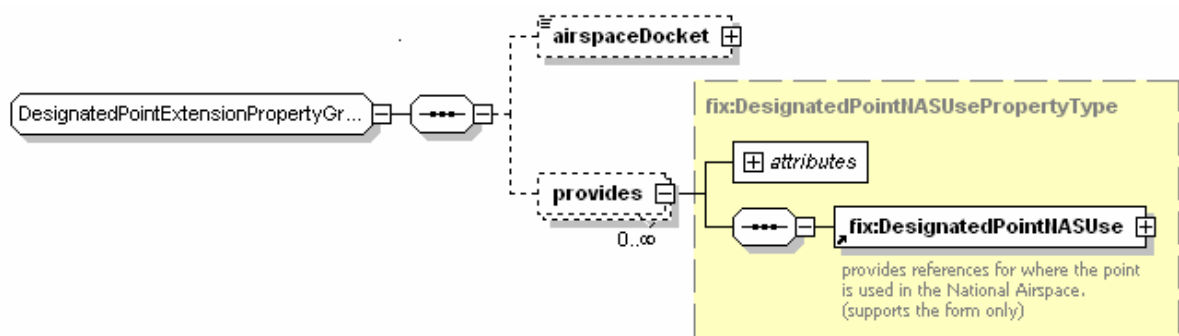


2.2.2.3 XML Schema Output

Classes with the stereotype of <<extension>> generate three related elements for that class.

- <classname>ExtensionPropertyGroup
- <classname>ExtensionType
- <classname>Extension

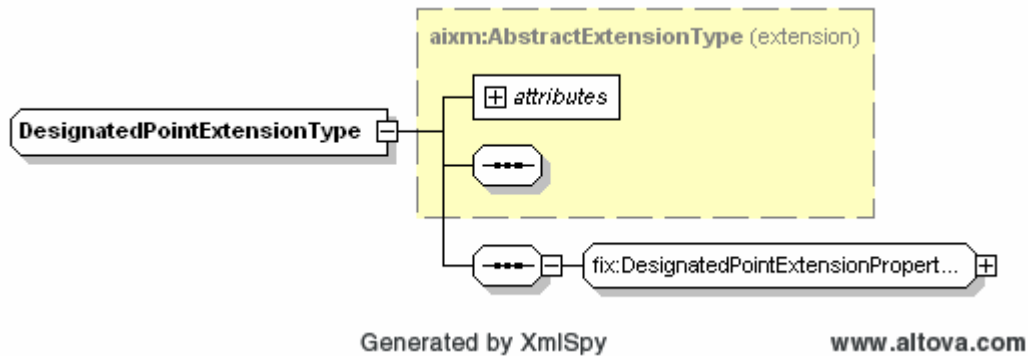
The <classname>ExtensionPropertyGroup contains the properties (elements and relationships) of the Extension.



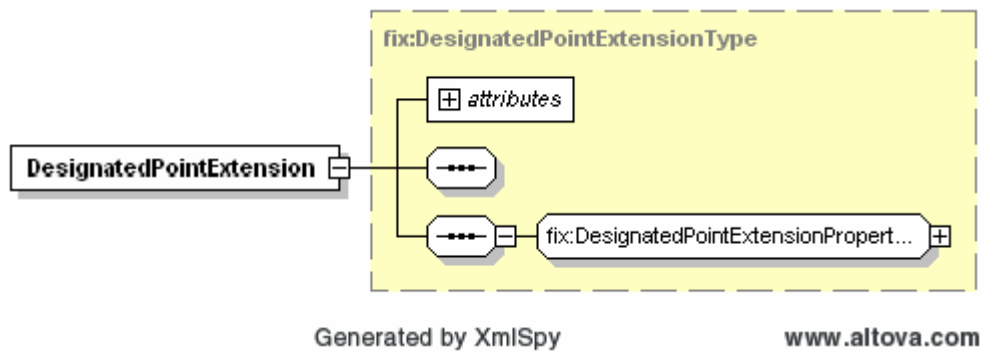
Generated by XmlSpy

www.altova.com

The <classname>ExtensionType element is generated as a XMLSchema <complexType.> and extends base type aixm:AbstractExtensionType.



The <classname>Extension element is generated as a XMLSchema <element>. The Extension element cannot stand alone, it may only exist as an extension to an AIXM base element. The Extension element does not have a timeslice. The Extension element attribute substitutionGroup is the substitutionGroup of the base type extension. Extension elements are not extensible.

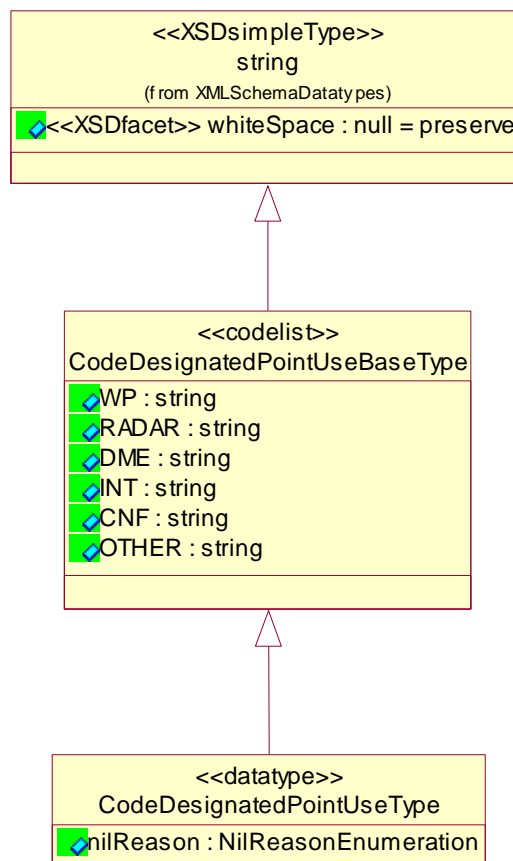


2.3 Data Type Extension Package

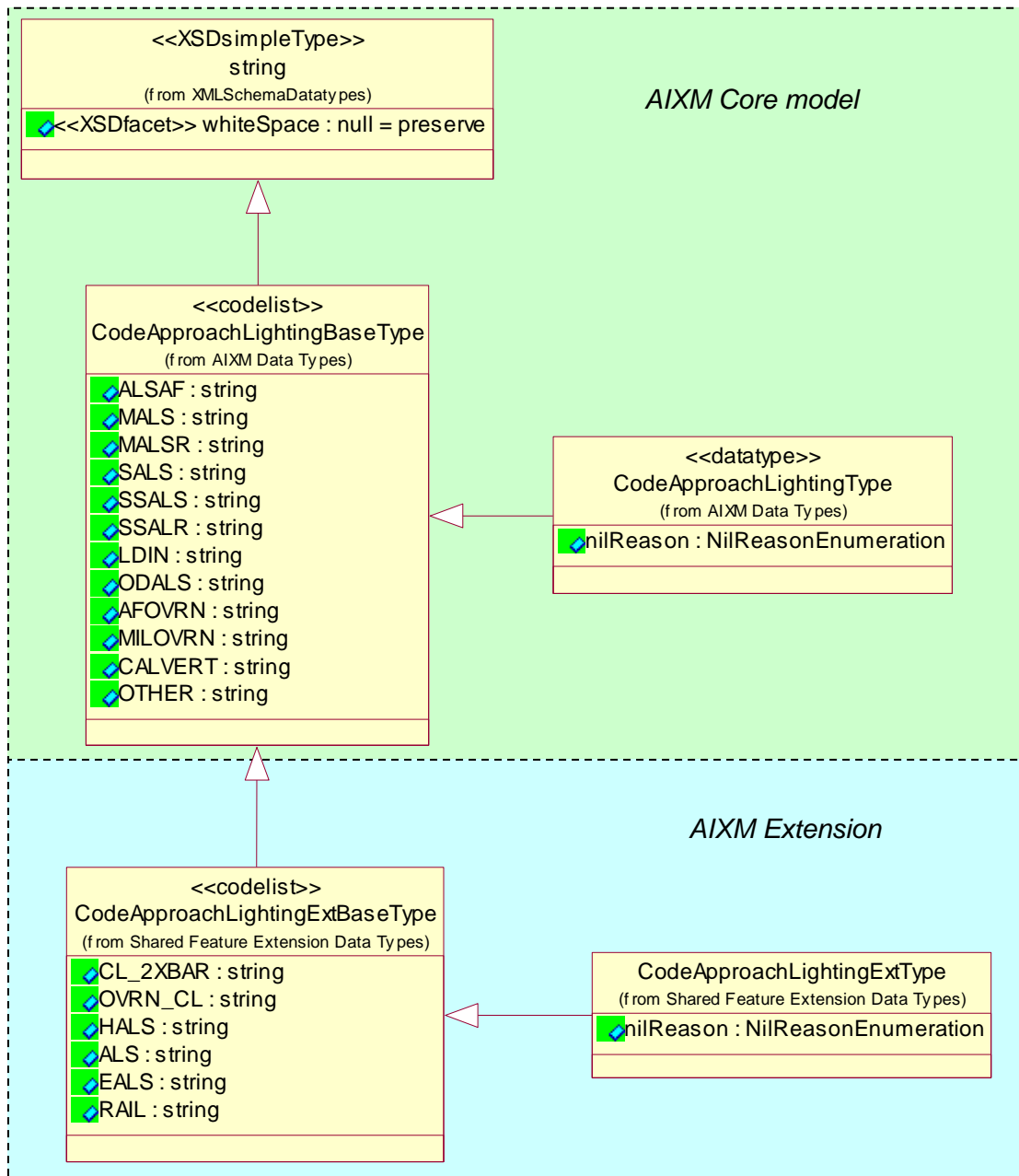
2.3.1 Overview

An extension, new feature or object class may require additional data types or codelists to capture the valid values for new attributes added to the class. To add new data types or codelists create a Data Type sub-package containing any new data types needed.

In the example below, an <<codelist>> is defined in an extension package. It is called CodeDesignatedPointUseBaseType, it has a generalization to the 'string' class and inherits the basic attributes of an XSD string variable. The <<datatype>> CodeDesignatedPointUseType is created with the nilReason property, as specified in [4]. This is the most common configuration for codelists.



AIXM Core <<codelists>> can also be extended in the Data Type sub-package. Extend a codelist by creating a class with the same name as the codelist and giving it a stereotype <<codelist>>.



Careful analysis must be done to ensure that the extended list of values remains normalised. It shall not duplicate values that already exist in the core <<codelist>> (including the OTHER value), but with other names/abbreviations.

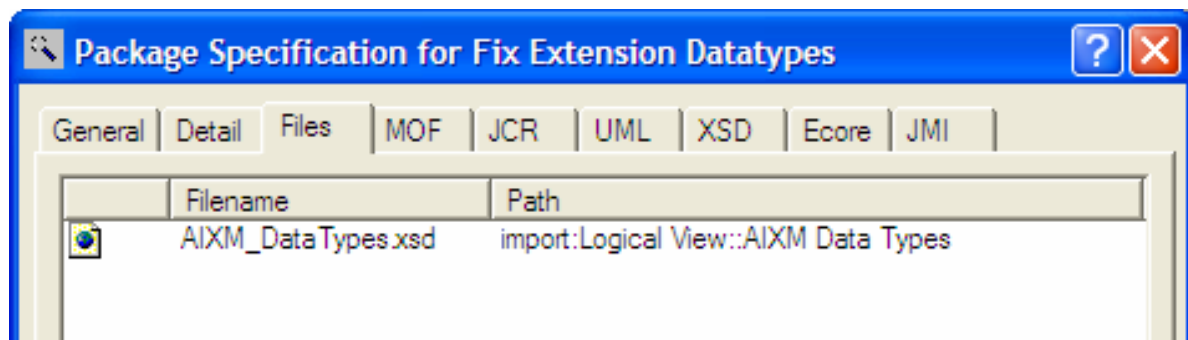
If additional values are required for an AIXM Core codelist for international data exchange, then AIXM Core model will need to be updated.

2.3.2 XML Schema Generation

Use the AIXM-DataTypeGenerator.ebs script to generate the data type extension XML Schema. Data Types are generated as a XMLSchema <simpleType> with the appropriate facets, Patterns and/or codelists defined.

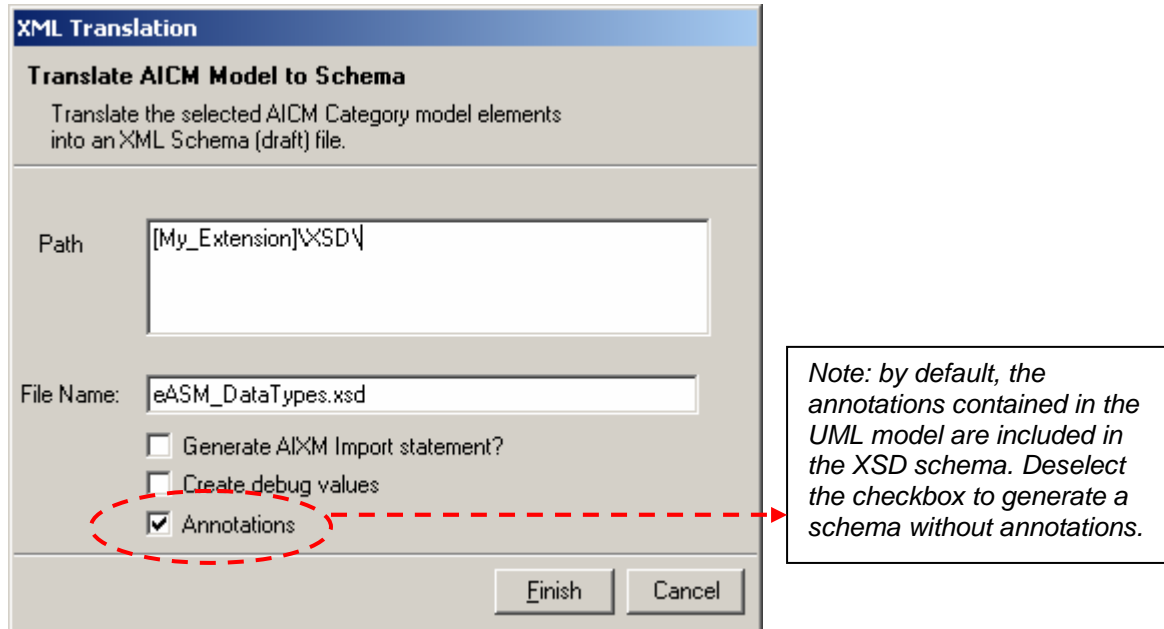
2.3.2.1 Imported and Included Schemas

Each data type sub-package must incorporate the AIXM Core Data Type schema. It is not required for this schema to be generated for the script to run in Rational Rose, but if the AIXM Core Data Type schema is not created and in the folder structure when the schema is opened it will have errors.



2.3.2.2 Executing the Script

In the image below, notice the File Name is pulled from the generateFilename property set in the package specification earlier. However the Path denotes the location of the UML model file, which should be changed appropriately. The checkbox for 'Include AIXM NS' is selected when executing scripts for packages that are not part of the AIXM Core set since the AIXM Namespace is automatically included for those schemas.

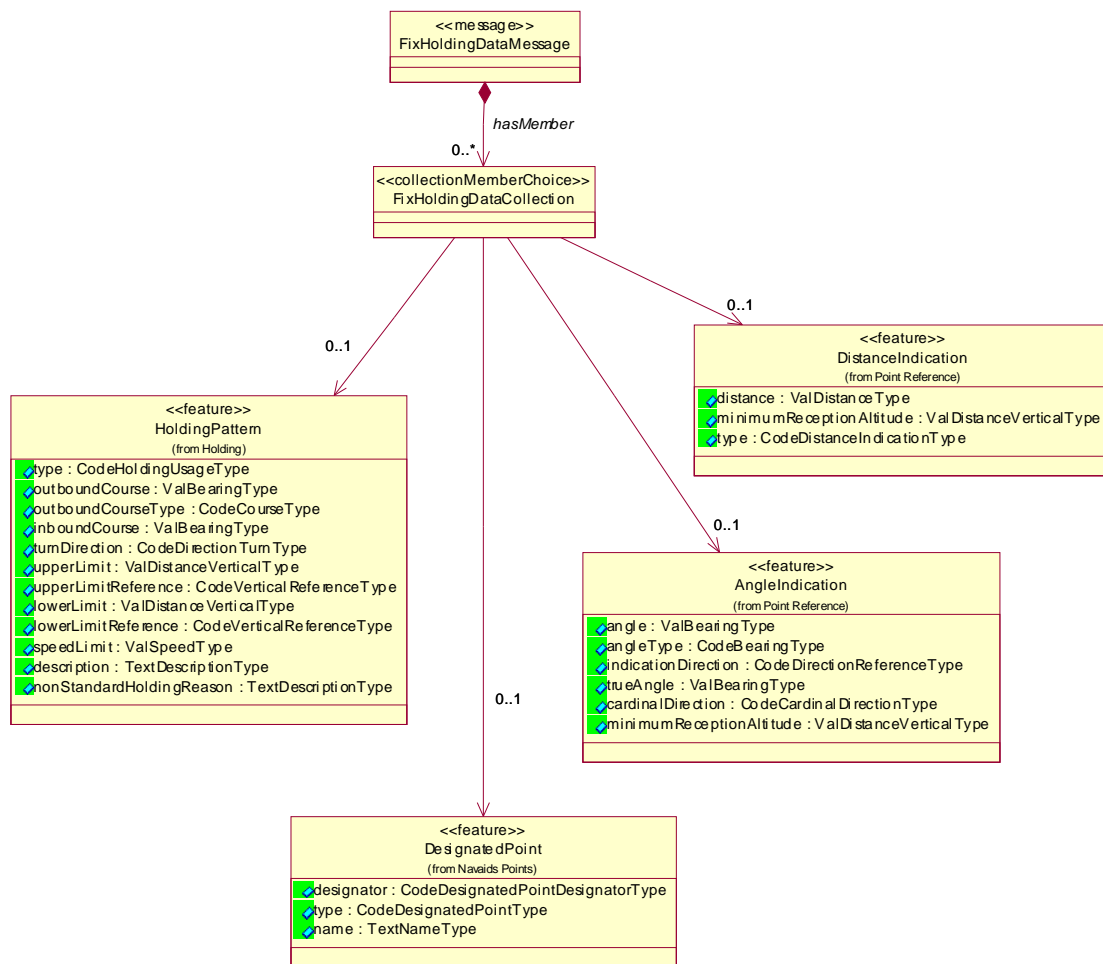


2.4 UML Message Package

2.4.1 Overview

The message package is used to generate an XML Schema for request and response messages. Below is an example of the FixHoldingData Response Message. This message includes the extensions described previously even though they do not appear in the diagram.

A Message is modelled in UML using the class object with a stereotype <<message>>. In this example the message is a limited collection of AIXM features with extensions. This is modelled by relating the collection of features; <<collectionMemberChoice>> to the message through the relationship “hasMember”.

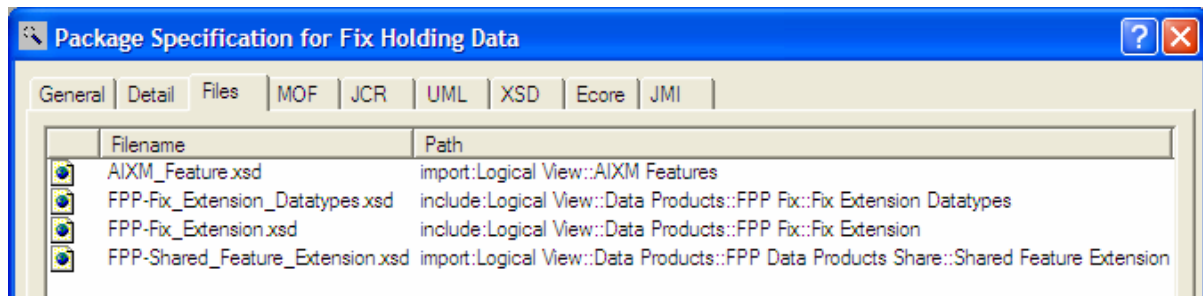


2.4.2 XML Schema Generation

Use the AIXM-ApplicationSchemaGenerator.ebs script to generate the message XSD. The script triggers the generation of the message element by recognizing the <<message>> stereotype.

2.4.2.1 Imported and Included Schemas

The UML Message sub-package brings together all of the related elements created in earlier processes such as extensions and data types. As before, the AIXM Core schema must be included as well as any other referenced schemas (i.e. shared or common objects that are to be used in multiple application schemas).



The accumulation of the imported and included XML Schemas is displayed below.

```
<import namespace="http://www.opengis.net/gml/3.2"
  schemaLocation="./ISO_19136_Schemas/gml.xsd"/>
<import namespace="http://www.aixm.aero/schema/5.1"
  schemaLocation="./AIXM_Feature.xsd"/>
<import namespace="http://www.w3.org/1999/xlink" schemaLocation="./xlink/xlinks.xsd"/>
<import namespace="http://www.faa.gov/avnis/shared" schemaLocation="./FPP-
  Shared_Feature_Extension.xsd"/>
<include schemaLocation="FPP-Fix_Extension_Data_Types.xsd"/>
<include schemaLocation="FPP-Fix_Extension.xsd"/>
```

2.4.2.2 Executing the Script

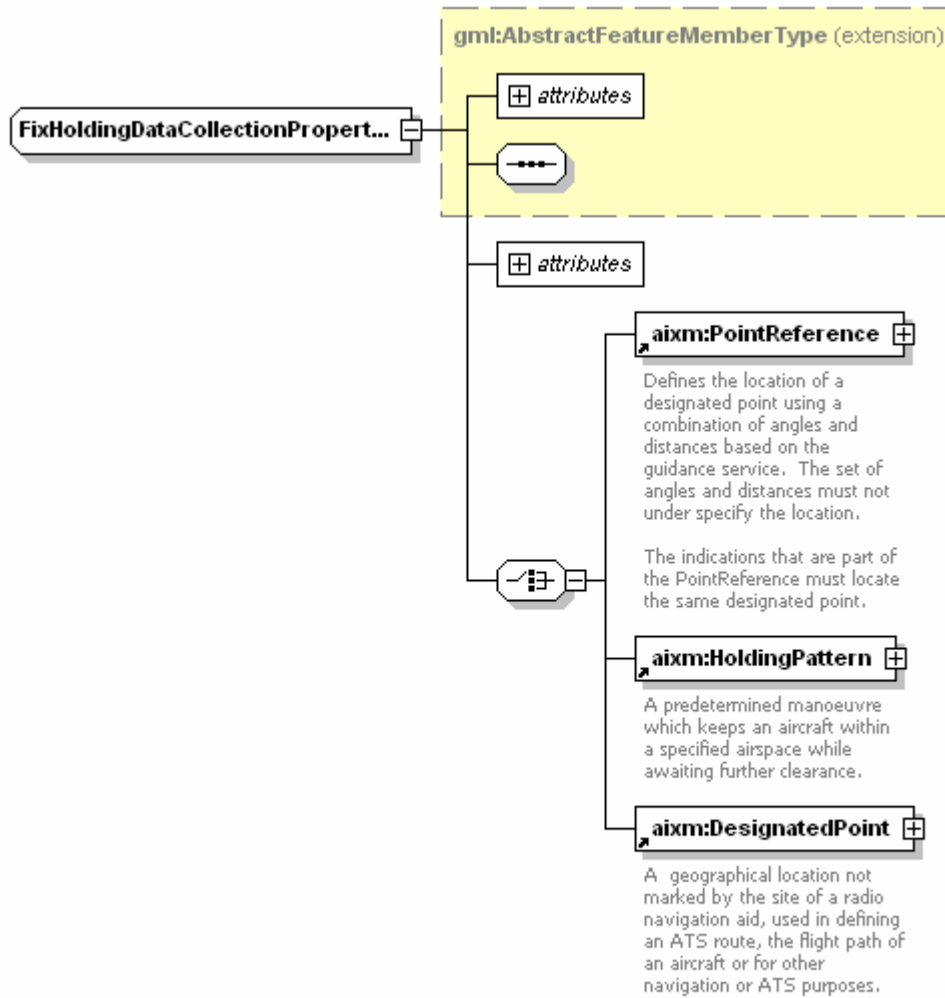
Follow the procedures outlined in section 2.2.2.2.

2.4.2.3 XML Schema Output

Classes with the stereotype of <<message>> following the AIXM feature collection response generates four related elements for that class.

- <classname>CollectionPropertyGroup
- <classname>MessagePropertyGroup
- <classname>MessageType
- <classname>Message

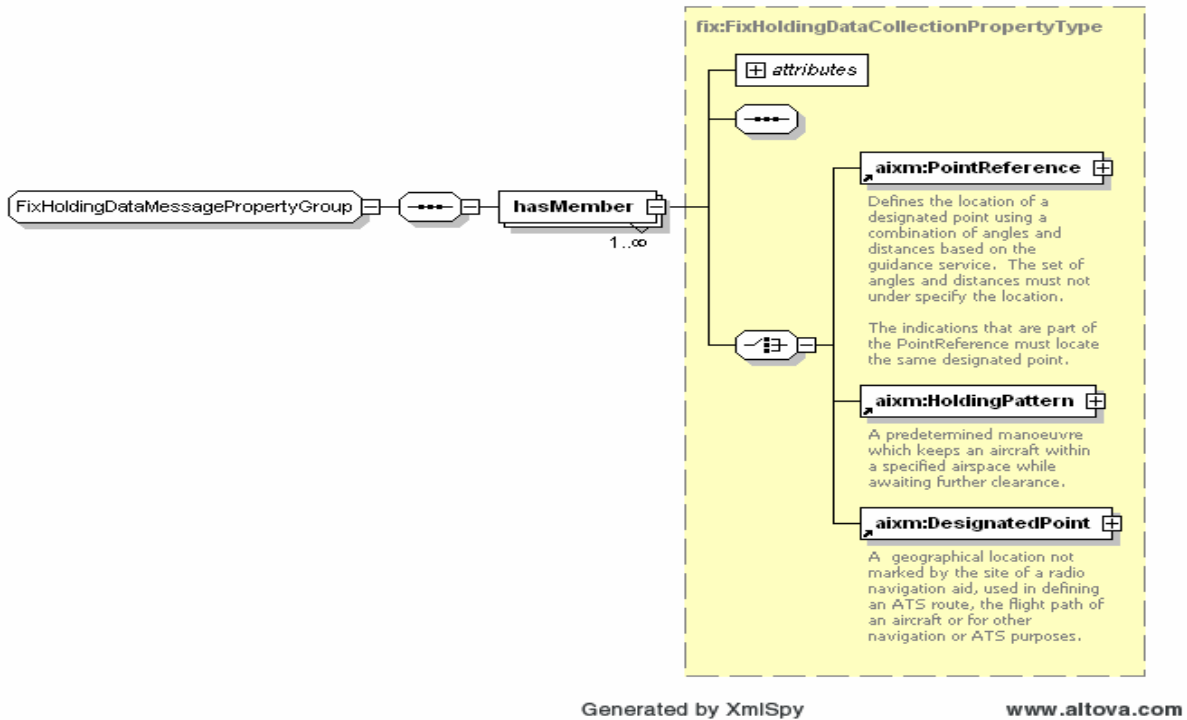
The <classname>CollectionPropertyGroup is generated as a XMLSchema <complexType>, which extends gml:AbstractFeatureMemberType, and includes a <choice> between the all the features it is pointing to.



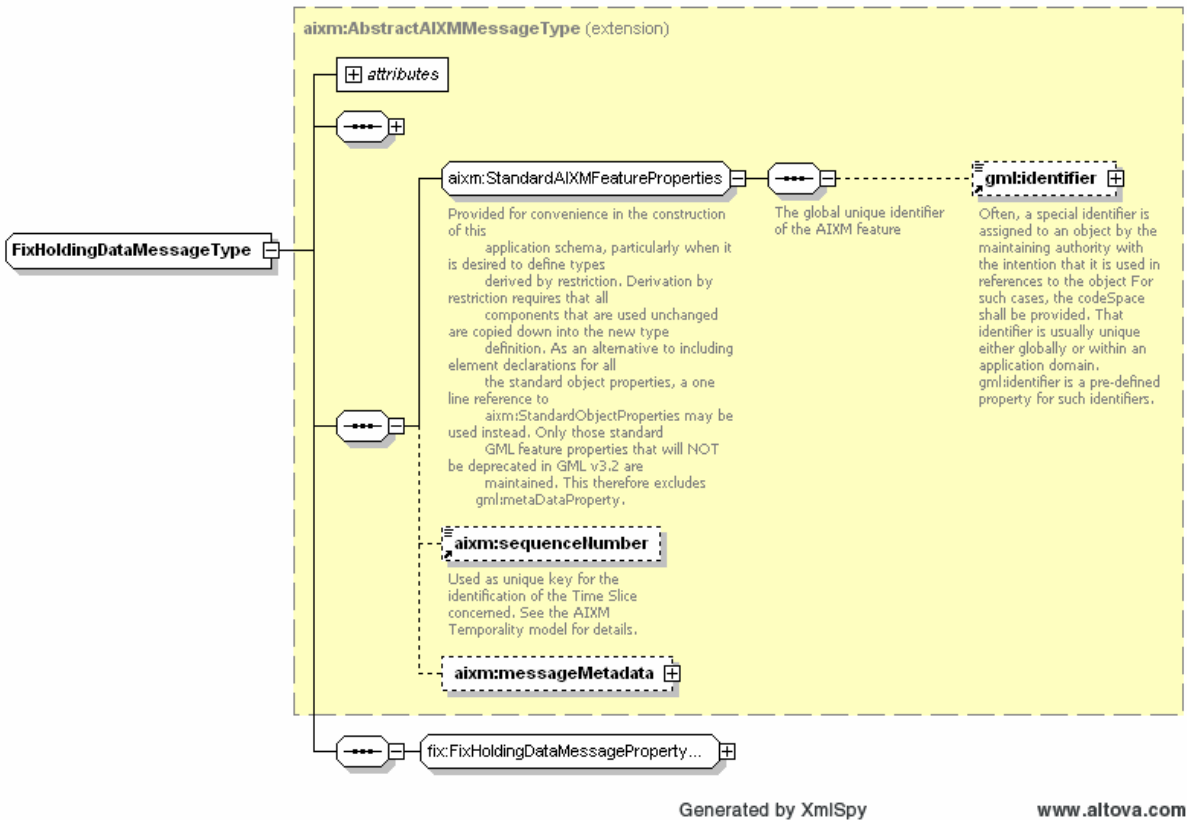
Generated by XmiSpy

www.altova.com

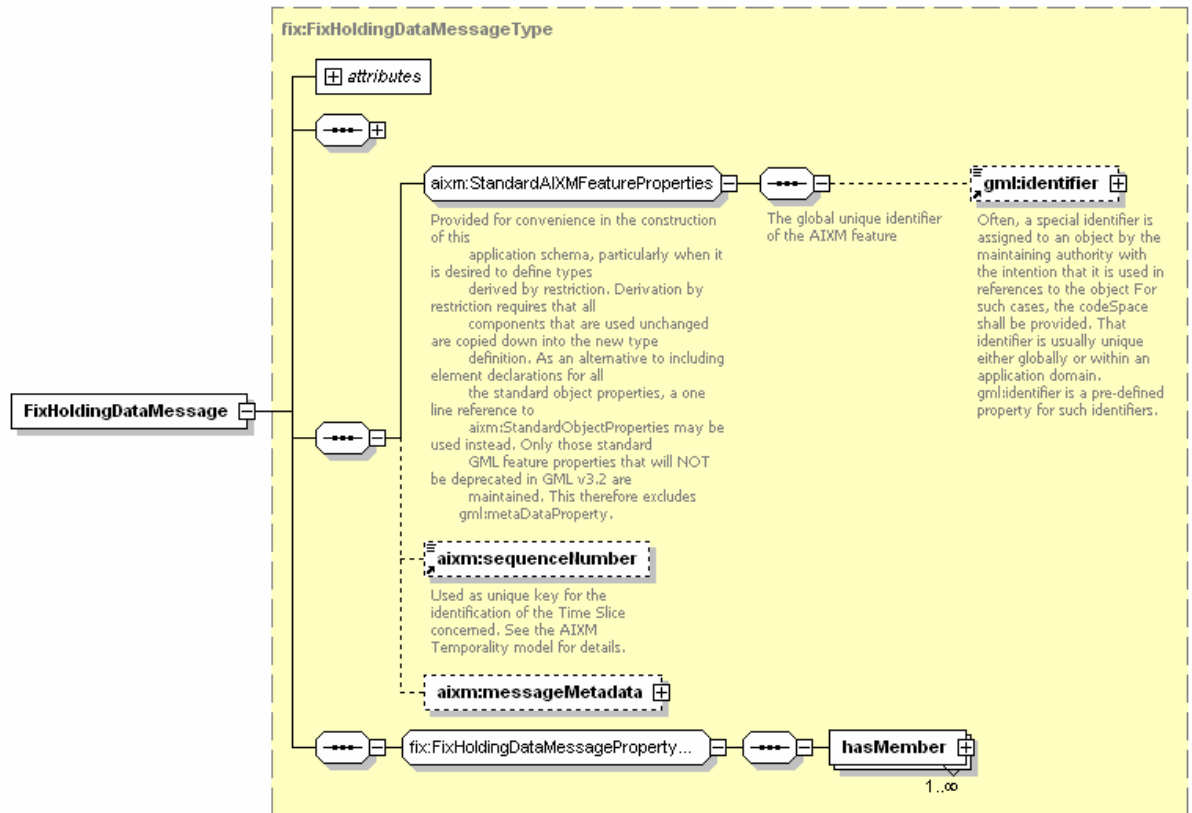
The <classname>MessagePropertyGroup is generated as a XMLSchema <group>, which contains the properties (elements and relationships) of the Message.



The <classname>MessageType element is generated as a XMLSchema <complexType> and extends base type aixm:AbstractAIXMMessageType.



The <classname>Message element is generated as a XMLSchema <element>. The associations are treated as objects. They are included in the schema.



Generated by XmlSpy

www.altova.com