# Introduction

## Purpose and content

This document provides guidance and clarifications for the implementation of FIXM. It aims to build a "community knowledge" about the implementation of FIXM. It will therefore evolve over time in order to capture more alternatives, options and recommendations related to the use of FIXM.

The implementation guidance provided in this document is structured as follows:

* **General Guidance on FIXM implementation**

This chapter provides general guidance for understanding the main FIXM components, outlines general requirements for valid FIXM core and FIXM extensions usage and describes the general rules (encoding rules, data plausibility rules, rules for absent data…) that are always applicable whatever the implementation context.

* **Guidance for** **Using FIXM in Support of FF-ICE**

This chapter provides specific guidance in support of the implementation of FF-ICE using FIXM. It introduces the FF-ICE Application Library, identifies a candidate set of FIXM-Based services enabling (part of) the information exchanges defined by the ICAO ATMRPP in the Manual on FF-ICE Implementation Guidance [10] and provides detailed examples of FF-ICE/R1 Services realizations.

* **Guidance for** **Translating FF-ICE FIXM Messages to ATS Messages**

This chapter defines a formal mapping between the FIXM Logical Model and Air Traffic Services (ATS) message content as defined in ICAO Doc 4444 [8].

* **Guidance for** **Using FIXM for other use cases**

This chapter provides guidance for using FIXM in a context other than FF-ICE. In particular, it helps FIXM implementers create their own application libraries and extensions.

*Note: The present version of the document captures the guidance information agreed by the FIXM community as of March 2018. Readers are invited to monitor the FIXM community discussions in the FIXM Work Area* *[6] about the implementation of FIXM, which may serve as useful complement or clarification.*

## Applicable FIXM version

The present version of the document supports the implementation of the following FIXM components:

* **FIXM Core v4.2.0** [1]
* **FIXM Applications**
  + **FF-ICE Message v1.0.0**
  + **Basic Message v1.0.0**

**What is new in FIXM Core 4.2.0?**

At high level, the scope declaration of FIXM Core 4.2.0 is the same as FIXM 4.1.0, namely to provide harmonised representation of the flight data structures exchanged in the context of FF-ICE/R1. FIXM Core 4.2.0, however, implements significant improvements compared to FIXM 4.1.0:

* FIXM Core 4.2.0 is based on the draft FF-ICE/R1 Implementation Guidance Manual version 0.91, therefore reflecting a more mature - but yet not final - version of the FF-ICE/R1 requirements specified by the ICAO ATMRPP;
* FIXM Core 4.2.0 contains more usable data structures for enabling a better representation of the feedback that an eAU can get from an eASP after submitting an FF-ICE flight plan;
* FIXM Core 4.2.0 supports nillable properties (which FIXM 4.1.0 did not)[1](https://www.fixm.aero/fixm_410.pl) that enable proper FF-ICE Flight Plan updates as described in the FF-ICE/R1 Implementation Guidance Manual;
* FIXM Core 4.2.0 comes together with a new FF-ICE Application Library that addresses the use of FIXM Core in the specific context of FF-ICE and which provides formal representation of the individual FF-ICE messages. A new “Basic Message” library is also available in order to provide basic messaging support for FIXM.
* FIXM Core 4.2.0 also implements a number of technical improvements and bug corrections.

Important note: This document does not detail the individual changes implemented in this new FIXM release. More information about these changes can be found in the FIXM Release Note and in the online repository of FIXM Change Requests from the FIXM Work Area. This document rather focuses on how to use the various FIXM components and data structures.

## Document terms

This is a guidance document describing recommended practices related to the use and implementation of FIXM. It has been subject to FIXM CCB review and endorsement and is therefore the official recommendation of the FIXM CCB.

This document is, however, non-normative and requirements described in it shall be considered mandatory only for those aiming to comply with this guidance.

The use of the word "SHALL" or "REQUIRED" indicates an absolute requirement of this guidance, i.e. a requirement to be strictly followed in order to conform to this document.

The use of the word "SHOULD" or "RECOMMENDED" indicates that there may exist valid reasons, in particular circumstances, to ignore a particular aspect of the guidance.

## How to improve this document

The FIXM implementation guidance evolves based on the feedback of the FIXM users. Improvement proposals, such as new content or corrections, can be posted at any time to the [FIXM Work Area](https://ost.eurocontrol.int/sites/FIXM/SitePages/Home.aspx), using the FIXM Community discussion forum. Improvement proposals can also be sent by email to the FIXM CCB ([FIXM.CCB@eurocontrol.int](mailto:FIXM.CCB@eurocontrol.int)) or alternatively to the FIXM Secretariat ([fixm.secretariat@eurocontrol.int](mailto:fixm.secretariat@eurocontrol.int)).

## Acronyms and Definitions

|  |  |
| --- | --- |
| **Acronym** |  |
| AIDC | ATS Interfacility Data Communications |
| AIXM | Aeronautical Information Exchange Model |
| AMQP | Advanced Message Queuing Protocol |
| AMXM | Aerodrome Mapping Exchange Model |
| ASBU | Aviation System Block Upgrade |
| ASP | ATM Service Provider |
| ATM | Air Traffic Management |
| ATMRPP | ATM Requirements and Performance Panel |
| ATS | Air Traffic Services |
| AU | Airspace User |
| CCB | Change Control Board |
| CDM | Collaborative Decision Making |
| eASP | Enhanced ATM Service Provider (i.e. FF-ICE capable ATM service provider) |
| eAU | Enhanced Airspace User (i.e. FF-ICE capable Airspace user) |
| FF-ICE | Flight and Flow Information for a Collaborative Environment |
| FIR | Flight Information Region |
| FIXM | Flight Information Exchange Model |
| FPL | Flight Plan |
| GML | Geography Markup Language |
| GUFI | Globally Unique Flight Identifier |
| ICAO | International Civil Aviation Organisation |
| IFR | Instrument Flight Rules |
| ISO | International Organization for Standardization |
| Navaid | Navigational Aid |
| OGC | Open Geospatial Consortium |
| PBN | Performance Based Navigation |
| R/R | Request/Reply |
| SID | Standard Instrument Departure |
| SOAP | Simple Object Access Protocol |
| SSR | Secondary Surveillance Radar |
| STAR | Standard Terminal Arrival Route |
| SWIM | System Wide Information Management |
| UPR | User Preferred Route |
| URL | Uniform Resource Locator |
| US NAS | US National Airspace System |
| UTC | Coordinated Universal Time |
| VFR | Visual Flight Rules |
| WGS-84 | World Geodetic System - 1984 |
| WSDL | Web Services Description Language |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |
| XSLT | Extensible Stylesheet Language Transformations |

|  |  |
| --- | --- |
| **Term** | **Definition** |
| FIXM XML schemas | The XML-based physical realization of the FIXM Logical Model |
| FIXM-based message | A message exchanged by a flight information service which has a content that satisfies the applicable FIXM requirements in terms of data structure, data completeness and data correctness. |
| FIXM-based service | A service created in accordance with the recommendations described in this guidance document. |

## References

**FIXM references**

1. [FIXM 4.1.0 Core](https://www.fixm.aero/fixm_410.pl), including the
   * [FIXM 4.1.0 Core XML Schemas](https://www.fixm.aero/releases/FIXM-4.1.0/FIXM_Core_v4_1_0_Schemas.zip) (Zip archive)
   * [Schema documentation](https://www.fixm.aero/releases/FIXM-4.1.0/doc/schema_documentation_core/index.html)
2. FIXM 4.1.0 XML samples
3. [FIXM Strategy v1.1](https://www.fixm.aero/documents/FIXM%20Strategy.pdf)
4. [FIXM 4.1.0 Modelling Best Practices](https://www.fixm.aero/releases/FIXM-4.1.0/FIXM_Core_v4_1_0_Modelling_Best_Practices.pdf)
5. [FIXM web site](http://www.FIXM.aero)
6. [FIXM Work Area](https://ost.eurocontrol.int/sites/FIXM/SitePages/Home.aspx)
7. [US NAS extension to FIXM 4.1.0](https://www.fixm.aero/fixm_nas_extension_420.pl)

**ICAO references**

1. PANS-ATM: Procedures for Air Navigation Services: Air Traffic

* Management, ICAO Doc 4444, 16th edition

1. [ICAO Doc

* 9965](<http://www.icao.int/Meetings/anconf12/Documents/9965_cons_en.pdf>): Manual on Flight and Flow Information for a Collaborative Environment

1. [ATMRPP/3-WP/766](https://ost.eurocontrol.int/sites/FIXM/Shared%20Documents/ICAO%20ATMRPP%20inputs%20for%20FIXM%204.1.0/ATMRPP3_WP_766_FF-ICE1%20Implementation%20Guidance_All.pdf)

* “Manual on FF-ICE Implementation Guidance”

1. ICAO Doc 7910: Location Indicators
2. [ICAO Doc

* 8643](<https://www.icao.int/publications/DOC8643/Pages/default.aspx>): Aircraft Type Designators

1. [ICAO Doc

* 9854](<http://www.icao.int/NACC/Documents/Meetings/2012/ASBU/Reference3.pdf>): Global Air Traffic Management Operational Concept, 1st edition

1. PAN AIDC ICD: PAN Regional (NAT and APAC) Interface Control Document

* for ATC Interfacility Data Communications (PAN AIDC ICD), version 1.0

1. ICAO Doc 10039: Manual on System Wide Information Management (SWIM)

* Concept

1. ATMRPP-WG/24-WP/564: Flight Plan Filing Provisions for FF-ICE

**Other references**

1. [Donlon AIP data Set](https://github.com/aixm/donlon): a fictitious set of digital AIS data sets complying with the ICAO Annex 15, 16th edition and the new PANS-AIM provisions, in AIXM 5.1.1 format.
2. [W3C XML Linking Language (xlink)

* v1.1](<https://www.w3.org/TR/xlink11/>)

The 'Donlon AIP data Set' is distributed under the following BSD license.

# General Guidance on FIXM implementation

## Target audience

This chapter provides general guidance for understanding the main FIXM components, outlines general requirements for valid FIXM core and FIXM extensions usage and describes the general rules (encoding rules, data plausibility rules, rules for absent data…) that are always applicable whatever the implementation context. It therefore targets all FIXM implementers.

## Understanding FIXM Core, the Application Libraries and the Extensions

### FIXM Core

#### What is it?

**FIXM Core** provides globally harmonized flight data structures that can be exchanged in various contexts. The main context for the use of **FIXM Core** is ICAO FF-ICE. Therefore, **FIXM Core** currently captures the flight data structures that are identified in the ICAO FF-ICE Implementation Guidance Manual 0.91. Only flight data structures that are globally applicable qualify for FIXM Core. Flight data structures that are local or regional in nature do not qualify for **FIXM Core**. An **Extensions** mechanism is implemented so that **FIXM Core** can be extended in order to cover these local or regional data structures, as appropriate.

**FIXM Core** exists as a standard for exchanging flight data rather than as a set of pre-defined messages, allowing flexible exchanges between users rather than enforcing rigid communication patterns. However, once a given exchange is well-defined, it is useful to be able to enforce syntax and content validation checks to ensure the data being exchanged is of high quality. This is addressed by **Application Libraries**.

#### What is a valid FIXM Core usage?

The general requirements for a valid **FIXM Core** usage are the following:

|  |  |
| --- | --- |
| **REQUIREMENT ON DATA STRUCTURE** |  |
| **Requirement** | To qualify as valid usage of FIXM Core, the flight-related content of a given message, or relevant part thereof, shall be syntactically valid against the FIXM Core XML Schemas. |
| **Rationale** | The valid usage of FIXM Core implies that the flight-related content of a message exchanged between two parties is valid against the FIXM Core XML Schemas. If a message includes additional information not in scope of FIXM Core, it must be structured so that its relevant part is valid against the FIXM Core XML Schemas. |
| **Important note** | Being syntactically valid against the FIXM Core XML Schemas implies the FIXM Core hierarchy is respected. FIXM Core is not expected to be used only as a library of flight datatypes. |
| **How to check this** | The content of a message, or relevant part thereof, validates without error against the FIXM Core XML schemas when tested / parsed by XML validation tools. |

?> Example of FIXM core usage satisfying the requirement on data structure:

<fx:aerodrome>  
  
 <fb:locationIndicator>EBBR</fb:locationIndicator>  
  
</fx:aerodrome>

This example displays an aerodrome reference involving a four-letter ICAO location indicator. It complies with the structural rules for aerodrome references defined by the FIXM Core XML schemas.

!> Example of FIXM core usage **NOT** satisfying the requirement on data structure:

<fx:aerodrome>  
  
 <fb:locationIndicator>BRU</fb:locationIndicator>  
  
</fx:aerodrome>

This example displays an aerodrome reference based on property locationIndicator. The value “BRU” does not respect the pattern [A-Z]{4} enforced by FIXM for property locationIndicator. This example does NOT comply with the structural rules for aerodrome references defined by the FIXM XML schemas and does not qualify as valid FIXM usage.

This example below features a valid XML schema that defines a Flight Identification structure comprising the departure & arrival aerodrome references, the aircraft identification and the estimated off-block time. It also features an example XML sample that is valid against this schema.

<xs:schema xmlns:wrong=”fixm\_as\_library\_of\_types” xmlns:fx=”http://www.fixm.aero/flight/4.2" xmlns:fb=”http://www.fixm.aero/base/4.2"\[…\] >  
[…]  
 <xs:element name=”FlightIdentification” type=”wrong:FlightIdentificationType”/>  
 <xs:complexType name=”FlightIdentificationType”>  
 <xs:sequence>  
 <xs:element name=”departureAerodrome” type=”fb:AerodromeReferenceType”/>  
 <xs:element name=”arrivalAerodrome” type=”fb:AerodromeReferenceType”/>  
 <xs:element name=”ACID” type=”fb:AircraftIdentificationType”/>  
 <xs:element name=”EOBT” type=”fb:TimeType”/>  
 </xs:sequence>  
 </xs:complexType>  
</xs:schema>

<wrong:FlightIdentification xmlns:wrong=[…] xmlns:fb=”http://www.fixm.aero/base/4.2" xmlns:xs=”http://www.w3.org/2001/XMLSchema-instance" xs:schemaLocation=[…]“>  
 <wrong:departureAerodrome>  
 <fb:name>LES BARAQUES</fb:name>  
 </wrong:departureAerodrome>  
 <wrong:arrivalAerodrome>  
 <fb:name>NORTHFALL MEADOW</fb:name>  
 </wrong:arrivalAerodrome>  
 <wrong:ACID>BLXI</wrong:ACID>  
 <wrong:EOBT>1909-07-25T04:41:00.000Z</wrong:EOBT>  
</wrong:FlightIdentification>  
<!– https://en.wikipedia.org/wiki/Louis\\_Bl%C3%A9riot\#1909\\_Channel\\_crossing –>

The example schema above is not FIXM Core and is not a FIXM extension. It is a fictitious, standalone XML schema that defines its own hierarchy of elements, but which reuses types from the core FIXM XML schemas for typing these elements. The reuse of FIXM datatypes is highlighted in blue in the schema description.

This example illustrates the reuse of FIXM Core as a library of datatypes. While this practice is technically feasible and produces valid schemas, it is not considered a valid FIXM Core usage because it breaks the hierarchy of properties defined by FIXM Core. An information service relying on such an implementation practice would fail to satisfy the FIXM Core requirement on data structure.

|  |  |
| --- | --- |
| **REQUIREMENT ON DATA CORRECTNESS** |  |
| **Requirement** | To qualify as valid usage of FIXM core, the flight-related content of a given message, or relevant part thereof, shall satisfy the minimum set of rules addressing data plausibility and consistency. |
| **Rationale** | The flight-related content of a message being syntactically correct and complete may still not make sense from an operational or plausibility perspective. Additional business rules are required to check the correctness of the encoded information, such as the consistency between model elements. |
| **How to check this** | The content of a message, or the relevant part thereof, validates without error against the applicable business rules addressing data correctness. Chapter 2.4.13 lists business rules addressing data correctness which are always applicable whatever the context of the exchange. Additional business rules addressing data correctness may exist which are specific to particular use-cases. |

Example of FIXM core usage satisfying the requirement on data correctness

<fx:verticalRange>

<fb:lowerBound>

<fb:flightLevel uom="FL">240</fb:flightLevel>

</fb:lowerBound>

<fb:upperBound>

<fb:flightLevel uom="FL">250</fb:flightLevel>

</fb:upperBound>

</fx:verticalRange>

This example shows the FIXM encoding of vertical range [FL240;FL250]. It satisfies the basic data plausibility/correctness rule “*The lowerBound shall always be lower than the upperBound*” that is identified in Chapter 2.4.13. It qualifies as valid FIXM core usage.

Example of FIXM core usage NOT satisfying the requirement on data correctness

<fx:aircraft>

<fx:aircraftType>

<fx:numberOfAircraft>2</fx:numberOfAircraft>

<fx:type>

<fx:icaoAircraftTypeDesignator>MIR2</fx:icaoAircraftTypeDesignator>

</fx:type>

</fx:aircraftType>

<fx:aircraftType>

<fx:numberOfAircraft>1</fx:numberOfAircraft>

<fx:type>

<fx:icaoAircraftTypeDesignator>RFAL</fx:icaoAircraftTypeDesignator>

</fx:type>

</fx:aircraftType>

<fx:formationCount>2</fx:formationCount>

</fx:aircraft>

This example represents a description of a fictitious formation of military aircraft composed of two Mirages 2000 and one Rafale which altogether constitute a single (formation) flight. This example is valid from a data structure point of view (it validates against the FIXM core XML schemas) but is not correct in so far as the sum of all AircraftType.numberOfAircraft properties does not match Aircraft.formationCount, which breaks a rule from Chapter 2.4.13. This example does not qualify as valid FIXM core usage.

### Application Libraries

#### What is it?

An **Application Library** is a FIXM component that addresses the use of FIXM Core in a given context. It can be of global, regional or local applicability, depending on the context. An **Application Library** essentially provides context-specific **‘message data structures’** and **‘message templates’** which enables harmonized representation of the FIXM-based messages exchanged using SWIM information services, as outlined in the figure below.

Figure 1: General structure of a message and role of an Application Library

An **Application Library** captures messaging related data elements and reuses and restricts relevant subsets of the FIXM Core data structures. FIXM Core is independent and does not require an update when changes in an application library occur.

An **Application Library** may also leverage **Extensions**, as illustrated on the picture opposite.

An example of an Application Library is the **FF-ICE Application Library** developed and released by the FIXM CCB. This library addresses the use of FIXM core in the specific context of FF-ICE. It provides harmonized FF-ICE Message data structure (e.g. data structures for representing the FF-ICE Filing Status, the FF-ICE Planning Status etc.) and the FF-ICE message templates (e.g. the template for the FF-ICE Filed Flight Plan Message, the template for the FF-ICE Flight Cancellation Message etc.), in line with the FF-ICE Implementation Guidance Manual.

More details about this FF-ICE Application Library can be found in Chapter 3.2 .

#### Message Data Structures

Message Data structures designate at high level the data structures that are necessary for understanding the meaning and purpose of the information that is exchanged in a given context. They commonly include message identifiers and timestamps, codes identifying business types of messages, and any context-specific data that qualify the associated message interactions[2].

Examples of message data structures can be found in the FF-ICE Implementation Guidance Manual. The Figure below shows the message data structures associated with the FF-ICE Flight Cancellation Message.

Figure 2: Example of Message Data structures from FF-ICE

#### Message Templates

A message template is a more restrictive subset of message and flight data structures that is relevant to a given information exchange. In SWIM terms, a message template provides guidance for formatting a given information service payload.

By removing unused fields, adjusting multiplicities, and adding or further limiting pattern constraints, a template can tailor the broad standard represented by FIXM to reflect the content requirements of a particular message exchange. Templates offer message-specific guidance and validation rules while remaining entirely compliant with the broader FIXM structures.

A list of benefits for employing templates is detailed below.

|  |  |  |
| --- | --- | --- |
| **Benefit of templates** | **Without templates** | **With templates** |
| Reduced Development Overhead | Increased development overhead as each user must independently interpret how message content requirements should be represented in FIXM format. | Tailored schemas reduce development overhead by providing additional guidance for creating messages with a FIXM-based content. |
| Consistent Message Structure | Individual interpretations of requirements could lead to inconsistent message content implementation across users. | Making dedicated implementation templates available to all users should improve implementation consistency. |
| Improved XML Validation | XML-based validation limited to data syntax checking with no guidance for required vs. optional or allowed vs. not allowed content (failing to fully leverage a major benefit of using XML). | XML-based validation enforces both syntax and content completeness rules (fully leveraging benefits of XML-based validation). |

The use of message templates therefore improves interoperability, data quality, and ease and cost of development for any exchange they are applied to. They provide FIXM users with guidance and structure while at the same time allowing FIXM to remain open and flexible.

**XML representation of FIXM-based Message Templates**

The XML representation of FIXM-based Message templates is currently achieved by restricting complex types defined by FIXM. Restricting complex types is a standard-based approach for removing unwanted elements and/or attributes and to apply tighter restraints to multiplicities, patterns, and facets. Complex type restrictions also provide built-in validation: if the restriction is not correctly formed in relation to the parent type then the resulting schemas will not validate.

**Benefits of XSD restrictions**

* XSD restrictions are explicit: using an XSD schema with restrictions means using the rules of the base XSD schema plus additional rules that are explicitly declared;
* XSD restrictions provide some built-in validation for quality assurance
* XSD restrictions represent a natural use of the XSD standard;
* XSD restrictions deliver benefits in terms of model development and maintenance. [3](https://www.fixm.aero/documents/FIXM%20Strategy.pdf)

**Potential shortcomings of XSD restrictions**

The online literature about XML schema design generally considers that the restrictions of XSD complex types are the most difficult and therefore the least supported part of the XML schema specification. Implementers experiencing issues with the FIXM templates are invited to report their problems to the FIXM community, with details about the development environment being used. Alternatives to XSD restrictions may be then considered, as appropriate (see next section).

**XSD Profiles as a potential alternative to XSD Restrictions**

An XSD profile would represent a reduced, further restricted subset of the original model. This approach is very similar to using restrictions but accomplishes the task by directly creating smaller, parallel models of the adjusted packages rather than producing them via a restriction. The figure below illustrates at high-level the differences.

Figure 3: XSD Restriction vs XSD Profile

XSD profiles would not restrict the types from the base reference and would not bring any additional complexity. They could therefore be processed by marshalling tools in a smoother way compared to XSD restrictions.

XSD profiles may be therefore developed as an alternative to XSD restrictions for representing FIXM-based message templates.

#### How to build an application library?

APPENDIX B provides detailed guidance for creating application libraries.

### Extensions

#### What is it?

An extension designates a supplement to FIXM that supports additional (commonly local or regional) requirements from a particular organisation or community of interest. An extension may supplement FIXM Core by defining additional flight data structures exchanged locally or regionally, and/or may supplement an existing Application Library by defining additional messaging data structure exchanged locally or regionally.

#### What is a valid use of an extension?

A number of rules are established in order to ensure that extensions are not developed as a replacement of FIXM Core or a subset thereof.

The requirements on FIXM extensions are provided below. They are equally applicable to verified and non-verified extensions, but are enforceable only for verified extension. Non-verified extensions satisfying the requirements below will be recognised as a valid usage of the FIXM extension mechanism.

**Requirement on extension design**

|  |  |
| --- | --- |
| **Requirement** | To qualify for a valid FIXM extension, an extension shall be designed in accordance with the modelling principles described in APPENDIX A. |
| **Rationale** | The successful development of an extension, and its successful integration with the FIXM core packages, requires rules on extension design to be followed consistently by all implementers. |
| **How to check this** | Checking that an extension satisfies this requirement cannot be automated and requires manual analysis of the extension content by the FIXM community. As a general principle, extensions to FIXM core that are proposed for online publication on the FIXM web site should be checked against this requirement. |

**Requirement on extension content**

|  |  |
| --- | --- |
| **Requirement** | To qualify as a valid FIXM extension, an extension shall never contain a model element that would redefine, or supersede, a model element that is already defined in FIXM Core. |
| **Rationale** | FIXM core is an information exchange model capturing flight information that is globally harmonised. Redefining or superseding the FIXM core content in an extension would amount to diverging from this globally harmonised content and would go against the fundamental harmonisation objectives of FF-ICE and FIXM. |
| **How to check this** | Checking that an extension satisfies this requirement cannot be automated and requires manual analysis of the extension content by the FIXM community. As a general principle, extensions to FIXM core that are proposed for online publication on the FIXM web site should be checked against this requirement. |

Example of FIXM extension satisfying the requirement on extension content

Figure 4: Example of FIXM extension satisfying the requirement on extension content

This example is an extract from the US NAS extension to FIXM 4.1.0 [7]

* Copyright (c) US Federal Aviation Administration (FAA), available on [[www.FIXM.aero](http://www.FIXM.aero)](http://www.FIXM.aero). This extract features a class named “NasFlight” (in blue on the diagram) that extends the core model element “Extension” and which defines a content that supplements the core model element “Flight”. The content of class “NasFlight” does not replace or supersede any of the existing properties of the core “Flight” class. The example therefore qualifies as valid usage of the extension mechanism.

Example of FIXM extension NOT satisfying the requirement on extension content

Figure 5: Example of FIXM extension NOT satisfying the requirement on extension content

This example features a fictitious extension to FIXM Core which models one class entitled “WrongFlight” (in blue on the diagram). This class defines a property named “gufi” that is typed using CharacterString. The extension essentially redefines the “gufi” property from the FIXM core model element “Flight” and loosens its format, allowing any type of character string to be populated. This is an example of a FIXM extension redefining content from FIXM Core. It does NOT qualify as valid usage of the FIXM extension mechanism.

#### How to build an extension?

The FIXM extension mechanism distributes class-specific extension hooks throughout the model that implementers can leverage to define their specific data structures.

The key benefits of the approach are the following:

1. ability to allow Extension validation
2. multiple co-existing Extensions
3. co-location of Extension data with the Core data it extends
4. ability to easily remove extensions and pare down the model

This permissive approach enables FIXM users to enrich the core FIXM datasets with as many information elements as necessary, as required by the applicable implementation context.

APPENDIX A provides a rulebook and detailed guidance for creating extensions.

#### Ignoring extension data

Consumers of FIXM information may not need, and/or may not be able to process and interpret extension data supplementing a core FIXM dataset.

Using XSLTs is one approach for removing unwanted Extension data (known or unknown) from a FIXM XML dataset, as appropriate. An example of an XSLT that removes all Extension content is provided below:

<?xml version="1.0" encoding="UTF-8"?>  
<xsl:stylesheet version="2.0" xmlns:xsl="<http://www.w3.org/1999/XSL/Transform">>  
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>  
<xsl:template match="@\*|node()">  
<xsl:copy>  
<xsl:apply-templates select="@\*|node()"/>  
</xsl:copy>  
</xsl:template>  
<xsl:template match="\*:extension"/>  
</xsl:stylesheet>

## Tested Development Environments

*Ask FAA to share their findings with respect of which development environments work properly with FIXM 4.2.0*

## General encoding rules

### Date/Time Specification

FIXM requires times to be expressed in UTC.

A constraint is placed on class *Base.Types.Time*, the class used to represent all date/time values in FIXM, imposes the use of the trailing character ‘z’ to indicate UTC, in line with the W3C XSD specification.

Example: 20th July 1969 at 20:18UTC is expressed as  
1969-07-20T20:18:00.000Z

**Note to implementers**

The mapping of the XSD type dateTime to native structures in various development contexts is not always 1-1 and may exhibit a wide variety of difficulties depending on the tooling and runtime context. In particular, the trailing character ‘z’ indicating UTC may actually be stripped/omitted, leading to FIXM times being interpreted as local times instead of UTC times by some applications. FIXM implementers are therefore invited to crosscheck that their systems correctly interpret FIXM times as UTC time.

### Geographical positions

FIXM captures the concept of Geographical Position as defined by ICAO Annex 15.

*Position (geographical). Set of coordinates (latitude and longitude) referenced to the mathematical reference ellipsoid which define the position of a point on the surface of the Earth.*

This model element maps to the ISO 19107 “Point” construct, defined as a single location given by a direct position.

A geographic location consists of a co-ordinate reference system and geographic co-ordinates.

Co-ordinate reference system

ICAO Annex 11 chapter 2.29.1 states that World Geodetic System — 1984 (WGS-84) shall be used as the horizontal (geodetic) reference system for air navigation. The Coordinate Reference System reference is critical for the correct encoding and processing of FIXM positions\*. This is because a CRS not only indicates the geodetic datum and ellipsoid for which point coordinates are expressed but also the order of the coordinate axes in which coordinate values are provided, e.g. latitude before longitude – which is an important convention for the aviation domain.\* [copied from [OGC 12-028r1](https://portal.opengeospatial.org/files/?artifact_id=62061)] The EPSG:4326 CRS is the recommended choice for AIXM 5.1 data sets that use the WGS-84 reference datum.

FIXM implements a fixed co-ordinate reference system: “urn:ogc:def:crs:EPSG::4326”.

Geographic co-ordinates

The EPSG:4326 CRS has latitude as the primary axis, which indicates that **the order of the values in the fb:pos element is** **first latitude**, **second longitude**. This ordering convention is the one applied to the aviation domain.

The co-ordinates are represented in FIXM by a two-valued sequence[4](https://www.fixm.aero/releases/FIXM-4.1.0/FIXM_Core_v4_1_0_Modelling_Best_Practices.pdf), the first being the latitude and the second being the longitude, each of which is a floating point number (the decimal value in degrees). The direction is determined by the sign of the value, as specified in the next table.

|  |  |  |
| --- | --- | --- |
| Sign | Latitude | Longitude |
| Positive | N | E |
| Negative | S | W |

Note the latitude and longitude values are encoded as double in FIXM. Imposition of range restriction (-90≤latitude≤90, -180≤longitude≤180) does not appear in the model since different elements of the sequence of values have different constraints.

Examples

On EXAMPLE 1 above, number ‘59.0’ represents the latitude and number ‘-30.0’ represents the longitude.

Miscellaneous

The W3C XML schema 1.0 specification defines three special values for float/double: positive infinity, negative infinity and not-a-number. In this context, a “pos” element expressed as <fb:pos>INF -INF</fb:pos> or <fb:pos>NaN NaN</fb:pos> would be syntactically correct; it would validate against the core FIXM XML schemas. However, it would not represent any plausible location. The use of these special values is therefore not accepted when exchanging geographical positions in FIXM.

Why FIXM does not use the Geography Markup Language (GML)

FIXM does not adopt the GML standard for the representation of geospatial data. The reasons for not adopting GML are the following:

* Wherever a GML dependency is introduced, there would be a need for users to use the GML schemas and therefore to understand GML, which increases implementation costs.
* The Flight Planning community is not traditionally familiar with geospatial concepts. The introduction of GML would become an important drawback for FIXM adoption in support of FF-ICE/R1.
* Introducing a dependency on GML would make FIXM more difficult to implement particularly in certain environments. For instance, .NET technologies have been identified as *incompatible* with the GML usage.

### References to published aeronautical information

Describing the predicted movement of a flight commonly means indicating which parts of the infrastructure (ATS routes, waypoints, radio navigation aids etc.) are expected to be used by the flight. This is enabled in FIXM by a set of “references” constructs. These references are not flight-specific information; they pertain to the aeronautical information domain.

Different formats exist for exchanging aeronautical information, whose usages depend on specific implementation considerations and actual context within the overall data chain. For instance:

* AIXM is developed in order to enable the provision in digital format of the aeronautical information that is in the scope of Aeronautical Information Services (AIS). AIXM 5.1 covers both the content of Aeronautical Information Publications (AIP) and of the NOTAM information.
* The ARINC 424 standard defines a format for navigation (and communication) information, including but not limited to, aerodrome, runway, navaid, airway and terminal approach procedure information that is exchanged between data suppliers and avionics vendors.
* The Aerodrome Mapping Exchange Schema (AMXM XML Schema) is an exchange format specification for AMDB as standardized by and dedicated to the EUROCAE/RTCA Aeronautical Databases WG44/SC217. It is an XML Schema implementation of the DO-291C/ED-119C AMXM UML model.

**FIXM does not import any of these formats or profiles thereof.** FIXM defines its own structures for referring to aeronautical information that are **self-contained** but mappable to their AIXM/AMXM… equivalents thanks to the reuse of a common semantics. FIXM also provides an **optional** mechanism enabling these self-contained references to be supplemented with, but not replaced by, **hypertext** **references** to AIXM features.

The following sections provide guidance for correctly encoding these references in FIXM.

#### Generic hypertext references

If an AIXM 5.1 feature exists that corresponds to the element being referred to, an **optional** hypertext reference to that AIXM feature may be provided. This reference shall be expressed in accordance with chapter 3.4.1 (*Abstract references using UUID*) of the [AIXM feature Identification and Reference](http://www.aixm.aero/sites/aixm.aero/files/imce/AIXM51/aixm_feature_identification_and_reference-1.0.pdf) document developed by the AIXM community.

This hypertext reference may be used - or ignored - by the receiving system depending on its capabilities.

Example:

<fx:*[FIXMelement]* href="urn:uuid:81e47548-9f00-4970-b641-8ff8f99098a5">

Important note: FIXM does not import the W3C XML Linking Language (xlink) v1.1 schemas in order to represent the hypertext references. FIXM mimics the xlink Locator attribute named “href” but defines it within the FIXM (Base) namespace[5](http://www.FIXM.aero).

#### References to Waypoints

OPTION 1 - Minimum reference

As a minimum, the coded designator of waypoint shall be provided, as published in the AIPs.

OPTION 2 - Unambiguous reference

*(OPTION 2 = OPTION 1 + supplementary position information)*

The coded designator of a waypoint is not always sufficient for unambiguously referring to that element. The 5-letter coded designator of a waypoint is supposed to be unique world-wide (according to ICAO Annex 11) but is not in reality. There are at least 5% duplicates/triplicates/even more…

FIXM adds an optional property 'position' which may be used as a complement to the 'designator' information in order to remove any ambiguity on the designator.

Important note: the combinations of fields [designator + position] shall not be interpreted as a natural key uniquely identifying the waypoint, in so far as producing and consuming systems/services may use different aeronautical information sources with different degree of precisions for lat/long, leading to small variations of the position information. The supplementary fields shall be used for disambiguation purposes only (i.e. plausibility checks based on position) in case of duplicate/triplicate/…

OPTION 3 - Minimum reference with supplementary AIXM pointer

*(OPTION 3 = OPTION 1 + supplementary hypertext reference)*

Option 3 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

OPTION 4 - Complete reference

*(OPTION 4 = OPTION 2 + OPTION 3)*

Option 4 corresponds to the combination of Option 2 and Option 3. See explanations above.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM references to fictitious Waypoint “TEMPO” that is ‘published’ in AIXM 5.1 as part of the fictitious [Donlon dataset](https://github.com/aixm/donlon/blob/master/Donlon.xml). The data is entirely fictitious, located somewhere in the middle of the Atlantic Ocean. The examples shall NEVER BE USED AS OPERATIONAL DATA.

Note about the pattern constraint for waypoint designators

FIXM supports waypoint designators of 1 to 5 characters. This design is intentional. In most cases, waypoints have a 5-letter designator, as prescribed by ICAO. However, these designators might be shorter in some particular cases. Examples of shorter waypoint designators could be found in various sources:

* In chapter 7 of the ARINC 424 specification. This chapter provides rules for forming the designators of waypoints in the absence of published designators. These rules can lead to designators of less than 5 characters. The following extracts from ARINC 424-19 provide some examples, which are highlighted in blue.
* In the [US FAA’s publication of Instrument Flight Procedures encodings in ARINC 424-18 format](https://www.faa.gov/air_traffic/flight_info/aeronav/digital_products/cifp/download/). For instance:
  + Localizer only approach on RWY06 at KFOD

SUSAP KFODK3FL06 AFOD 010FOD K3D 0V IF 18000 0 NS 854081209

SUSAP KFODK3FL06 AFOD 020FF06 K3PC0E TF + 02800 0 NS 854091209

SUSAP KFODK3FL06 AFOD 030FF06 K3PC0EE AR PI IFODK3 2430006119800100PI

* 02800 0 NS 854101209

SUSAP KFODK3FL06 AHIMSU 010HIMSUK3PC0E A IF FOD K3 15000140 D 18000 0 NS 854111310

SUSAP KFODK3FL06 AHIMSU 020SHRLAK3PC0EE BR AF FOD K3 216501401500 D + 02800 0 NS 854121310

SUSAP KFODK3FL06 ALESFI 010LESFIK3PC0E A IF FOD K3 25400140 D 18000 0 NS 854131310

SUSAP KFODK3FL06 ALESFI 020SHRLAK3PC0EE BL AF FOD K3 216501402540 D + 02800 0 NS 854141310

SUSAP KFODK3FL06 L 010SHRLAK3PC0E I IF IFODK3 24300163 PI + 02800 18000 0 NS 854151310

SUSAP KFODK3FL06 L 020FF06 K3PC0E F CF IFODK3 2430006106300101PI + 02800 FO K3PN0 NS 854161209

SUSAP KFODK3FL06 L 021ATLOJK3PC0E S CF IFODK3 2430003406300028PI + 01820 -324 0 NS 854171310

SUSAP KFODK3FL06 L 030RW06 K3PG0GY M CF IFODK3 2430001306300021PI 01134 -324 0 NS 854181209

SUSAP KFODK3FL06 L 040 0 M CA 0630 + 02800 0 NS 854191209

SUSAP KFODK3FL06 L 050FOD K3D 0VY L DF + 02800 0 NS 854201209

SUSAP KFODK3FL06 L 060FOD K3D 0VE R HM 1204T010 + 02800 0 NS 854211209

* GPS approach on RWY29 at KFOT

SUSAP KFOTK2FP29 ADINSE 010DINSEK2EA0E A IF 18000 P PS 857251211

SUSAP KFOTK2FP29 ADINSE 020JEWPEK2PC0E TF + 06000 P PS 857261310

SUSAP KFOTK2FP29 ADINSE 030SPHERK2PC0EE TF + 04700 P PS 857271310

SUSAP KFOTK2FP29 AKNEES 010KNEESK2EA0E IF 18000 P PS 857281211

SUSAP KFOTK2FP29 AKNEES 020YAGERK2EA0E A TF + 06800 P PS 857291211

SUSAP KFOTK2FP29 AKNEES 030SPHERK2PC0EE TF + 04700 P PS 857301310

SUSAP KFOTK2FP29 APLYAT 010PLYATK2EA0E A IF 18000 P PS 857311211

SUSAP KFOTK2FP29 APLYAT 020JEVGYK2PC0E TF + 06000 P PS 857321310

SUSAP KFOTK2FP29 APLYAT 030SPHERK2PC0EE TF + 04700 P PS 857331310

SUSAP KFOTK2FP29 P 010SPHERK2PC0E I IF + 04700 18000 P PS 857341310

SUSAP KFOTK2FP29 P 011JEVSYK2PC0E S TF + 03700 P PS 857351310

SUSAP KFOTK2FP29 P 020ELLYSK2PC0E F TF + 02700 IZPUH K2PCP PS 857361310

SUSAP KFOTK2FP29 P 021SP29 K2PC0E A TF + 01840 -356 P PS 857371211

SUSAP KFOTK2FP29 P 030IZPUHK2PC0EY M TF 00617 -356 P PS 857381310

SUSAP KFOTK2FP29 P 040 0 M CA 2757 + 01500 P PS 857391211

SUSAP KFOTK2FP29 P 050FOT K2D 0VY R DF + 03000 P PS 857401211

SUSAP KFOTK2FP29 P 060FOT K2D 0VE R HM 1610T010 + 03000 P PS 857411510

* In the European AIS Database (EAD). A query on the EAD helped identify the following:
  + 40 occurrences of designated points with a 1-letter designator
  + 190 occurrences of designated points with a 2-letters designator
  + 356 occurrences of designated points with a 3-letters designator
  + 3046 occurrences of designated points with a 4-letters designator

Most of the designated points with 2, 3 or 4-letter designators actually correspond to designated points formed after the position of a navaid or an aerodrome. However, some occurrences cannot be related to these cases. The following table quotes a few random examples.

|  |  |  |  |
| --- | --- | --- | --- |
| **Designator** | **Lat** | **Long** | **Type** |
| *1-letter designator* |  |  |  |
| N | 503241N | 0042718E | ADHP |
| E | 502941N | 0044206E | ADHP |
| A | 475826.0000N | 0161445.0000E | OTHER |
| B | 475202.0000N | 0161514.0000E | OTHER |
| B | 475734.0000N | 0161614.0000E | OTHER |
| *2-letter designator* |  |  |  |
| S1 | 460217N | 0142702E | OTHER |
| S2 | 460552N | 0142748E | OTHER |
| S3 | 460845N | 0142452E | OTHER |
| W1 | 461846N | 0141449E | OTHER |
| W2 | 461744N | 0142049E | OTHER |
| A1 | 453243N | 0181709E | OTHER |
| A2 | 423842N | 0175651E | ADHP |
| A3 | 434049N | 0155502E | ADHP |
| *3-letter designator* |  |  |  |
| PJ1 | 544005N | 0253057E | OTHER |
| PJ2 | 554243N | 0211434E | OTHER |
| PJ3 | 561350N | 0221534E | OTHER |
| TP1 | 481018.5959N | 0113540.1135E | ADHP |
| TP2 | 481314.6479N | 0113003.5398E | ADHP |
| 13A | 512900.0000N | 0185100.0000E | OTHER |
| 17A | 520800.0000N | 0174500.0000E | OTHER |
| 20A | 514400.0000N | 0160800.0000E | OTHER |
| 21A | 521300.0000N | 0162800.0000E | OTHER |
| 27A | 523500.0000N | 0160500.0000E | OTHER |
| ME1 | 462338N | 0155433E | OTHER |
| ME2 | 463440N | 0155009E | OTHER |
| DX1 | 481323.7979N | 0122604.5443E | ADHP |
| DX1 | 505951.6149N | 0141519.3372E | ADHP |
| DX2 | 505625.7259N | 0141418.8105E | ADHP |
| *4-letter designator* |  |  |  |
| SJ91 | 000224S | 1044205E | OTHER |
| FF36 | 375713.7253S | 1751802.2481E | ADHP |
| SM01 | 464002.55N | 0170535.52E | ADHP |
| SM02 | 464750.28N | 0165926.87E | ADHP |
| LDD5 | 514138.7284N | 0191615.9435E | ADHP |
| PG23 | 545132N | 0230742E | OTHER |
| IF09 | 431021.1N | 0252819.1E | ADHP |
| MAPT | 492818.3679N | 0083229.2734E | ADHP |
| SDF2 | 495218.7277N | 0112216.9347E | ADHP |
| FF16 | 513503.7055N | 1124320.1363W | ADHP |
| ECHO | 504818N | 0051950E | ADHP |
| ECHO | 482129.75N | 0703422.93W | OTHER |

#### References to Navaid

OPTION 1 - Minimum reference

As a minimum, the coded designator of a navaid shall be provided, as published in the AIPs.

OPTION 2 - Unambiguous reference

*(OPTION 2 = OPTION 1 + supplementary position & navaidServicetype information)*

The coded designator of a navaid is not always sufficient for unambiguously referring to that element. The en-route navaids (VOR, DME, NDB) designator is supposed to be unique (according to ICAO Annex 11) within 600 NM. This means that these designators are not unique world-wide. For airport navaids, there is no limitation.

FIXM adds two optional properties 'position' and 'navaidServiceType' which may be used as a complement to the 'designator' information in order to remove any ambiguity on the designator.

Important note: the combination of fields [designator + position + navaid service type] shall not be interpreted as a natural key uniquely identifying the navaid, in so far as producing and consuming systems/services may use different aeronautical information sources with different degree of precisions for lat/long, leading to small variations of the position information. The supplementary fields shall be used for disambiguation purposes only (i.e. plausibility checks based on position and navaid service type) in case of duplicate/triplicate/…

OPTION 3 - Minimum reference with supplementary AIXM pointer

*(OPTION 3 = OPTION 1 + supplementary hypertext reference)*

Option 3 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

OPTION 4 - Complete reference

*(OPTION 4 = OPTION 2 + OPTION 3)*

Option 4 corresponds to the combination of Option 2 and Option 3. See explanations above.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM references to fictitious VOR DME “BOR” that is ‘published’ in AIXM 5.1 as part of the fictitious [Donlon dataset](https://github.com/aixm/donlon/blob/master/Donlon.xml). The data is entirely fictitious, located somewhere in the middle of the Atlantic Ocean. The examples shall NEVER BE USED AS OPERATIONAL DATA.

#### References to Aerodromes

OPTION 1 - Minimum reference

The minimum aerodrome reference shall consist of the aerodrome location indicator, if provided by ICAO Doc 7910 [11]. If the aerodrome has no ICAO Doc 7910 location indicator, the minimum aerodrome reference shall consist of the name of the aerodrome and its geographical location, namely the aerodrome reference point.

OPTION 2 - Minimum reference with supplementary AIXM pointer

*(OPTION 2 = OPTION 1 + supplementary hypertext reference)*

Option 2 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM references to fictitious aerodrome “DONLON” that is ‘published’ in AIXM 5.1 as part of the fictitious [Donlon dataset](https://github.com/aixm/donlon/blob/master/Donlon.xml). The data is entirely fictitious, located somewhere in the middle of the Atlantic Ocean. The examples shall NEVER BE USED AS OPERATIONAL DATA.

Important note: FIXM enables the encoding of richer aerodrome “reference” structures, such as

<fx:destinationAerodrome href="urn:uuid:1b54b2d6-a5ff-4e57-94c2-f4047a381c64">

<fb:iataDesignator>DLN</fb:iataDesignator>

<fb:locationIndicator>EADD</fb:locationIndicator>

<fb:name>DONLON</fb:name>

<fb:referencePoint srsName="urn:ogc:def:crs:EPSG::4326">

<fb:pos>52.3716666666667 -31.9494444444444</fb:pos>

</fb:referencePoint>

</fx:destinationAerodrome>

The provision of the name and reference point in addition to the location indicator is technically possible but does not serve the purpose of identifying the aerodrome. This implementation practice only aims to provide consumers with richer information about the aerodrome being referred to. Whether or not to use this supplementary information is at the discretion of the consuming system / service.

#### References to Runway Directions

OPTION 1 - Minimum reference

The minimum Runway Direction reference shall consist of the Runway Direction designator.

OPTION 2 - Minimum reference with supplementary AIXM pointer

*(OPTION 2 = OPTION 1 + supplementary hypertext reference)*

Option 2 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM references to fictitious Runway Direction “09L” that is ‘published’ in AIXM 5.1 as part of the fictitious [Donlon dataset](https://github.com/aixm/donlon/blob/master/Donlon.xml). The data is entirely fictitious, located somewhere in the middle of the Atlantic Ocean. The examples shall NEVER BE USED AS OPERATIONAL DATA.

#### References to Enroute ATS routes

OPTION 1 - Minimum reference

The minimum Enroute ATS Route reference shall consist of the Enroute ATS Route designator as published in the AIP. Enroute ATS Route designators are not unique. However, the systematic pairing of an ATS route designator with a route point (i.e. a significant point belonging to that ATS route) in FIXM is considered sufficient for enabling unambiguous identification of the ATS Route being referred to.

OPTION 2 - Minimum reference with supplementary AIXM pointer

*(OPTION 2 = OPTION 1 + supplementary hypertext reference)*

Option 2 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM references to fictitious Enroute ATS Route “UA4” that is ‘published’ in AIXM 5.1 as part of the fictitious [Donlon dataset](https://github.com/aixm/donlon/blob/master/Donlon.xml). The data is entirely fictitious, located somewhere in the middle of the Atlantic Ocean. The examples shall NEVER BE USED AS OPERATIONAL DATA.

#### References to SIDs and STARs

OPTION 1 - Minimum reference

The minimum SID or STAR reference shall consist of the SID or STAR designator as published in the AIP.

OPTION 2 - Minimum reference with supplementary AIXM pointer

*(OPTION 2 = OPTION 1 + supplementary hypertext reference)*

Option 2 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM references to SID “AMOLO 5B” that is published in the French AIPs.

FIXM also supports the supplementary provision of the abbreviated designator of the SID or the STAR which is commonly used in FMS databases and in some ground automation systems. The ‘abbreviated designator’, if provided, should be the designator obtained after applying the rules for shortening names specified by the ARINC 424 specification, chapter 7.4. Example:

<fx:standardInstrumentDeparture>

<fb:abbreviatedDesignator>AMOL5B</fb:abbreviatedDesignator>

<fb:designator>AMOLO5B</fb:designator>

</fx:standardInstrumentDeparture>

#### References to Airspace

OPTION 1 - Minimum reference

The minimum airspace reference shall consist of the airspace location indicator, if provided by ICAO Doc 7910 [11]. If the airspace has no ICAO Doc 7910 location indicator, the minimum airspace reference shall consist of the coded designator of the airspace as published in the AIP.

OPTION 2 - Minimum reference with supplementary AIXM pointer

*(OPTION 2 = OPTION 1 + supplementary hypertext reference)*

Option 2 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

Examples (NOT for OPERATIONAL USE)

#### References to (ATC) Units

OPTION 1 - Minimum reference

The minimum ATC unit reference shall consist of the location indicator of the unit, if provided by ICAO Doc 7910 [11]. If the unit has no ICAO Doc 7910 location indicator, the minimum airspace reference shall consist of the name of the unit or any alternate name, as published in the AIP.

OPTION 2 - Minimum reference with supplementary AIXM pointer

*(OPTION 2 = OPTION 1 + supplementary hypertext reference)*

Option 2 corresponds to Option 1 with an additional hypertext reference as described in chapter Generic hypertext references.

Examples (NOT for OPERATIONAL USE)

### Relative points

A relative point is a bearing and distance from a reference navaid. Encoding a relative point in FIXM requires the ‘bearing’, ‘distance’ and ‘referencePoint’ properties to be provided. All of these properties shall be provided.

FIXM enables a relative point to be supplemented with an optional ‘position’ value for storing the actual position of the relative point if already known. The exchange of this information may prove useful in order to save consuming systems / services from (re)computing the position of the relative point. Whether or not to use this supplementary information is at the discretion of the consuming system / service.

Examples (NOT for OPERATIONAL USE)

The table below depicts examples of FIXM encodings of relative points that are derived from the fictitious [Donlon dataset](https://github.com/aixm/donlon/blob/master/Donlon.xml). The data is entirely fictitious, located somewhere in the middle of the Atlantic Ocean. The examples shall NEVER BE USED AS OPERATIONAL DATA.

### Vertical distances

**Definitions**

The term vertical distance collectively refers to altitudes, elevations and heights, as defined by ICAO

Figure 6: Differences between Elevation, Altitude, Height and Ellipsoid height

* **Altitude** = The vertical distance of a level, a point or an object considered as a point, measured from mean sea level (MSL). **[ICAO]**
* **Elevation** = The vertical distance of a point or a level, on or affixed to the surface of the earth, measured from mean sea level. **[ICAO]**
* **Height** = The vertical distance of a level, a point or an object considered as a point, measured from a specified datum. **[ICAO]**
* **Ellipsoid height** = The height related to the reference ellipsoid, measured along the ellipsoidal outer normal through the point in question. **[ICAO]**

**FIXM representation of vertical distances**

FIXM supports the representation of altitudes expressed in feet or meters (FIXM construct ‘Altitude’), of altitudes expressed as flight level number or standard metric level (FIXM construct ‘FlightLevel’) and of ellipsoid heights & heights SFC expressed in feet or meter (FIXM construct ‘Height’ used in conjunction with a VerticalReference).

These vertical distances are specialisations of the generic class Measure which serves as the parent class for all measure types including speeds, angles, pressures, temperatures etc. Therefore, altitudes, flight levels and heights are always encoded as double values, although integer values are expected. “Double Integer” conversion can be handled differently depending on the technical context. This may lead to e.g. flight level value 100 being expressed as 100.00…0001 and the flight level value 101 being expressed as 100.99…99999. It is acknowledged that the current FIXM design may create value persistence problem across applications, in particular if rounding or truncation are applied further down. FIXM implementers are therefore invited to verify the persistence of vertical distances values across their software.

**Examples**

The following examples show valid FIXM encoding of altitudes and flight levels expressed as integer.

<fb:altitude uom="FT">10000</fb:altitude>

<fb:altitude uom="M">3500</fb:altitude>

<fb:flightLevel uom="FL">290</fb:flightLevel>

<fb:flightLevel uom="SM">1130</fb:flightLevel>

The following example shows the encoding of a flight level expressed as a double. This encoding is technically permitted by FIXM but is NOT recommended.

<fb:flightLevel uom="FL">290.0</fb:flightLevel>

### Sequence numbers

The FIXM Logical Model specifies several ordered repeating sequences. The FIXM XML schemas add an optional sequence number attribute to the repeating elements in order to ensure that the order of a sequence is always preserved, even after XSLT manipulation.

The sequence number should be a sequentially increasing integer with a value beginning at zero. These sequence numbers are only meant for ordering, not identification, purposes. As such, the set of sequence numbers taken as a whole should always be contiguous. If an element were removed from a sequence, the numbering in subsequent representations should be reset to reflect this, not maintained so that a gap is formed.

Example

### Contact Information

Postal Address

TODO

Telephone Contact

TODO

Online Contact

In FIXM, the online contact information can include an email address and/or a network address.

A network address is always formed of two pieces of information: the **linkage** and the **network** information.

* The former captures the expression of the network address. This is supported by property OnlineContact.linkage.
* The latter captures the network on which the address is valid. This is supported by property OnlineContact.network.

Property OnlineContact.network provides a choice between predefined network types and free text. Network information should be preferably encoded using the property NetworkChoice.**type** populated with the applicable enumerated value from enumeration TelecomNetworkType. If none of the enumerated values is suitable, the property NetworkChoice.other shall be used. The ATM Information Reference Model provides additional telecom network types that should be used for populating NetworkChoice.other, as appropriate. These additional AIRM values are available at the following link:

<http://airm.aero/viewer/1.0.0/logical-model.html#CodeTelecomNetworkType>

The type of network affects the format of the linkage information.

* When the network is INTERNET, the linkage should be a resolvable URL
* When the network is AFTN, the linkage should be a valid AFTN address

Examples

The following example illustrates contact information formed of an email address and an Internet address expressed as a resolvable URL.

<fb:onlineContact>

<fb:email>[fixm.secretariat@eurocontrol.int](mailto:fixm.secretariat@eurocontrol.int)</fb:email>

<fb:linkage><https://www.fixm.aero/content/contact.pl&lt;/fb:linkage>>

<fb:network>

<fb:type>INTERNET</fb:type>

</fb:network>

</fb:onlineContact>

The following example illustrates contact information formed of an AFTN address. The example features the EUROCONTROL NM ‘AFTN address’ for Flight Plan Message Submission to IFPS (FP1 - Brussels (Haren)).

<fb:onlineContact>

<fb:linkage>EUCHZMFP</fb:linkage>

<fb:network>

<fb:type>AFTN</fb:type>

</fb:network>

</fb:onlineContact>

The following example illustrates contact information formed of a SITA address. ‘SITA’ is not a value captured in FIXM enumeration TelecomNetworkType but is part of the reference ATM vocabulary provided by the AIRM ([AIRM codelist CodeTelecomNetworkType](http://airm.aero/viewer/1.0.0/logical-model.html#CodeTelecomNetworkType)). The example features the EUROCONTROL NM ‘SITA address’ for Flight Plan Message Submission to IFPS (FP1 - Brussels (Haren)).

<fb:onlineContact>

<fb:linkage>BRUEP7X</fb:linkage>

<fb:network>

<fb:other>SITA</fb:other>

</fb:network>

</fb:onlineContact>

### Version numbers

*This is a placeholder for further guidance on how to encode version numbers. This chapter will be populated in a future version of the document.*

### Aircraft Types

*This is a placeholder for further guidance on how to encode aircraft types. This chapter will be populated in a future version of the document.*

### Rules for encoding a 4D Trajectory

*This is a placeholder for further guidance on how to encode a 4D trajectory. This chapter will be populated in a future version of the document.*

### Rules for encoding Constraints

*This is a placeholder for further guidance on how to encode constraints. This chapter will be populated in a future version of the document.*

### General rules for data correctness

*Important note: in the present version of the document, limited effort could be spent on the documentation of FIXM business rules addressing data correctness. The list below provides an initial set of rules that were identified during the writing of this document or that are already captured in the FIXM model as model element notes. Future versions of the document will enrich this table based on implementers’ feedback, and may also revisit the overall formulation and description method for these rules, in particular in the light of the* [*AIXM experience with regards to business rules*](http://aixm.aero/page/business-rules)*.*

### Rules for absent data

FIXM supports the representation of fields that are explicitly absent or that are deleted. It does so by leveraging the XSD specification for Elements which includes the *nillable* attribute. This “nillable” attribute specifies whether an explicit null value can be assigned to the element. When nillable is set to “true” in the element definition, this in turn enables an instance of the element to have the built-in nil attribute with a value set to “true”. Example:

<xs:complexType name="FlightType">  
[...]  
<xs:element name="dangerousGoods" type="fx:DangerousGoodsType" **nillable="true"** [...]>  
[...]  
</xs:complexType>

<fx:Flight>  
<fx:dangerousGoods **xsi:nil="true"**/>  
</fx:Flight>

FIXM does not support the exchange of a “nil reason” to explain why an element is nil. The interpretation of a nil element therefore depends on the context of the information exchange:

* A nil element included in an FF-ICE Flight Plan Update message will indicate that this flight plan data item is to be deleted. This interpretation is dictated by the FF-ICE Implementation Guidance Manual which states the following:

*7.4.3.6 A Flight Plan Update is only required to contain those items that have changed (in addition to the mandatory items specified for an Update message), i.e. it is not necessary to resend complete flight data. Data items that were included in the previous version of the flight plan and have not been included in the Flight Plan Update will remain unchanged. This means that a mechanism is required to identify when a flight plan data item is to be deleted.”*

* A nil element included in an FF-ICE Flight Data Response Message will indicate that the data item is explicitly declared as not available to the flight data requestor.

Future FIXM versions may support the exchange of an additional “nil reason” attribute, if the need for it is identified by the FIXM Community.

*Note: the support for nillable elements has implied a significant design change in FIXM Core 4.2.0. The previous FIXM Core versions relied extensively on XSD attributes, which are not nillable. These XSD attributes were converted to XSD elements in FIXM Core 4.2.0 so that the built-in XSD attribute nillable could be leveraged.*

##### Declaring null Measures and Geographical Position

The FIXM Measures types enforce the provision of the “uom” attribute together with the numeric value of the measure. Likewise, the FIXM Geographical Position type enforces the provision of the srsName attribute “urn:ogc:def:crs:EPSG::4326” together with the position. This design guarantees that the required unit of measure and coordinate reference system are always provided in order to enable the correct interpretation of measures and positions. However, it requires a special workaround when null values are to be exchanged.

The provision of a null value for a measure or a position still requires the mandatory attribute “uom” or “srsName” to be provided, even if meaningless. For instance, the following XML data would NOT validate against the FIXM Core schema, because the uom for a Mass is missing.

<fx:desired>

<fx:takeoffMass xsi:nil="true/">

</fx:desired>

Therefore, the following rules apply when declaring null Measures or Geographical Position.

When a measure is to be declared null,

* Information provider side: Provide a fake uom in order to ensure proper schema validation
* Information consumer side: Ignore the fake uom provided together with the null measure

When a geographical position is to be declared null:

* Information provider side: Provide the srsName “urn:ogc:def:crs:EPSG::4326” in order to ensure proper schema validation
* Information consumer side: Ignore the provided srsName provided together with the null position.

Example of valid null measure declaration with a fake uom to be ignored:

<fx:desired>

<fx:takeoffMass xsi:nil="true" uom="KG"/>

</fx:desired>

## Other Topics

### The use of other exchange models

*An information exchange model is designed to enable the sharing of information in a digital format within a specific domain*[6](https://ost.eurocontrol.int/sites/FIXM/SitePages/Home.aspx) (e.g. AIXM for the aeronautical information domain or FIXM for the flight information domain). However, some ATM operations may require ATM information to be treated and exchanged in a more interrelated way. For example, a single traffic flow management message may include both airspace geometries (aeronautical information domain) as well as information about the flights passing through them (flight information domain).

Satisfying these cross-cutting information needs can be done in different manners. Combining data from existing information exchange models (e.g. AIXM and FIXM) is one approach. This document briefly touches on two possible options for doing so in an attempt to aid FIXM users wishing to create multi-model data exchanges.

#### Correlation references

The first option to consider is breaking down a multi-model data transmission into separate messages for each involved exchange model. These messages are supplemented with correlation references to all other component messages of the overall multi-model transmission. Supplying a list of unique message identifiers for each entry in the message group should be sufficient. This lets an end user know that such a message should not be processed alone and which other messages are intended to accompany it. Message identifiers and references can likely be handled by the data exchange’s messaging layer without the need to modify the exchange models themselves to accommodate this approach. However, the end user must perform the correlation work themselves. This would include creating procedures for how to handle a situation where only a subset of a message group is received.

#### Correlation models

The second option to consider is creating a new model that provides direct references within itself to the required exchange models – thus allowing multiple models to be used together in a single data transmission. This approach frees end users from the burden of correlating messages (and the associated dangers of partial data loss). However, this approach cannot be as easily leveraged by systems already capable of handling the underlying exchange models used. The new model must first be created and incorporated into the systems that transmit and receive this data.

# Using FIXM in Support of FF-ICE

## Target audience

This chapter provides specific guidance in support of the implementation of FF-ICE using FIXM. It therefore targets FF-ICE Implementers.

## The FF-ICE Application Library for FIXM

The FF-ICE Application Library is an Application Library[7](https://www.fixm.aero/fixm_nas_extension_420.pl) for FIXM that addresses the specific use of FIXM Core in the context of ICAO FF-ICE. It provides harmonized FF-ICE Message data structures and the individual FF-ICE Message templates in line with the requirements on FF-ICE Messages defined by the ICAO FF-ICE Implementation Guidance Manual (ICAO Doc 9965 Volume II).

The content of the FF-ICE Application Library is the following:

Figure 7: Overview of the FF-ICE Message Application Library content

The FF-ICE Message Application Library is developed and published by the FIXM CCB, together with FIXM Core.

### FF-ICE Message data structures

The FF-ICE message data structures are the data elements that specifically qualify the FF-ICE Messages. They do not describe a Flight but are necessary for understanding the purpose and meaning of an FF-ICE information exchange. The FF-ICE message data modelled by the FF-ICE Application Library include:

* A model element representing generically an FF-ICE Message with its identifier, timestamp, type etc. An enumeration provides the possible types of FF-ICE Messages: Filed Flight Plan message, Submission Response message, Filing Status message etc.
* Model elements representing the different FF-ICE statuses with their possible values: Planning statuses CONCUR / NON\_CONCUR / NEGOTIATE, Filing statuses ACCEPTABLE / NOT\_ACCEPTABLE, Submission statuses ACK / MANUAL / REJECT etc.
* Model elements representing the FF-ICE participants and their properties, which are used for identifying the operational stakeholders sending and receiving FF-ICE messages, or the list of relevant ASPs etc.

The FF-ICE Message data structures are traceable to the FF-ICE Implementation Guidance Manual Appendix B. For instance:

Figure 8: Example of FF-ICE Message data structures tracing to the FF-ICE Implementation Guidance Manual, Appendix B

The FF-ICE message data structures other than Choices and Codelists are extendable. This enables implementers to accommodate additional FF-ICE message data structures required locally or regionally, in support of local or regional FF-ICE requirements. Extension hooks are defined in a similar fashion as for FIXM Core data structures.

The picture below provides an overview of the FF-ICE Message data structures modelled in the FF-ICE Application Library.

Figure 9: Overview of the FF-ICE Message Data Structures

## FF-ICE Message Templates

### Overview

The FF-ICE Message templates are the representations of the individual FF-ICE messages that are exchanged by the FF-ICE Services. Thirteen message templates are defined in the FF-ICE Application library v1.0.0. They correspond to the thirteen FF-ICE Messages described in the FF-ICE/R1 Implementation Guidance Manual, Appendix C. The following table provides the correspondence between the FF-ICE message templates from the library and their corresponding description in the FF-ICE Implementation Guidance Manual, Appendix C.

Table 1: Correspondences between FF-ICE Message templates and their ICAO Doc 9965 Volume II description

|  |  |
| --- | --- |
| **FF-ICE Message templates** | **Associated Requirements from the FF-ICE Implementation Guidance Manual, Appendix C** |
| FiledFlightPlan | C-4 Filed Flight Plan |
| FilingStatus | C-5 Filing Status |
| FlightArrival | C-13 Flight Arrival |
| FlightCancellation | C-8 Flight Cancellation |
| FlightDataRequest | C-10 Flight Data Request |
| FlightDataResponse | C-11 Flight Data Response |
| FlightDeparture | C-12 Flight Departure |
| FlightPlanUpdate | C-9 Flight Plan Update |
| PlanningStatus | C-3 Planning Status |
| PreliminaryFlightPlan | C-2 Preliminary Flight Plan |
| SubmissionResponse | C-1 Submission Response |
| TrialRequest | C-6 Trial Request |
| TrialResponse | C-7 Trial Response |

The FF-ICE Message templates define concretely the restricted subsets of the FF-ICE Message data elements of the FIXM Core flight elements that are relevant for each FF-ICE message transaction. They explicitly declare which elements are mandatory, optional or irrelevant in each case, and enforce stricter content patterns as appropriate.

### Example of the FF-ICE ‘Flight Cancellation’ Message

The following table is a simplified version of table C-8 from the FF-ICE Implementation Guidance Manual, Appendix C. It describes the content of the FF-ICE Flight Cancellation Message and indicates which fields are mandatory (highlighted **in bold** in this document) or optional (highlighted in *in italic*).

Table 2: Example of the FF-ICE Flight Cancellation Message

|  |  |  |
| --- | --- | --- |
| **Data Category** | **Data Item** | **Requirement** |
| Message Information | **List of Recipients** | **Mandatory** |
|  | **Message Originator** | **Mandatory** |
|  | *Request for Translation and Forwarding* | *Optional* |
|  | *Requested Recipients* | *Optional* |
|  | **Message Date-Time** | **Mandatory** |
|  | **Message Identifier** | **Mandatory** |
|  | **Type of Request/Response** | **Mandatory** |
|  | *AFTN Address* | *Optional* |
|  | *Contact Information* | *Optional* |
| Flight Identification | **GUFI** | **Mandatory** |
|  | **GUFI Originator** | **Mandatory** |
|  | **Aircraft Identification** | **Mandatory** |
| Departure/Destination Data | **Departure Aerodrome** | **Mandatory** |
|  | **Destination Aerodrome** | **Mandatory** |
|  | **Estimated Off-Block Time** | **Mandatory** |

The FF-ICE Application library translates this table into an implementable message template. This is illustrated by the picture below. The message template resulting from the translation of this table is displayed with a blue background.

Figure 10: The FF-ICE Flight Cancellation Message Template

**Explanations**

**XSD complex type restrictions** are used to pare down the Flight and the FF-ICE Message data structures to just those fields that are applicable to the Flight Cancellation Message, as well as to enforce stricter optionality and content patterns where appropriate.

The XSD complex type restrictions are implemented by creating a new class that generalizes the class to be restricted and then applying the <<XSDrestriction>> stereotype to the generalization connector, as shown in brown on the picture opposite.

XML elements being irrelevant in the context of the message template are eliminated by removing them from the model, as shown in red on the picture opposite. Elements being mandatory in the context of the message template have their cardinality set to (at least) 1 in the restriction, as shown in blue.

In general, all optionality, cardinality, and pattern restrictions are implemented by applying the desired changes to the restricted class.

Because XSD complex type restrictions must use the same namespace as the types they restrict, it is necessary to change their names. The convention used in the FF-ICE Application Library is to prepend each restricted class with “Ffice” plus an initialism of the message being modeled – hence **FficeFC** for the FF-ICE \*\*F\*\*light \*\*C\*\*ancellation Message.

Restricted classes require the restricted versions of associated sub-classes. XSD complex type restrictions are therefore linked together to form an entire restricted message. This provides clear guidance on how the FF-ICE message template is constructed.

## FF-ICE/R1 Services Description Example

The purpose of this chapter is to provide informative examples of service descriptions to realize the FF-ICE/R1 services[8] exchanging FIXM flight plans.

### Role of FIXM in support of FF-ICE

As described in section XXX FIXM 4.2.0 introduces the FF-ICE Application Library, a new component including a series of XML Schema templates that support the encoding of the messages identified by FF-ICE/R1. While these structures are an essential building block for realizing the FF-ICE information exchanges, fully materializing the FF-ICE information exchanges requires further descriptions of FF-ICE/R1 Services that exchange FIXM-based FF-ICE/R1 messages.

FF-ICE/R1 Service implementation considerations are strictly speaking beyond the scope of FIXM. This chapter only provides an example of service description of FF-ICE/R1 Service implementations in order to illustrate how FIXM can be used in support of FF-ICE and to help implementers develop solutions for FF-ICE.

Describing a service implementation implies making a series of decisions such as technology selection or naming strategies for service operations. This chapter has no ambition to impose a particular selection of technologies or particular service implementation practices. It provides practical examples for illustration purpose only.

### Target audience

This chapter primarily targets an audience with knowledge of FF-ICE who are interested in understanding how the information exchanges described in the FF-ICE Implementation Guidance [10] can be realized using FIXM and a selection of SOA technologies (e.g. AMQP, WSDL, SOAP). This chapter also serves as a technical introduction to the information exchanges described in the FF-ICE Implementation Guidance Manual, for implementers not yet fully familiar with the FF-ICE concept.

This chapter assumes the reader is familiar with the FF-ICE/R1 principles, procedures and terminology outlined in the Manual on FF-ICE Implementation Guidance [10].

### The FF-ICE/R1 Services

The FF-ICE/R1 Implementation Guidance [10] describes services as a set of messages[9] that should be exchanged in the context of a described behaviour (procedures).

The following diagram illustrates the services identified by FF-ICE/R1 and the associated messages.

A message is a discrete unit of communication intended by the source for consumption by a given recipient or group of recipients. For example, the ***Planning Service*** is described as a set of exchanges of messages such as ***Preliminary Flight Plan Message*** or ***Planning Status Message***.

### Technology Selection

In order to document the FIXM-Based examples in this section, two sets of technologies have been selected that accommodate the need to exchange the FF-ICE/R1 messages:

* **SOAP Web Service technologies** allow to exchange request and reply messages where the initiative is taken by the service consumer.[10]
* **AMQP** supports the exchange of notification messages where the initiative is taken by the service provider.

Appendix XXX Section YYY provides detailed rational for the technology selection.

### Planning Service Description Example

|  |  |
| --- | --- |
| **Service Name** |  |
| Planning Service |  |
| **Abstract** |  |
| The Planning Service enables a CDM process between the eAU and the eASP(s) concerning the intended operation of a flight. It is described in [10], Page II-5-21 Section 5.1. |  |
| **Provision** |  |
| **Provider** | Example Air Services (XAS) |
| **Provider Description** | XAS is a fictitious eASP used for illustration purposes. |
| **Provider Type** | The Planning Service is an optional service expected or recommended to be provided by an eASP whose airspace is complex and/or regularly constrained. |
| **Categorisation** |  |
| **Service Type** | Swim compliant |
| **Life Cycle Stage** | Operational |
| **Business Activity Type** | Demand and capacity balancing |
| **Information Category** | Cooperative network information exchange |
| **Intended Consumer** | The Planning Service is consumed by eAU(s). |
| **Application Message Exchange Pattern** | Request Reply |
| **Operational Need** |  |
| Assist the operator in determining the optimal route/trajectory for a flight by identifying the operational environment and ATM constraints applicable to the flight as proposed. |  |
| Enable ATM service providers to obtain an earlier, more detailed and more accurate assessment of the anticipated traffic demand. |  |
| **Functionality** |  |
| The ability to submit a preliminary flight plan and associated messages (Update, Cancel) and to provide the appropriate response messages (Submission Response, Planning Status) |  |

|  |  |
| --- | --- |
| **Service Interface** |  |
| **Name** | Planning Provider eASP Interface |
| **Description** | TBD |
| **Interface provision side** | Provider side interface |
| **TI primitive message exchange pattern** | Synchronous request response |
| **Service interface binding** | SWIM\_TI\_YP\_1\_0\_WS\_SOAP |
| **Network interface binding** | TI\_YP\_1\_0.IPV4\_UNICAST |

Appendix XXX Section YYY provides detailed information regarding… TBD

### Filing Service Description Example

|  |  |
| --- | --- |
| **Service Name** |  |
| Filing Service |  |
| **Abstract** |  |
| The Filing Service enables the submission of filed flight plans (eFPL) in order to obtain air traffic services. It is described in [10], Page II-6-1 Section 6. |  |
| **Provision** |  |
| **Provider** | Example Service Provider |
| **Provider Description** | The Example Service Provider is a non-existent eASP used for illustration purposes. |
| **Provider Type** | The Filing Service is provided by eASP(s). |
| **Categorisation** |  |
| **Service Type** | Swim compliant |
| **Life Cycle Stage** | Operational |
| **Business Activity Type** | Demand and capacity balancing |
| **Information Category** | Cooperative network information exchange |
| **Intended Consumer** | The Filing Service is consumed by eAU(s). |
| **Application Message Exchange Pattern** | Request Reply |
| **Operational Need** |  |
| An eFPL should be filed in order to obtain air traffic services |  |
| **Functionality** |  |
| The ability to submit a filed flight plan and associated messages (Update, Cancel) and to provide the appropriate response messages (Submission Response, Filing Status) in accordance with FF-ICE procedures. |  |

|  |  |
| --- | --- |
| **Service Interface** |  |
| **Name** | Filing Provider eASP Interface |
| **Description** | TBD |
| **Interface provision side** | Provider side interface |
| **TI primitive message exchange pattern** | Synchronous request response |
| **Service interface binding** | SWIM\_TI\_YP\_1\_0\_WS\_SOAP |
| **Network interface binding** | TI\_YP\_1\_0.IPV4\_UNICAST |

Appendix XXX Section YYY provides detailed information regarding… TBD

## XML Samples

*Align XML Samples with Example services and Application Library*

## Extensions and Restrictions

*Explain here how a Library can be extended (both core and/or the application specific elements) and it can be also restricted via templates. To accommodate regional(or other) requirements.*

## Translating FF-ICE FIXM Messages to ATS Messages

### Target audience

FIXM is not developed as a permanent alternative to the traditional ICAO FPL 2012 format but covers the content of the legacy ATS Messages. This chapter targets FIXM implementers who want to realise a conversion from FF-ICE Messages to ATS message content.

### Overview

The transition from present day practices to FF-ICE operations is likely to be somewhat protracted. This is a topic that is being pursued actively by the ATMRPP [ATMRPP-WG/24-WP/564] [16], and is recognized as a key issue in the System Wide Information Management (SWIM) concept [15] [ICAO Doc 10039]. During that transition period, there will be stakeholders who are able to send and receive flight plan information employing FIXM, while others will employ ICAO ATS messages. In such a hybrid environment, it is expected that a significant effort will be expended translating between the FIXM format and the ATS message format. It is critical for interoperability purposes, and to ensure meaning is not lost in translation, that the conversion between FIXM and ATS message content is precisely defined, and that all stakeholders employ the same translation rules.

There is not a direct correspondence between ATS messages and FIXM, though there is a close association. At the message level, the association is with the FF-ICE messages described in section3.3.1. The mapping from FF-ICE Messages to ATS messages focuses on the individual ATS message fields (7, 8, etc.) rather than the messages themselves. In general, the mapping is independent of the message type: regardless of which ATS message field 7 appears in, the aircraft identification always maps to the same FIXM element. In the cases where an ATS message field item maps to different FIXM elements based on the message type (e.g. field 13b is estimated off block time in a FPL, but actual take off time in a DEP), that difference is made explicit in the mapping rule.

### ATS Message Content to FIXM Logical Model Map

#### Purpose & Scope

This chapter provides a mapping between the Flight Information Exchange Model (FIXM) Logical Model v4.1.0 and International Civil Aviation Organisation (ICAO) Air Traffic Services (ATS) message content as defined in ICAO Doc 4444 [PANS-ATM] [8].

The mapping provides traceability from ATS message content to FIXM ensuring complete coverage of ATS messages.

This chapter defines a mapping from ICAO Doc 4444 [PANS-ATM] ATS message fields to FIXM logical model elements. The scope covers all message content defined in appendix 3 of PANS-ATM [8]. Supporting description is provided where the mapping from ATS message content to the logical model is not clear. The reader is assumed to be familiar with ICAO ATS messages and the FIXM Logical Model.

This chapter does not address the FIXM Extensible Markup Language (XML) schemas. The mapping from the logical model to the XML schemas is relatively straightforward.

#### Guidelines

Section **Error! Reference source not found.** maps the individual data elements in ATS messages to the corresponding elements in the FIXM logical model. It is not always clear how the structural aspects of an ATS message are captured in a FIXM object. This section provides explanation and guidelines where the structure is not clear.

The ATS message format consists of a mixture of structured and free text. The free text components create problems when decoding ATS messages, regardless of whether the goal is to create FIXM objects from the ATS messages. The format of ATS messages is in part dictated by the need for such messages to be readable by a human (presentation is a concern), whereas FIXM focuses purely on the content and structure (presentation is not a concern). Those free text aspects of ATS messages that cause difficulties when decoding are highlighted and discussed.

##### Emergency Message Originator

ATS field 5b is the originator of the emergency message. It consists of eight letters: location indicator (4), ATS unit designator (3), and either ‘X’ or a letter identifying the ATS unit division.

It is only possible to create a valid ATS message field 5b from a FIXM flight if the attribute atcUnitNameOrAlternate is eight upper case letters.

##### SSR Mode

ATS field 7b is Secondary Surveillance Radar (SSR) mode. PANS-ATM restricts this to mode A only. FIXM supports SSR code but does not include explicitly a field for mode (that mode A alone is supported is implicit in the class name: *ModeACode*).

When creating ATS message content from a FIXM object, set the SSR mode (field 7b) to A.

##### Number of Aircraft

ATS field 9a is the number of aircraft. PANS-ATM restricts this value to be in the range 2 through 99. FIXM allows any non-negative number.

When creating ATS message content from a FIXM object, if the Aircraft.formationCount value is greater than 99, truncate to 99.

A similar comment applies to other ATS message fields that contain counts:

* Field 18 TYP (range 2..10);
* Field 19b (range 1..99);
* Field 19f (range 1..99 for number of dinghies, 1..999 for dinghy capacity).

##### Wake Turbulence Category

PANS-ATM supports wake turbulence categories L, M and H (field 9c). However, aircraft operators who operate A380’s often specify a wake turbulence category of J. FIXM supports the value J.

Since J is in common use, when creating a FIXM object from ATS message content, if wake turbulence category J is specified, include that value in the ATS message.

##### Navigation/Communication Capabilities

###### No or Unserviceable Equipment

The value ‘N’ in field 10a of an ATS messages indicates, “no COM/NAV/approach aid equipment for the route to be flown is carried, or the equipment is unserviceable”. FIXM does not explicitly model the field 10a code ‘N’. Rather it leaves that code implicit to avoid redundancy. The relevant items in the FIXM logical model are class *FlightCapabilities* and its associations *navigation*, *communication* and *standardCapabilities*.

* When creating a FIXM object from ATS message content, ignore code ‘N’ in field 10a[11].
* When creating ATS message content from a FIXM object, insert ‘N’ in field 10a if an instance of class *FlightCapabilities* is absent, or it is present and associations *navigation*, *communication* and *standardCapabilities* are all absent.

###### Standard Equipment

The value ‘S’ in field 10a of an ATS message indicates, “Standard COM/NAV/approach aid equipment for the route to be flown is carried and serviceable”. ‘S’ is not specific to navigation or communication capabilities. As such, FIXM represents standard equipment and capabilities via class *StandardCapabilitiesIndicator* that is associated with *FlightCapabilities*, being the lowest level class that sits above the navigation and communication capabilities in the class hierarchy.

###### PBN Approved

The value ‘R’ in field 10a of an ATS message indicates performance based navigation (PBN) capability codes are included in field 18 PBN. FIXM does not explicitly model the field 10a code ‘R’. Rather it leaves that code implicit to avoid redundancy.

* When creating a FIXM object from ATS message content, ignore code ‘R’ in field 10a[12].
* When creating ATS message content from a FIXM object, insert ‘R’ in field 10a if one or more PBN codes are present in the navigation capabilities.

###### Other Equipment and Capabilities

The value ‘Z’ in field 10a of an ATS message indicates that other communication/navigation capabilities are defined in field 18 (NAV, COM, DAT). FIXM does not explicitly model field 10a code ‘Z’. Rather, it leaves that code implicit to avoid redundancy.

* When creating a FIXM object from ATS message content, ignore code ‘Z’ in field 10a[13].
* When creating ATS message content from a FIXM object, insert ‘Z’ in field 10a if at least one of the “other navigation, communication or datalink capabilities” fields is present in the FIXM object.

PANS-ATM states: *If the letter G is used, the types of external GNSS augmentation, if any, are specified in item 18 following the indicator NAV/*. An ATS message may have content in field 18 NAV/ with ‘G’ in field 10a but not ‘Z’. The above rule always inserts ‘Z’ in field 10a if there is content in field 18 NAV/. Consequently, an ATS-FIXM-ATS round trip may insert a ‘Z’ in field 10a that was not in the original ATS message.

###### Equipment/Capabilities Example

Figure 33 presents a flight plan in ICAO 4444 format, with equipment and capabilities related to navigation and communication highlighted.

(FPL-QFA8-IS

-B744/H-SDE2E3FGHIJ3J5M1RWYZ/LB1D1

-KDFW0400

-N0501F280 DCT ABI J4 INK/N0504F300 J50 ELP J26 HMO V2 GRN

2704N11627W 26N119W 2544N12000W 24N126W/M084F320 22N133W 19N139W

16N144W/M084F340 11N152W 06N159W/M084F360 01N166W 01S169W

0500S17435W 06S176W 12S176E/M084F380 18S168E 2125S16300E GUXIB R587

HARVS Q21 SAVER G329 BN DCT

-YBBN1519

-PBN/A1B1D1L1S1 NAV/GPSRNAV RNVD1A1 DOF/130202 REG/VHOEG

DLE/INK0100 26N119W0200 SEL/MQDE

PER/D RIF/GUXIB R587 MEPAB G591 LTO NWWW)

Figure 33: Sample Flight Plan

Figure 34 presents the equipment/capabilities portion of the flight plan as a FIXM object model. Only the highlighted items in Figure 33 are included in the diagram.

Figure 34: Equipment and Capabilities Object Model

##### Surveillance Capabilities

The value ‘N’ in field 10b of an ATS messages indicates, “no surveillance equipment for the route to be flown is carried, or the equipment is unserviceable”. FIXM does not explicitly model the field 10b code ‘N’. Rather it leaves that code implicit to avoid redundancy. The relevant items in the FIXM logical model are class *FlightCapabilities* and its association *surveillance*.

* When creating a FIXM object from ATS message content, ignore code ‘N’ in field 10b[14].
* When creating ATS message content from a FIXM object, insert ‘N’ in field 10b if an instance of class *FlightCapabilities* is absent, or it is present but no surveillance capability codes are present.

##### Date/Time/Duration Specification

###### UTC

Date/time values in ATS messages are always expressed in Coordinated Universal Time (UTC). Likewise, FIXM requires times to be expressed in UTC.

A constraint is placed on class *Base.Types.Time*, the class used to represent all date/time values in FIXM, such that only UTC times can be expressed. The constraint mandates that the time zone is ‘Z’.

Example: 20th July 1969 at 20:18UTC is expressed as

1969-07-20T20:18:00.000Z

In ATS messages, times are expressed in hours and minutes only, while FIXM supports seconds and fractions of seconds. When converting FIXM to ATS message content, seconds should be truncated.

###### Date of Flight

The flight departure time is encoded in field 13b of an ATS message, and is expressed as a four digit UTC value (HHMM). The date on which the flight departs optionally appears in field 18 DOF (YYMMDD). FIXM encodes such values as a full date/time, not as distinct date and time values. As such, the full and unambiguous departure date/time of a flight is composed from fields 13b and 18 DOF[15].

Figure 35 presents the object model corresponding to highlighted parts of the following flight plan fragment.

-YSSY2315

-N0501F280 ....

-YBBN0115

-DOF/141105

Figure 35: Departure Date/Time Object Model

###### Estimated Flight Time

ATS field 16b is the total estimated elapsed time from take-off to landing, consisting of four digits in HHMM format. FIXM models this as a duration which may be of arbitrary length. Consequently, a FIXM flight may include a duration that is not expressible in an ATS message.

The same comments applies to other ATS message fields that contain durations:

* Field 18 EET;
* Field 18 DLE;
* Field 19a.

##### Route

An ATS message route description (field 15) is captured in FIXM by class *RouteTrajectoryGroup* in package *Flight.FlightRouteTrajectory.RouteTrajectory*.

The initial cruising speed (field 15a) and level (field 15b) are captured in class *FlightRouteInformation*. Field 15b of an ATS route may contain the token ‘VFR’ instead of a level. In that case the *cruisingLevel* attribute of *FlightRouteInformation* should be omitted.

The route is modelled as a series of route elements (class *RouteTrajectoryElement*) each consisting of a route point and the designator of the ATS route to the next point, together with associated information such as delay and changes.

Note this package also accommodates 4D trajectories hence is far richer in content than is required for ATS message routes. When mapping from ATS messages to FIXM there is no requirement to create a corresponding 4D trajectory.

###### Varieties of Route

The mapping from field 15 to FIXM is complicated by the fact that a FIXM object, class *Flight*, can be associated with up to five routes or trajectories to support FF-ICE processes. The associations are:

* negotiating (exchange between eASP and eAU during the Planning Phase)
* agreed (by the eASP and eAU during the Planning Phase)
* filed (by the eAU)
* current (latest known information)
* desired (by the eAU)

When mapping ATS message content to FIXM it must be decided which of the associations is employed to model the route information. Such a decision cannot be made with respect to field 15 in isolation. The decision is dependent on the message type in which the route occurs. Table 3 presents the mapping between kinds of route and the message types that contain field 15 (including those where field 15 is incorporated in field 22).

Table 3: Messages Types Supporting Field 15

|  |  |
| --- | --- |
| Message Type | Route Association |
| ALR | current |
| FPL | filed |
| CHG | filed |
| CPL | current |
| CDN | current |

###### Route Text

The primary purpose of FIXM is to provide a structured representation of flight information, with individual elements clearly delineated to ensure the intent of the route is communicated unambiguously. The attribute *routeText* in class *FlightRouteInformation* allows the text of a route description, namely the content of field 15c in the ATS message, to be recorded in the model. This is provided to support legacy systems, and to support stakeholders who may not be in full possession of all necessary aeronautical resource information. When translating from field 15c of an ATS message to FIXM the route structure should be decoded and made explicit.

* When creating a FIXM object from ATS message content, whenever possible decode the route and populate the FIXM route structure.
* When the FIXM route structure is populated, population of the route text is optional.
* For legacy systems where it is not possible to decode the route, the route text only may be populated.
* When the route text is populated, strip leading and trailing white space, and replace multiple contiguous white space characters by a single space.
* When creating ATS message content from FIXM, if the route structure is available, create the field 15c text from the route structure (even if the route text is available).
* Never populate the route text with anything other than a string that complies with the PANS-ATM field 15c route definition.

###### SID and STAR

A SID, if present in the route, is the first item in the route description. A STAR, if present in the route, is the last item in the route description. FIXM encodes the SID and STAR as route designators in the route: attributes *standardInstrumentDeparture* and *standardInstrumentArrival* in class *RouteDesignatorToNextElementChoice*.

* A SID, if included in a route, must appear in the first element of the sequence of instances of *RouteTrajectoryElement*. The *elementStartPoint* attribute of the same element must not be populated.
* A STAR, if included in a route, must appear in the last element of the sequence of instances of *RouteTrajectoryElement*.

###### Direct Route Segments

In ICAO ATS messages, the presence of DCT between two route points indicates the aircraft will fly between the points outside a designated ATS route. In the ICAO ATS message it is also allowed to specify two consecutive route points that are separated neither by an ATS route designator nor by DCT; this is most commonly seen in User Preferred Routes (UPR). In such a case there is an implied DCT between the route points.

FIXM models DCT through class *OtherRouteDesignator*, related to class *RouteDesignatorToNextElementChoice* by attribute otherRouteDesignator.

* When creating a FIXM object from ATS message content, indicate a direct route segment by setting attribute *otherRouteDesaignator* to DIRECT.
* When creating ATS message content from a FIXM object, if *otherRouteDesignator* of class *RouteDesignatorToNextElementChoice* is set to DIRECT, insert “DCT” into the ATS route.
* When creating ATS message content from a FIXM object, if an instance of class *RouteDesignatorToNextElementChoice* is not present, or is present and the value of attribute *otherRouteDesignator* is UNSPECIFIED, do not insert any text between the current and next point.

Refer to Figure 37 for an example of “DCT” in a route.

###### Route Truncation

The token ‘T’ in a route description indicates the route has been truncated. That is, the entire route through to the destination has not been presented. When included, the route truncation indicator must be the last item in the route description.

Route truncation is modelled by class *RouteTruncationIndicator* in package *RouteTrajectory*, and is associated with class *RouteTrajectoryElement*. A route is modelled by an ordered sequence of instances of *RouteTrajectoryElement*. The truncation indicator may only be associated with the last element in the sequence (it is meaningless to truncate a route prior to the last element).

Refer to Figure 37 for an example of route truncation.

###### Route Changes

Route and trajectory information is captured in the FIXM logical model in package *Flight.FlightRouteTrajectory*. The route itself is captured in sub-package *RouteTrajectory*, while changes to speed and level in a route are captured in sub-package *RouteChanges*.

This section defines how the route changes defined in PANS-ATM are mapped to the FIXM logical model. There are three variants allowed in an ATS message: speed/level change, cruise/climb, and cruise/climb with no specific upper limit. One example of each of those changes and how they map to the FIXM logical model is presented in Table 4.

Table 4: Route Changes

Notes:

* The token ‘C’ is inserted in a flight plan to indicate a cruise climb phase. This does not appear in the FIXM logical model. The presence of an instance of class *CruiseClimbStart* indicates cruise climb, as demonstrated in Figure 36.
* The token ‘PLUS’ is used to indicate cruise climb is planned to commence above the specified level. This does not appear in the FIXM logical model. ‘PLUS’ is indicated by an instance of *CruiseClimbStart* where *level* (of class *FlightLevelOrAltitudeOrRangeChoice*) is populated with an instance of FlightLevelOrAltitudeChoice), whereas a cruise/climb with an upper limit is indicated by an instance of *CruiseClimbStart* where *level* is populated with an instance of *VerticalRange*.
* *CrusingSpeedChange* and *CruisingLevelChange* have an optional association *activation*. There is no necessity to populate this attribute.

Figure 36 presents examples of the three kinds of level constraint.

Figure 36: Route Changes Object Model

Figure 37 presents the object model corresponding to the (contrived) ATS message field 15 route

N0430F220 GORLO2N 3910N02230W/N0415F240 DCT C/IVA/N0415F240F250 B9 ENTRA VFR T

Figure 37: Route Object Model

###### RIF

ICAO field 18 RIF, if present, contains the route details to the revised destination, followed by the revised destination aerodrome. This is modelled in FIXM by class *ReclearanceInFlight* in package *Flight.Arrival*. The route component is modelled by attribute *routeToRevisedDestination* and the destination by attribute *filedRevisedDestinationAerodrome*.

The route component is constructed via the same rules as for field 15c. However, in FIXM the route to revised destination is modelled as an unstructured string.

Figure 38 presents the object model corresponding to field 18 RIF of the sample flight plan in Figure 33:

RIF/GUXIB R587 MEPAB G591 LTO NWWW

Figure 38: Route to Revised Destination Object Model

###### DLE

ICAO field 18 DLE, if present, contains points along the route at which delay will occur; the aircraft essentially goes ‘off plan’ for the stated duration. Each DLE point must appear in the route (field 15c). For ATS messages, this requires that a consistency check be performed on the flight plan to ensure the DLE points are listed in the route. FIXM avoids the need for a check and the duplication of route points by incorporating a delay value in the corresponding route element. Specifically, the delay duration appears in attribute *delayValue* of class *EnRouteDelay*, which is associated with class *RouteTrajectoryElement*.

The *EnRouteDelay* class additionally has attributes *delayReason*, *delayReference* and *delayType*. When creating a FIXM object from ATS message content, the attributes *delayReason*, *delayReference* and *delayType* should be omitted.

Figure 39 presents the object model for a fragment of the route in the flight plan contained in Figure 33, incorporating the information in field 18 DLE:

DLE/INK0100 26N119W0200

Figure 39: Route Delay Object Model

##### Aircraft Type

When the type of aircraft that conducts a flight does not have an ICAO aircraft type designator [ICAO Doc 8643] [12] or the flight is a formation, the value ZZZZ is inserted in field 9b and the aircraft type information is inserted in field 18 TYP. The following fragment is an example.

-10ZZZZ/M

....

-TYP/2F15 5F5 3B2

Note the structured nature of the TYP field: two F15s, five F5s, and three B2s. The value in field 18 TYP may exhibit structure as in this example above for a formation. However, this may not be so in other cases, where the (non-designator) type of aircraft is listed, as in

-ZZZZ/L

....

-TYP/ECLIPSE 500

Figure 40 presents the object model corresponding to each of the above flight plan fragments.

Figure 40: Aircraft Type Object Model

Notes:

* If it is not possible to decode the content of field 18 TYP, create a single instance of class *AircraftType* to record the entire content of 18 TYP.
* The sum of the *numberOfAircraft* attributes in the instances of *AircraftType* class should equal the *formationCount* attribute in class *Aircraft*.
* If the number of aircraft is 1, the *formationCount* and *numberOfAircraft* attributes may be omitted (though may be included as in Figure 40).

##### Aircraft Registration

The registration mark for an aircraft may include a ‘-’ character, e.g. VH-ABC. While PANS-ATM does not explicitly state that ‘-’ should be omitted when including field 18 REG, it is rare that ‘-’ is included, i.e. VHABC is the norm. FIXM does not support ‘-’ in the registration.

When creating a FIXM object from an ATS message, strip the ‘-’ character if present in the registration.

FIXM supports multiple registrations. ATS messages support a single registration. When creating a FIXM object from an ATS message, the registration is a sequence one length one.

When creating an ATS message from a FIXM object, if the FIXM object contains multiple registrations, select the first registration in the sequence.

##### Departure Aerodrome

When the departure aerodrome for a flight does not have an ICAO location indicator code [ICAO Doc 7910] [11], the value ZZZZ is inserted in field 13a and the departure point is inserted in field 18 DEP. According to PANS-ATM the content of 18 DEP is “name and location of departure aerodrome” where the location is expressed either as a latitude/longitude or as a bearing and distance from a designated point. In the case the aircraft did not take off from an aerodrome, the first point of the route or the marker radio beacon may be specified. This can be problematic when decoding 18 DEP for the population of FIXM:

* Analysis of flight plans received by Airservices Australia shows that the majority of flight plans that include 18 DEP contain only a latitude/longitude in 18 DEP. This is, strictly speaking, not compliant with PANS-ATM, yet it is common practice.
* The name of the departure aerodrome may consist of multiple words so it may not be obvious how to parse the content of 18 DEP.

Figure 41 presents two object models corresponding to the following flight plan fragment.

-ZZZZ1231

....

-DEP/WESTMEAD HOSPITAL 3349S15059E

Figure 41: Departure Aerodrome Object Model

The first shows the fully decoded 18 DEP. The second shows the approach where 18 DEP cannot be decoded successfully: insert the entire content of 18 DEP in the *name* attribute of *AerodromeReference*.

##### Destination Aerodrome

When the destination aerodrome for a flight does not have an ICAO location indicator code [ICAO Doc 7910] [11], the value ZZZZ is inserted in field 16a and the destination point is inserted in field 18 DEST. According to PANS-ATM the content of 18 DEST is “name and location of destination aerodrome” where the location is expressed either as a latitude/longitude or as a bearing and distance from a named significant point. This can be problematic when decoding 18 DEST for the population of FIXM:

* Analysis of flight plans received by Airservices Australia shows that the majority of flight plans that include 18 DEST contain only a latitude/longitude in 18 DEST. This is, strictly speaking, not compliant with PANS-ATM, yet it is common practice.
* The name of the departure aerodrome may consist of multiple words so it may not be obvious how to parse the content of 18 DEST.

Refer to section Departure Aerodrome for an equivalent example in the context of field 18 DEP.

##### Arrival Aerodrome

An ATS arrival message records the arrival aerodrome in field 17. If the arrival aerodrome does not have an ICAO location indicator code, the value ZZZZ is inserted in field 17a and the arrival aerodrome name is recorded in field 17c.

FIXM accommodates both a destination (intended) aerodrome and an actual arrival aerodrome.

* Record the actual arrival aerodrome in attribute *arrivalAerodrome.locationIndicator* of class *Arrival.AerodromeReference*;
* If the actual arrival aerodrome does not have an ICAO location indicator, record the arrival aerodrome name against attribute *arrivalAerodorme.name* of class *Arrival.AerodromeReference*.

Figure 42 presents an object model for destination/arrival information assuming reception of the FPL

(FPL-RAQ-VG

-C172/L-V/C

-YBSU0540

-N0115A035 DCT

-YRED0021

-DOF/140622 REG/RAQ)

Followed by the ARR

(ARR-RAQ-YBSU-YRED-ZZZZ0622 CABOOLTURE)

Figure 42: Arrival Aerodrome Object Model

##### Alternate Destination

When the alternate destination aerodrome for a flight does not have an ICAO location indicator code [ICAO Doc 7910] [11], the value ZZZZ is inserted in field 16c and the alternate destination point is inserted in field 18 ALTN. Although similar to 18 DEP and 18 DEST there is an added complication that up to two alternates may be specified, hence 18 ALTN could include two name/location pairs.

The alternate destination aerodromes are captured by FIXM in attribute *destinationAerodromeAlternate* of class *Arrival*.

The following flight plan fragment presents field 16 and field 18 items that relate to destination aerodrome and alternates.

-ZZZZ0035 YSBK ZZZZ

……..

-DEST/WESTMEAD HOSPITAL 3348S15059E ALTN/EASTERN CREEK

Figure 43 presents the FIXM representation in an object model.

Figure 43: Destination and Alternate Object Model

Decoding is problematic if two free text names are included in ALTN. For example, consider the flight plan fragment

-YSBK0035 ZZZZ ZZZZ

……..

-ALTN/WESTMEAD HOSPITAL EASTERN CREEK

where “WESTMEAD HOSPITAL” and “EASTERN CREEK” are distinct points. None of the tokens is a latitude/longitude or a bearing&distance, so it is very difficult to distinguish them. In this case create a single alternate location (instance of *AerodromeReference*) and set the *name* attribute to “WESTMEAD HOSPITAL EASTERN CREEK”.

##### En-Route Alternate

ICAO field 18 RALT, if present, indicates the (one or more) en-route alternates. Each alternate is one of:

* ICAO location indicator;
* Aerodrome name as listed in Aeronautical Information Publication (AIP);
* Geographic location as a latitude/longitude;
* Bearing and distance from a designated point.

An en-route alternate is represented in the model by attribute *alternateAerodrome* of class *EnRoute* in package *Flight.EnRoute*. Each alternate is an *AerodromeReference* (see section AerodromeReference).

Figure 44 presents two object models that represent the en-route alternate listed below.

RALT/YSBK WESTMEAD HOSPITAL SY102025

Figure 44: En-Route Alternate Object Model

The first shows the fully decoded 18 RALT. The second shows the approach where 18 RALT cannot be decoded successfully: insert the entire content of 18 RALT in the *name* attribute of *AerodromeReference*.

##### Take-off Alternate

ICAO field 18 TALT, if present, indicates the (one or more) take-off alternates. Each alternate is one of:

* ICAO location indicator;
* Aerodrome name as listed in AIP;
* Geographic location as a latitude/longitude;
* Bearing and distance from a designated point.

A take-off alternate is represented in the model by attribute *takeOffAlternateAerodrome* of class *Departure* in package *Flight.Departure*. Each alternate is an *AerodromeReference* (see section AerodromeReference).

Refer to section En-Route Alternate for an equivalent example in the context of en-route alternate.

##### Air Filed

When a flight plan is filed in the air, the value AFIL is inserted in field 13a and the ATS unit from which supplementary flight plan information can be obtained is specified in field 18 DEP. The mapping employs the attribute *flightPlanSubmitter* of class *Flight* for this purpose, though the name is not immediately suggestive of the purpose for which it is being used. In this situation the following rules should be applied:

* Populate the *name* attribute of *PersonOrOrganization* (via attribute *flightPlanSubmitter*) with the content of field 18 DEP.
* Populate the *airfileIndicator* of class *Departure* (with the constant value AIRFILE).
* Populate the attribute *airfileRouteStartTime* of class *FlightRouteInformation* in package *Flight.FlightRouteTrajectory.RouteTrajectory* with the content of field 13b.
* The departure aerodrome (*aerodrome*) and departure time (*estimatedOffBlockTime*) of class *Departure* are not populated.

Figure 45 presents the FIXM representation of the following air filed flight plan (fragment) as an object model.

-AFIL1254

....

-DEP/YBBBZQZA

Figure 45: AFIL Object Model

##### Remarks

The remarks item (RMK/) of field 18 of a flight plan maps to attribute *remarks* of class *Flight*. The content of remarks should not include the ‘RMK/’ label. That is, a flight plan containing

RMK/TCAS II EQUIPPED

results in

remarks = “TCAS II EQUIPPED”

The same is true of all field 18 items. The item label is not included in the content; it is implied by the structure.

##### Supplementary Information

Supplementary information (field 19) contains additional information about a flight that is not transmitted in the flight plan.

Field 19b is the number of persons on board. Appendix 2 of PANS-ATM suggests ‘TBN’ is inserted in field 19b if the number of persons on board is not known. Appendix 3 suggests field 19b is omitted if the value is not known. FIXM does not allow a distinction between the absence of field 19b and its presence with value ‘TBN’.

* When converting ATS message content to FIXM, if field 19b is populated with ‘TBN’, omit the *personsOnBoard* attribute from the FIXM *SupplementaryData* object.
* When converting a FIXM object to ATS message field 19, if the *personsOnBoard* attribute is absent, do not include any text for field 19b.

The above rule means an ATS-FIXM-ATS round trip would cause the text ‘P/TBN’ to be removed from the original ATS message.

Figure 46 presents the object model corresponding to the following field 19 example.

–E/0745 P/6 R/VE S/M J/L D/2 8 C YELLOW A/YELLOW RED TAIL N/145E C/SMITH

Figure 46: Supplementary Information Object Model

##### Alerting Search and Rescue Information

Field 20 of an ATS message, alerting search and rescue information, consists of eight items, each of which, if not known by the originator, is replaced by ‘NIL’ or ‘NOT KNOWN’. The first five items are precisely defined, but the final three are free text fields, which leads to difficulties when decoding.

It is beyond the scope of this chapter to address such a decoding issue.

##### Radio Failure Information

Field 21 of an ATS message, radio failure information, consists of six items, each of which, if not known by the originator, is replaced by ‘NIL’ or ‘NOT KNOWN’. The first four items are precisely defined, but the final two are free text fields, which leads to difficulties when decoding.

It is beyond the scope of this chapter to address such a decoding issue.

#### Base Constructs

The ATS messages to FIXM logical model map in section 4.4.4 at times maps an ATS message field to a structured FIXM entity without providing further detail. This occurs with ‘lower level’ entities that are defined in the FIXM *Base* package. One such example is field 15a, which is mapped to the *Base* class *TrueAirspeed*.

This section provides further detail for the mapping to *Base* classes where those classes represent compound values.

##### FlightLevelOrAltitude

A level or altitude is captured in FIXM by the class *FlightLevelOrAltitudeChoice* in package *Base.RangesAndChoices*. It consists of choice between flight level (class *FlightLevel*) or altitude (class *Altitude*). In each case a unit of measure is specified (respectively *UomFlightLevel* and *UomAltitude*) and a vertical distance (class *VerticalDistance* in package *Base.Measures*) expressed as a floating point number. Table 5 provides a mapping between the level/altitude in PANS-ATM ATS messages and the level/altitude in FIXM.

Table 5: Level/Altitude Mapping

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ATS Message** | **FIXM** |  |  |  |
| Type | Value | FlightLevelOrAltitude | *Uom* | Value |
| F | Imperial flight level | FlightLevel | FL | Imperial flight level |
| S | Metric flight level |  | SM | Metric flight level |
| A | Altitude in hundreds of feet | Altitude | FT | Altitude in feet |
| M | Altitude in tens of metres |  | M | Altitude in metres |

Notes:

* For ICAO flight level type ‘F’, the ICAO and FIXM values are the same (though the ICAO value is a whole number while the FIXM value is a floating point number).
* For ICAO flight level type ‘S’, the ICAO and FIXM values are the same (though the ICAO value is a whole number while the FIXM value is a floating point number).
* For ICAO altitude type ‘A’, multiply by 100 when converting to FIXM.
* For ICAO altitude type ‘A’, divide by 100 and round when converting from FIXM.
* For ICAO altitude type ‘M’, multiply by 10 when converting to FIXM.
* For ICAO altitude type ‘M’, divide by 10 and round when converting from FIXM.
* Since rounding is necessary when converting from FIXM, a round trip transformation is not guaranteed to fully preserve meaning. For example, the FIXM altitude ‘2640 feet’ becomes ‘A026’ in which if converted back to FIXM becomes ‘2600 feet’.

##### TrueAirspeed

In ATS messaging speed is either true air speed or Mach number. This is captured by class *TrueAirspeed* in package *Base.Measures*. It consists of the unit of measurement (class *UomAirspeed*) and a (floating point) value. Table 6 provides a mapping between the speed in ATS messages and the speed in FIXM.

Table 6: Speed Mapping

|  |  |  |  |
| --- | --- | --- | --- |
| **ATS Message** | **FIXM** |  |  |
| Type | Value | UomAirspeed | Value |
| K | Kilometres per hour | KM\_H | Kilometres per hour |
| N | Nautical miles per hour | KT | Nautical miles per hour |
| M | Hundredths of Mach number | MACH | Mach number |

Notes:

* In an ATS message the Mach value is represented by a three-digit string, which when interpreted as a number is 100 times greater than the Mach value (e.g. M080 is Mach 0.8).
* Converting from FIXM to ATS message, multiply by 100 and round.
* Converting from ATS message to FIXM, divide by 100.
* Since rounding is necessary when converting from FIXM, a round trip transformation is not guaranteed to preserve meaning. For example, the FIXM Mach value of ‘0.841’ becomes ‘M084’ which when converted back to FIXM becomes ‘0.84’.

##### GeographicalPosition

In ATS messages, a geographic location is defined either in full degrees, with a corresponding direct representation in decimal degrees, or in degrees and minutes

dd1mm1[NS]ddd2mm2[EW]

which is converted by

decimal latitude = dd1 + (mm1/60)

decimal longitude = ddd2 + (mm2/60)

When converting from FIXM to ATS messages, the position can only be represented to the nearest minute, resulting in a loss of precision.

A round trip starting from FIXM may not preserve meaning. Example:

FIXM latitude: 12.42 degrees

Convert to ATS: 1225N (12 degrees, 25 minutes)

Convert back to FIXM: 12.4166… degrees

##### SignificantPoint

A significant point can be a designated point (navaid or waypoint), a geographic location (latitude/longitude), or a relative point (bearing and distance from a designated point). Three subclasses of *SignificantPoint* (which is abstract) capture the options:

* Class *DesignatedPointOrNavaid* models the designator value via attribute *designator* of class *SignificantPointDesignator*.
* Class *RelativePoint* models the relative point via attributes *referencePoint* (a *DesignatedPointOrNavaid*), *bearing* and *distance*.
* Class *PositionPoint* models the point via attribute *position* of class *GeographicalPosition* (section GeographicalPosition).

Examples of significant points are presented in Figure 41 and Figure 44.

##### Frequency

Radio frequency can appear in fields 20d and 21b of an ATS message. In all examples in PANS-ATM this is presented as an unadorned decimal number (e.g. 126.7). The expanded text in PANS-ATM describing the examples always states MHz.

The global guidance for ATC Interfacility Data Communications (AIDC) [PAN AIDC ICD] [14] is more specific as presented in Table 7.

Table 7: PAN AIDC ICD Frequency

|  |  |  |
| --- | --- | --- |
|  | **Range** | **Units** |
| HF | 2850 to 28000 | kHz |
| VHF | 117.975 to 137.000 | MHz |
| UHF | 225.000 to 399.975 | MHz |

The mapping for ATS messages follows the PAN AIDC ICD convention.

FIXM captures radio frequency in class *Frequency* of package *Base.Measures*. This class has a mandatory associated value: the unit of measure (class *UomFrequency*). The frequency unit is either KHZ or MHZ. If the frequency is four or five digits without a decimal point, set the *uom* attribute to KHZ, otherwise set the *uom* attribute to MHZ.

# Using FIXM for other use cases

FIXM provides globally harmonized flight data definitions and a standardized, hierarchical organization for representing information about a flight. Though its main requirements driver is currently ICAO FF-ICE, FIXM is intended as a general standard for representing all flight data, and the use of FIXM is encouraged for any situation in which flight data is exchanged between systems. Providing flight data in FIXM format has the additional benefit of allowing the data to be more easily ingested by any system already set up to process other FIXM feeds. Essentially, the more FIXM is used, the more useful it becomes.

To help illustrate the use of FIXM outside of FF-ICE, this chapter imagines a fictitious FIXM user, Example Air Services (XAS), that expands their use of FIXM through increasingly complex use cases. Each section below includes a high-level example of how the user accomplishes their goals using the type of FIXM product described in the section. When appropriate, these use cases are paired with the step-by-step examples of how to build various FIXM products provided in the Appendices at the end of this document.

## Using FIXM Core without an Application Library

In some cases, the nature of the messaging infrastructure employed for a particular data exchange makes the use of Application Libraries unnecessary or irrelevant (perhaps due to the infrastructure’s robust metadata/messaging header support) or the nature of the exchange itself does not require any accompanying message data structures (perhaps due to the exchange’s simplicity). In these situations, the use of FIXM Core alone should be sufficient.

FIXM Core is the repository in which all globally applicable flight data structures reside. The root field of the entire flight information hierarchy is the Flight class (in the physical model, the Flight element).

When using FIXM Core for data representation, all XML documents must begin with this Flight element. Similarly, the Fixm.xsd schema file is the root schema of FIXM Core. Whether validating FIXM Core XML documents or using automated code generation utilities (such as JAXB), this is the schema file that should be referenced.

*Example: Departure/Arrival Alerts*

Our fictitious user XAS begins their use of FIXM wanting to publish departure and arrival alerts for flights they monitor. XAS sets up a publishing service with which they send out arrival and departure messages to a single endpoint their consumers can monitor to receive the alerts. Due to the simplicity of their service, the use of FIXM Core alone is sufficient for their needs. XAS constructs XML messages starting with the Flight element to convey their data and instructs their consumers to validate these messages against FIXM Core’s Fixm.xsd schema file. Below is an example of how the XML payload of a departure alert coming from this service may appear.

<?xml version="1.0" encoding="UTF-8"?>

<fx:Flight xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:fb="<http://www.fixm.aero/base/4.2">>

<fx:departure>

<fx:actualTimeOfDeparture>2020-01-01T00:03:00Z</fx:actualTimeOfDeparture>

<fx:aerodrome>

<fb:locationIndicator>KBOS</fb:locationIndicator>

</fx:aerodrome>

</fx:departure>

<fx:flightIdentification>

<fx:aircraftIdentification>ABC1234</fx:aircraftIdentification>

</fx:flightIdentification>

<fx:gufi codeSpace="urn:uuid">3e7f6a63-6c3b-4f0f-844b-4b84338ed103</fx:gufi>

</fx:Flight>

## Using FIXM Core with an Application Library

### Basic Message Application Library

The Basic Message Application Library is intended to enhance FIXM Core by providing basic messaging support for users, including message types and timestamps, as well as the ability to batch multiple flight messages together into a single aggregate message. It also provides extension hooks for users who wish to add their own custom messaging fields. Users who only require this basic level of message support are encouraged to use the Basic Message Application Library.

This application library contains two root fields that can be used as an entry point: Message and MessageCollection.

When using Basic Message for data representation, all XML documents must begin with one of these two elements. Similarly, like Fixm.xsd for Core, the BasicMessage.xsd schema file is the root schema of the Basic Message Application Library and is the file that should be referenced for validation or use with any XML utilities.

Unlike FF-ICE Message, the Basic Message Application Library focuses only on providing users with generic and reusable message data structures. It does not provide any message template since it is not linked to any particular operational use of FIXM.

Users who wish to include additional message data structures beyond what is provided in Basic Message (but who do not wish to create templates for a pre-defined set of messages) are encouraged to do so via creating an Extension to Basic Message (see Section 4.3.1 and Section 4.3.3 below for more on this). Users who wish to create message templates for their systems are encouraged to do so via creating their own Application Library (see Section 4.2.2 for details).

*Example: Batch Updates*

Returning to our fictitious user, XAS has launched a successful departure and arrival alert service using FIXM Core alone but is now interested in expanding their capabilities. Some of XAS’s consumers suffer from network outages and have requested an additional service which they could use to invoke a bulk update containing all the alerts they might have missed during such an outage.

XAS determines that Basic Message should be sufficient to meet the needs of this new service. The MessageCollection element allows XAS to batch together as many alerts as needed for the update, and the timestamp associated with each message provides the additional benefit of letting the consumer know exactly when the alert had originally been sent. XAS decides to construct all updates using MessageCollection as the root element to make parsing the updates more consistent and instructs recipients of these updates to validate the XML against BasicMessage.xsd. Below is a snippet of what the XML payload of such an update may look like.

<?xml version="1.0" encoding="UTF-8"?>

<msg:MessageCollection xmlns:msg="<http://www.fixm.aero/app/msg/1.0"> xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:fb="<http://www.fixm.aero/base/4.2">>

<msg:message>

<msg:flight>

<fx:departure>

<fx:actualTimeOfDeparture>2020-01-01T00:03:00Z</fx:actualTimeOfDeparture>

<fx:aerodrome>

<fb:locationIndicator>KBOS</fb:locationIndicator>

</fx:aerodrome>

</fx:departure>

<fx:flightIdentification>

<fx:aircraftIdentification>ABC1234</fx:aircraftIdentification>

</fx:flightIdentification>

<fx:gufi codeSpace="urn:uuid">3e7f6a63-6c3b-4f0f-844b-4b84338ed103</fx:gufi>

</msg:flight>

<msg:timestamp>2020-01-01T00:03:01Z</msg:timestamp>

<msg:type>DEPARTURE</msg:type>

</msg:message>

.

.

.

<msg:message>

<msg:flight>

<fx:arrival>

<fx:actualTimeOfArrival>2020-01-01T23:58:00Z</fx:actualTimeOfArrival>

<fx:arrivalAerodrome>

<fb:locationIndicator>KLAX</fb:locationIndicator>

</fx:arrivalAerodrome>

</fx:arrival>

<fx:flightIdentification>

<fx:aircraftIdentification>XYZ1234</fx:aircraftIdentification>

</fx:flightIdentification>

<fx:gufi codeSpace="urn:uuid">3808e010-3c24-4a04-afd2-f62ba9ec43f6</fx:gufi>

</msg:flight>

<msg:timestamp>2020-01-01T23:58:01Z</msg:timestamp>

<msg:type>ARRIVAL</msg:type>

</msg:message>

<msg:timestamp>2020-01-02T00:05:00Z</msg:timestamp>

<msg:type>BULK\_UPDATE</msg:type>

</msg:MessageCollection>

### Creating an Application Library

If the organization of Basic Message does not suit the user’s data exchange or if the user wants to create message templates to more fully lock down and describe their message structures and content, they should consider creating their own custom Application Library.

As described in Section 2.2.2 above, Application Libraries enhance FIXM Core by adding context specific message data structures and as well as stricter validation rules via message templates. An Application should define its own namespace to distinguish it from FIXM Core as well as creating one or more root elements to be used as an entry point into the Library. If the Application includes message templates, it may have more than one root schema: one for using the Application Library alone with no further restrictions and one (or more) for use with the templates. The FF-ICE Message Application Library is a good example of this, with users referencing FficeMessage.xsd for unrestricted use of the Library, FficeTemplates.xsd for making use of all thirteen templates used to represent the FF-ICE messages, or one of the thirteen template-specific schemas files corresponding to each FF-ICE message.

While the content and organization of an Application Library depends entirely on the needs of the data exchange it is intended to support, the FF-ICE Message Application and Basic Message Libraries should provide a useful set of examples for how to build a Library with and without associated templates. To supplement this, Appendix A below provides step-by-step instructions on how to create a simple Application.

*Example: Upgraded Alerts*

At this point, our fictitious user XAS has decided to upgrade their original alert service to be able to send departure and arrival messages to specific recipients (rather than maintaining a single, common endpoint for all consumers) as well as making use of templates to clearly lockdown the expected format of the alert messages. To accomplish this, XAS decides to create their own Application Library.

This custom Application Library defines its own namespace (“<http://www.fixm.aero/app/example/1.0”>) and root element(“ExampleMessage”) as well as a number of header fields needed to represent data XAS wants to exchange with each alert (“sender”, “recipient”, “timestamp”, and “type”). XAS then goes on to create two templates: one that locks down the content of a departure alert and another for the arrival alert. Details on how to build this Application Library along with more specifics as to its content are supplied below in Appendix A.

With the Application Library built, XAS instructs consumers to make use of the ExampleTemplates.xsd file described in Appendix A when validating the new alert messages. Below is an example of how the XML payload of one of the new arrival alert messages coming from this service may appear.

<?xml version="1.0" encoding="UTF-8"?>

<xmg:ExampleMessage xsi:type="xmg:ExampleDA\_ExampleMessageType" xmlns:xmg="<http://www.fixm.aero/app/example/1.0"> xmlns:fb="<http://www.fixm.aero/base/4.2"> xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance">>

<xmg:flight>

<fx:departure>

<fx:actualTimeOfDeparture>1903-12-17T03:35:00Z</fx:actualTimeOfDeparture>

<fx:aerodrome>

<fb:name>KILL DEVIL HILL</fb:name>

<fb:referencePoint srsName="urn:ogc:def:crs:EPSG::4326">

<fb:pos>36.019970 -75.668760</fb:pos>

</fb:referencePoint>

</fx:aerodrome>

</fx:departure>

<fx:flightIdentification>

<fx:aircraftIdentification>WRF01</fx:aircraftIdentification>

</fx:flightIdentification>

<fx:gufi codeSpace="urn:uuid">18611e54-52b8-4fb5-a2fa-12173b1d39db</fx:gufi>

</xmg:flight>

<xmg:recipient>

<fb:name>HISTORY</fb:name>

</xmg:recipient>

<xmg:sender>

<fb:name>ORVILLE WRIGHT</fb:name>

</xmg:sender>

<xmg:timestamp>2020-01-15T17:20:33Z</xmg:timestamp>

<xmg:type>DEPARTURE</xmg:type>

</xmg:ExampleMessage>

## Using FIXM Core with an Extension

### Creating a new Extension

If a FIXM user requires additional fields beyond what is available in FIXM Core or an Application Library, Extensions can be used to meet this need. Similar to Applications, Extensions should define their own namespaces to distinguish them from FIXM Core, Application Libraries, and each other. Extensions should also provide a root schema file (that will import FIXM Core and/or the Application Library the Extension works with) for use with XML validators and utilities. Unlike Applications, Extensions should not define their own root elements but, rather, make use of the root elements defined in whatever schemas they extend.

As noted in 2.2.3 above, there are a number of general guidelines for constructing FIXM Extensions (e.g., make use extension hooks, don’t duplicate Core fields, etc.). Apart from that, the content and organization of an Extension is largely dependent on the data set the user wishes to represent. Appendix B below provides a detailed, step-by-step example of how to create a simple Extension that should help guide any users interested in creating their own.

*Example: Position Reports*

Our fictitious user XAS next decides to branch out beyond just providing departure and arrival alerts. A number of consumers have been asking if periodic position reports are available for the flights XAS monitors. While XAS has this data, there are currently no fields present in FIXM Core for representing an active flight’s current position. To address this, XAS decides so create their own FIXM Extension.

This new Extension defines its own namespace (“<http://www.fixm.aero/ext/example/1.0”>) as well as a number of fields XAS wishes to add to what is currently available in FIXM (“sequenceNumber” and “position”). Based on lessons learned during their work with departure and arrival alerts, XAS choses to apply the Extension to Basic Message as well as Core to take advantage of Basic Message’s existing MessageCollection element. This allows XAS the option of using Message as a root element if they wish to send single position reports or MessageCollection if they wish to send many at once. Details on how to build this Extension along with more specifics as to its content are supplied below in Appendix B.

With the Extension built, XAS instructs consumers to make use of the Example.xsd file described in Appendix B when validating position reports. Below is an example of how the XML payload of one of these position reports may appear.

<?xml version="1.0" encoding="UTF-8"?>

<msg:Message xmlns:xmp="<http://www.fixm.aero/ext/example/1.0"> xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:fb="<http://www.fixm.aero/base/4.2"> xmlns:msg="<http://www.fixm.aero/app/msg/1.0"> xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance">>

<msg:extension xsi:type="xmp:ExampleMessageType">

<xmp:sequenceNumber>1</xmp:sequenceNumber>

</msg:extension>

<msg:flight>

<fx:enRoute>

<fx:extension xsi:type="xmp:ExampleEnRouteType">

<xmp:position srsName="urn:ogc:def:crs:EPSG::4326">

<fb:pos>36.019970 -75.668760</fb:pos>

</xmp:position>

</fx:extension>

</fx:enRoute>

<fx:flightIdentification>

<fx:aircraftIdentification>WRF01</fx:aircraftIdentification>

</fx:flightIdentification>

<fx:gufi codeSpace="urn:uuid">6964698b-2074-4fef-868f-ebe65f47a105</fx:gufi>

</msg:flight>

<msg:timestamp>1903-12-17T03:35:00Z</msg:timestamp>

<msg:type>POSITION\_REPORT</msg:type>

</msg:Message>

### Using Multiple Extensions

There are times when a FIXM user may wish to create an XML document that makes use of more than one Extension at the same time. Assuming the Extensions are built following FIXM best practices, this should present no difficulties.

FIXM encourages users to attach their Extensions to Core or an Application via the built-in extension hooks located throughout the models. The primary reason for this is to allow FIXM to support multiple Extensions simultaneously. Each Extension hook has a high multiplicity (standardly 0..2000) that allows many different Extensions to target the same areas of FIXM without interfering with each other.

The simplest way to allow multiple Extensions to work together in a single XML document is to create a new schema file that imports all the needed components (Core, Application Library, and Extensions) into one place[16]. This new schema can be used as the root schema file for XML validators and utilities. Each Extension can then be applied to its own extension hook, and the multiple Extensions can used together as needed.

*Example: Multiple Extensions*

As XAS’s FIXM operations expand, they begin exchanging data with other systems that have created their own Extensions. One such system has an Extension which supplies enhanced handoff data used when flights transition from one controller to another.

XAS wishes to create a new position report service that contains this additional handoff data. They decide to pursue this by creating a new schema that allows the two Extensions to be used at the same time by importing all the needed components into one place and then instructing consumers to use this as the root schema file when validating these new reports. Below is an example of what that schema file may look like.

<?xml version="1.0" encoding="utf-8"?>

<xs:schema xmlns:xs="<http://www.w3.org/2001/XMLSchema"> xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:fb="<http://www.fixm.aero/base/4.2"> xmlns:msg="<http://www.fixm.aero/app/msg/1.0"> xmlns:xmp="<http://www.fixm.aero/ext/example/1.0"> xmlns:hdf="<http://www.fixm.aero/ext/handoff/1.0"> elementFormDefault="qualified" version="1.0.0">

<xs:import namespace="<http://www.fixm.aero/base/4.2"> schemaLocation="../../core/base/Base.xsd"/>

<xs:import namespace="<http://www.fixm.aero/flight/4.2"> schemaLocation="../../core/flight/Flight.xsd"/>

<xs:import namespace="<http://www.fixm.aero/app/msg/1.0"> schemaLocation="../../applications/basicmessage/BasicMessage.xsd"/>

<xs:import namespace="<http://www.fixm.aero/ext/example/1.0"> schemaLocation="../example/Example.xsd"/>

<xs:import namespace="<http://www.fixm.aero/ext/handoff/1.0"> schemaLocation="../handoff/Handoff.xsd"/>

</xs:schema>

### Using an Extension together with an Application

In principle, as can be seen in detail in Appendix B below, applying an Extension to an Application is no different than applying one to FIXM Core. In fact, applying an Extension to the Basic Messaging Application Library is the recommended approach for adding additional message data structures when no templates are needed, and applying Extensions to Application Libraries that include templates should be no different (assuming the templates retain their extension hooks). It is just a matter of importing the Application Library in question and making use of its extension hooks. That said, there are some aspects of using Extensions and templates together that have not yet been fully explored.

One area under active investigation is applying an Application Library directly to an Extension. To date, the only two Application Libraries that have been developed are Basic Message and FF-ICE Message. Both of these Applications only apply themselves to Core. In theory, an Application Library could directly import an Extension just as easily as it imports Core and apply templates to the Extension content in the same way it does to Core fields. As practical examples of this are explored, this section will be updated with more information about how to proceed (or warnings as to why this is discouraged).

*Example: Position Report Template*

Returning to our fictitious user one final time, XAS has created their own Application Library for distributing departure and arrival alerts but has a separate feed that makes use of a different set of schemas for distributing their position reports. XAS would prefer to consolidate their two feeds into one and use the same set of schemas for all of their data.

XAS decides to update their position report Extension to target their own Example Message Application Library rather than Basic Message and add a new POSITION\_REPORT enumeration to the Application’s type field (see Appendix A and Appendix B for details). This should be sufficient to allow XAS to use one set of schemas for all of their data sets. However, this creates an odd discrepancy between departure/arrival alerts and position reports. The alerts are fully described in the Application’s templates while position reports receive no such guidance. Without applying the Application to the Extension as well as Core, there does not seem to be a clear way forward to address this.

As noted above, the best way to approach this matter is currently under investigation so this example is only provided to illustrate the issues involved, not detail the recommended solution. This section will be updated as appropriate after best practices have been established.

# FIXM XML Samples

A set of sample FIXM formatted flight data [2] has been created and is available within the Implementation Guidance zip archive on the FIXM.aero website [5]. The intent is to provide implementers with examples of how an assortment of standard flight data items should be expressed in FIXM. This has been done primarily by using current ATS messages as a source of sample data.

*Note: While exchanging ATS messages is not the main intent of FIXM, ATS messages were used because they provide a real-world source of sample data. In addition, this activity provides concrete examples of the mapping from ATS Message Content to FIXM that is presented in Chapter* ***Error! Reference source not found.****.*

Samples are organized as a set of files, where each file contains FIXM representation of an ATS message type that is documented in ICAO Doc 4444 [PANS-ATM]. None of the sample files includes every possible field or every possible alteration of the field, but the set as a whole represents diverse operational data.

## FIXM XML Structural Overview

The FIXM XML Schema, based on the FIXM Logical Model, arranges information about flights in a different manner than the ATS message format. While the ATS format emphasizes space (and therefore bandwidth) savings, the FIXM format emphasizes structure and clarity. Here is a “skeleton” for a typical Flight Plan message. It does not show every field that will be included in upcoming FF-ICE messages but shows many of the common fields found in the ATS messages that have been converted to FIXM format. Some messages, such as Departure and Arrival, do not have route trajectory and aircraft information, for example, and are therefore much more compact than messages containing full Flight Plan data.

<?xml version="1.0" encoding="UTF-8"?>

<mesg:Message xmlns:fb="<http://www.fixm.aero/base/4.1"> xmlns:fx="<http://www.fixm.aero/flight/4.1"> xmlns:mesg="<http://www.fixm.aero/messaging/4.1"> xmlns:xlink="<http://www.w3.org/1999/xlink"> xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance"> messageType="FPL">

<!-- This message header area contains general information, such as senders and receivers -->

<mesg:uniqueMessageIdentifier>

<!-- A universally unique identification (uuid) for this message -->

</mesg:uniqueMessageIdentifier>/>

<mesg:flight>

<fx:aircraft>

<!-- Information such as aircraft type, registration and capabilities appears here -->

</fx:aircraft>

<fx:arrival>

<!-- Destination aerodrome and alternates appear here. In the case of an arrival message, arrival time and aerodrome also appear -->

</fx:arrival>

<fx:departure>

<!-- Departure aerodrome, alternates, and estimated off-block time go here -->

</fx:departure>

<fx:filed>

<!-- The route trajectory for the flight is shown here. It can contain climb and descent schedules and profiles, 0 to 2000 elements, route information, and take-off weight -->

<fx:element>

<!-- Each element contains information pertinent to a single point in the trajectory, such as distance, constraints, enroute delays, route changes, name of the point this element represents, and the name (if any) of the route along which the flight will proceed to the next element -->

</fx:element>

<fx:routeInformation>

<!-- Contains the route text where available, cruising level and speed, estimated elapsed time. -->

</fx:routeInformation>

</fx:filed>

<fx:flightIdentification>

<!-- The aircraft ID/call sign/tail number by which this flight is identified by air traffic services -->

</fx:flightIdentification>

<fx:gufi>

<!-- Globally unique identifier for this flight which can be used to correlated different transactions for this flight -->

</fx:gufi>

</mesg:flight>

</mesg:Message>

The sample messages were generated in three groups, described in the following sections.

*Note: As of the publication date of this document, the specific technical vision for the global FF-ICE/R1 implementation is still under discussion by ICAO ATMRPP. While FIXM achieves global harmonisation of the flight information exchanged by FF-ICE services, it is not yet decided whether particular aspects related to FF-ICE services implementation, such as Messaging, should be subject to global harmonisation.*

*The samples below rely on the Messaging package of FIXM 4.1.0. This package has been intentionally deprecated by* [*Change Request 28*](https://ost.eurocontrol.int/sites/FIXM/Change%20Requests/Deprecating_the_Messaging_package.docx) *as a signal to FIXM implementers that its use is not necessarily recommended for global use for the time being, considering the uncertainties with regards to the desired level of standardization for FF-ICE and the declared scope of FIXM as outlined in the FIXM Strategy v1.1, chapter 3. Nevertheless, these samples represent a valid approach for encoding FF-ICE/R1 and/or ATS messages with FIXM-based content. Other message encodings having FIXM-based content satisfying the requirements outlined in chapter 2.2 may equally qualify as a valid usage of FIXM.*

## FIXM Samples of Uncorrelated ATS Messages

The initial set of samples contained FIXM representation of multiple messages of types Flight Plan (FPL), Change (CHG), Delay (DLA), and Cancel (CNL). These are uncorrelated messages, in the sense that they are not based on one or several particular flights, and each one is interpreted individually.

The input data for the samples was “sanitized” to not show actual flight IDs, airline codes, or pilot names. In addition, they do not contain any references to any other messages in the set.

## FIXM Samples of ATS Messages for a Full Flight Life-Cycle

To further expand the sample data, a complete life-cycle set of messages was constructed for a single flight (DAL18). This approach allowed the use of most of the ATS message types in a realistic scenario. This data contained message header input and additional subfields in ATS message field 3, *message number* and *reference data* (see Field Type 3, Appendix 3, ICAO Doc 4444), not seen in the earlier samples. These fields indicate which ATS computers generated and/or exchanged the messages, and point back to the relevant message when providing response or follow-up. The field mesg:referenceMessage contains the message uuid of its associated message, both of which are in the Messaging part of the FIXM XML.

The scenario which the DAL18 messages illustrate is as follows:

* DAL18 is a transatlantic flight flying from Detroit (DTW), Michigan to London Heathrow (EGGL).
* The flight departs from Cleveland Center (KZOB), flies through Canadian airspace (CZYZ, …), through oceanic airspace (CZQX, EGGX) and arrives at the London Air Traffic Control Center (EGTT).
* The input messages are in ICAO ATS format. They include both messages between the airline and ATS facilities, and between ATS facilities.
* The messages contain meta-data comprised of AFTN send and receiving addresses along with the day and time of the message. To eliminate redundancy, only one or two of each type of message is shown.

*Note: The first two lines of each message (starting with “FF”) are header data indicating message time and senders and recipients of the message. The body of the ATS message is contained within parentheses.*

An explanation of the message sequence follows (each message below corresponds to a sample FIXM message):

1. The operator, Delta Airlines, files the initial flight plan message (FPL) to KZOB, which is the departure center. This message is incorrect with regard to the flight’s route information (a “0” was placed in the unit of measure field for the airspeed in the Route Designator Field Type 15). A rejection message (rather than an ACK) is generated in response to this message because of the incorrect unit of measure. The rejection message is not shown. *Note: This is not something expected to happen in reality, but is an artificial case created to illustrate a logical reject of a message.*

**FF KZOBZZQA**  
**240040 KATLDALQ**  
**(FPL-DAL18-IS**  
**-B763/H-SDE2E3FGHIJ3J5M1RWXYZ/LB1D1**  
**-KDTW0314**  
**-0454F330 DCT PISTN DCT KARIT DCT YXI DCT YMW DCT YLQ DCT**  
**BAREE N505A RIKAL/M078F350 NATR SUNOT/M078F350 NATR**  
**KESIX/M078F350 DCT REVNU/N0441F370 DCT LIFFY UL975 WAL UY53**  
**NUGRA DCT**  
**-EGLL0715 EGKK**  
**-PBN/A1B1C1D1L1O1S1T1 DOF/170324 NAV/RNVD1E2A1 REG/N172DN**  
**EET/CZYZ0018 CZUL0051 CZQX0240 EGGX0438 EISN0606 EGTT0631**  
**SEL/EQAC CODE/A120F3 RMK/TCAS AGCS EQUIPPED NRP USA)**

1. Delta Airlines re-files the corrected flight plan message to KZOB. The previously incorrect unit of measure field has been entered properly.

**FF KZOBZZQA**  
**240049 KATLDALQ**

**(FPL-DAL18-IS**  
**-B763/H-SDE2E3FGHIJ3J5M1RWXYZ/LB1D1**  
**-KDTW0314**  
**-N0454F330 DCT PISTN DCT KARIT DCT YXI DCT YMW DCT YLQ DCT**  
**BAREE N505A RIKAL/M078F350 NATR SUNOT/M078F350 NATR**  
**KESIX/M078F350 DCT REVNU/N0441F370 DCT LIFFY UL975 WAL UY53**  
**NUGRA DCT**  
**-EGLL0715 EGKK**  
**-PBN/A1B1C1D1L1O1S1T1 DOF/170324 NAV/RNVD1E2A1 REG/N172DN**  
**EET/CZYZ0018 CZUL0051 CZQX0240 EGGX0438 EISN0606 EGTT0631**  
**SEL/EQAC CODE/A120F3 RMK/TCAS AGCS EQUIPPED NRP USA)**

1. Approximately 90 minutes prior to departure, Delta Airlines notifies Cleveland Center (KZOB) of a pre-departure delay (DLA) of some 40 minutes due to awaiting passengers from a connecting flight.

**FF KZOBZZQA**  
**240145 KATLDALQ**

(**DLA-DAL18-KDTW0354-EGLL-0)**

1. 69 minutes prior to the expected departure time, the flight plan is transmitted to CZYZ Center.

**FF CZYZZRIP**  
**240245 KZCCZQZX**  
**(FPLKZOB/CZYZ078-DAL18/A5773-IS**

**-B763/H-SDE2E3FGHIJ3J5M1RWXYZ/LB1D1**

**-KDTW0354**

**-N0454F330 DCT PISTN DCT KARIT DCT YXI DCT YMW DCT YLQ DCT BAREE N505A RIKAL/M078F350 NATR SUNOT/M078F350 NATR KESIX/M078F350 DCT REVNU/N0441F370 DCT LIFFY UL975 WAL UY53 NUGRA DCT**

\*\*-EGLL0715 EGKK-PBN/A1B1C1D1L1O1S1T1 DOF/170324 NAV/RNVD1A1E2 REG/N172DN EET/CZYZ0018 CZUL0051 CZQX0240 EGGX0438 EISN0606 EGTT0631 SEL/EQAC CODE/A120F3 RMK/TCAS AGCS EQUIPPED NRP USA)  
\*\*

1. CZYZ immediately responds with a logical acknowledgment message (LAM) from the ATS system.

**FF KZCCZQZX**

**240245 CZYZZRIP**  
**(LAMCZYZ/KZOB106KZOB/CZYZ078)**

1. When the flight departs, 17 minutes after the revised off block time, KZOB sends departure message to CZYZ.

**FF CZYZZRIP**  
**240411 KZCCZQZX**  
**(DEPKZOB/CZYZ080KZOB/CZYZ078-DAL18/A5773-KDTW0411-EGLL0715 EGKK-DOF/170324)**

1. CZYZ immediately responds with a logical acknowledgement message from the ATS system.

**FF KZCCZQZX**  
**240411 CZYZZRIP**  
**(LAMCZYZ/KZOB107KZOB/CZYZ080)**

1. When a flight plan is established, an estimate message (EST) is sent by KZOB to CZYZ notifying it of the expected transition for the flight.

**FF CZYZZRIP**  
**240412 KZCCZQZX**  
**(ESTKZOB/CZYZ156KZOB/CZYZ078-DAL18/A5773-KDTW-PISTN/0422F330-EGLL0715)**

1. CZYZ immediately responds with a logical acknowledgement message to the EST message.

**F KZCCZQZX**  
**240412 CZYZZRIP**

**(LAMCZYZ/KZOB184KZOB/CZYZ156)**

1. A change or modification message (CHG) is shown augmenting the time at the fix position for the coordination handling between KZOB and CZYZ Centers.

**FF CZYZZRIP**  
**240413 KZCCZQZX**  
**(CHGKZOB/CZYZ165KZOB/CZYZ078-DAL18/A5773-KDTW0411-EGLL-14/PISTN/0427F330)**

1. CZYZ immediately responds with a logical acknowledgement message to the CHG message.

**FF KZCCZQZX**  
**240413 CZYZZRIP**

**(LAMCZYZ/KZOB193KZOB/CZYZ165)**

1. The coordination message (CDN) between the KZOB and CZYZ ATS computer systems indicate that a handoff is accomplished at the boundary fix PISTN at an agreed upon time and flight level.

**FF CZYZZRIP**  
**240415 KZCCZQZX**  
**(CDNKZOB/CZYZ179CZYZ/KZOB078-DAL18/A5773-KDTW0411-EGLL-14/PISTN/0427F230F233A)**

1. CZYZ immediately responds with a logical acknowledgement message to the coordination handling.

**FF KZCCZQZX**  
**240416 CZYZZRIP**

**(LAMCZYZ/KZOB181KZOB/CZYZ179)**

1. An acceptance message (ACP) is then transmitted by CZYZ Center to ZOB for the coordination handling by Toronto Center.

**FF CZYZZRIP**  
**240415 KZCCZQZX**  
**(ACPCZYZ/KZOB180KZOB/CZYZ078-DAL18/A5773-KDTW0411-EGLL)**

1. CZYZ immediately responds with a logical acknowledgement message to the coordination handling.

**FF KZCCZQZX**  
**240416 CZYZZRIP**

**(LAMCZYZ/KZOB182KZOB/CZYZ180)**

1. Prior to the coordination exchange with the United Kingdom an amended current flight plan (CPL) is transmitted from CZQZ (Gander Operations Center) to EGGX (Shanwick Operations Center).

**FF EGGXZDZX**  
**240605 KZCCZQZX**  
**(CPLCZQZ/EGGX094-DAL18/A5773-IS-B763/H-SDE2E3FGHIJ3J5M1RWXYZ/LB1D1-KDTW0411-RAKIL/0641-N0454F330 DCT RIKAL/M078F350 NATR SUNOT/M078F350 NATR KESIX/M078F350 DCT REVNU/N0441F370 DCT LIFFY UL975 WAL UY53**  
**NUGRA DCT -EGLL0715 EGKK -PBN/A1B1C1D1L1O1S1T1 NAV/RNVD1E2A1 REG/N172DN EET/CZQX0240 EGGX0438 EISN0606 EGTT0631 SEL/EQAC CODE/A120F3)**

1. EGGX immediately responds with a logical acknowledgement message to the current flight plan.

**FF KZCCZQZX**  
**240605 EGGXZDZX**  
**(LAMCZQZ/EGGX033EGGX/CZQZ094)**

1. A coordination message (CDN) is transmitted for flight control at fix RAKIL between CZQX and EGGX Centers.

**FF EGGXZDZX**

**240616 KZCCZQZX**  
**(CDNCZQZ/EGGX037EGGX/CZQZ094-DAL18/A5773-KDTW-EGLL-14/RAKIL/0641F348F350A)**

1. EGGX immediately responds with a logical acknowledgement message to the coordination message.

**FF KZCCZQZX**  
**240616 EGGXZDZX**  
**(LAMEGGX/CZQZ040CZQZ/EGGX037)**

1. An acceptance message (ACP) is sent from EGGX Center to the CZQZ ATS computer for the coordination timing.

**FF KZCCZQZX**

**240617 EGGXZDZX**  
**(ACPEGGX/CZQZ039CZYZ/EGGX094-DAL18/A5773-KDTW0411-EGLL)**

1. CZYZ immediately responds with a logical acknowledgement message to the acceptance.

**FF EGGXZDZX**  
**240616** **KZCCZQZX**  
**(LAMCZQZ/EGGX041EGGX/EGGX039)**

1. The flight arrives per the actual flying time of 07 hours and 15 minutes and the arrival message (ARR) is sent from EGTT to KZOB.

**FF KZOBZZQA**  
**241127 EGTTZDZX**  
**(ARREGTT/KZOB245KZOB/EGTT094-DAL18/A5773-KDTW0411-EGLL1127)**

1. KZOB immediately responds with a logical acknowledgement message to the arrival message back to EGTT.

**FF EGTTZDZX**  
**241128 KZOBZZQA**  
**(LAMEGTT/KZOB256KZOB/EGTT245)**

## FIXM Samples of 4DT Data

A key component of FIXM is the ability to express a four-dimensional trajectory (4DT). Since no 4DTs are available in sample ATS messages, a small set of data representing 4DTs for three flights was converted to FIXM. This set was comprised of three “Creation Request” messages. These were used to create “Electronic Flight Plan” (eFPL) messages, which correspond to converted FPL messages with 4DT data merged into them. The 4DT data includes climb profile, descent profile, and desired trajectory. The trajectory data includes position, time, altitude, speed, and weather data associated with each trajectory element.

The FIXM data was generated using both the provided 4DT data and the route text, parsed into route elements, and merged together into a logical sequence. However, not every 4DT point corresponded to a route element, and not every route element had a corresponding 4DT data point. As a result, the FIXM trajectory contains a logical sequence of trajectory elements, with some of the trajectory elements containing associated 4DT data and some not. While this may not represent how an ideal 4DT message would appear if originally generated in FIXM format, it does produce a logical result that is FIXM compliant.

# How to create an Application Library

## Initial Download and Setup

To save time and reduce setup steps, it is recommended that you begin the process of creating your Application by downloading the FIXM Basic Message v1.0.0 full release[17].

1. Download the full release of FIXM Basic Message v1.0.0 from <https://fixm.aero/releases/Basic-Msg-1.0.0/FIXM_Basic_Message_v1.0.0_with_Core_v4.2.0_full_archive.zip>.
2. Unzip the full release and navigate to the “applications” directory under “schemas”.
3. Delete the “basicmessage” directory.
4. Navigate to the “uml” directory.
5. Rename the .eap file to something appropriate for your Application.

For this guided example, the .eap file should be renamed to “FIXM\_Example\_Message\_v1.0.0\_with\_Core\_v4.2.0.eap”. The starting directory structure of the example environment is shown below.

1. Open the model (the .eap file) with Sparx Enterprise Architect[18] (EA).
2. Click the arrow [] next to Applications in Project Browser to expand the Applications container.
3. Right click on the BasicMessage package in Project Browser and choose “Delete ‘<<XSDschema>> BasicMessage’”.
4. Click the Yes button to confirm the deletion.

## Create an Application Package

For this example, we will create a very simple Application that focuses on arrival and departure alerts. It will include Application header fields for message sender, recipient(s), timestamp, and alert type along with two templates, one for arrival alerts and one for departure alerts.

1. Right click on the Applications container and choose “Add a Package…”.
2. In the New Package dialogue box, change the *Name* field to something appropriate for your Application (in this example, “ExampleMessage”).
3. Select the *Create Diagram* radio button.
4. Then click OK.
5. This will bring up a New Diagram dialogue box. In the *Type* section:
   1. Choose “UML Structural” under *Select From*.
   2. Choose “Class” under *Diagram Types*.

Click OK to create the Application package and its associated diagram[19].

### Apply Schema Stereotype

In order to generate XSD schemas in Sparx EA, each package that corresponds to a schema file needs an XSDschema stereotype applied to it. To add an XSDschema stereotype to your package:

1. Double click (or right click and choose “Properties…”) on your Application package (here called “ExampleMessage”) in Project Browser to open the Package dialogue box.
2. In the newly opened Package dialogue box, click on the “…” box next to *Stereotype* (near the top right corner) to open the Stereotype dialogue box.
3. In the newly opened Stereotype dialogue box, select “XSDschema” and then click OK.
4. Then click OK in the Package dialogue box to apply the stereotype.

### Set Up Schema Properties

Once the stereotype is applied, you can configure your schema properties, setting up a number of XSD details such as schema file name, schema namespace, namespace prefixes, etc.

1. Once again, double click (or right click and choose “Properties…”) on your Application package in Project Browser. With the XSDschema stereotype applied, this should now open an XSD schema Properties dialogue box.
2. Fill in values for your schema properties as appropriate for your Application. In this example, we will use the following values:
3. *Target Namespace* set to: “<http://www.fixm.aero/app/example/1.0”>[20].
4. *Prefix* set to: “xmg”[21].
5. You must also fill in the *Schema File* field with the path to your application directory and an appropriate file name. In this example, the following *Schema File* entry will be used: .\schemas\applications\examplemessage\ExampleMessage.xsd.

IMPORTANT NOTE: Sparx EA will not automatically create any of the directories for the path to the file specified in the *Schema File* field. **They must be created outside of Sparx EA before generating the schemas.** FIXM is natively set up to generate its schema files under a directory named “schemas” that is located in the same directory as the Sparx EA file. To save time, rather than creating all of the directories needed by hand, you can instead:

1. Copy the “schemas” directory you modified back in [Initial Download and Setup](#initial-download-and-setup) into the directory where your Sparx EA file is located (in this example, the “uml” directory).
2. Under the “applications” directory beneath the “schemas” directory, add another directory with a name appropriate for your Application (here “examplemessage” was used).

In this example, this new “schemas” directory should be added to the “uml” directory and structured as shown below.

1. With all of these fields filled out, your XSD schema Properties dialogue box should look something like below. Click OK to save these settings.

### Add Schema Description and Tags

The final step in setting up your schema details is to add a description for your package and adjust the schema configuration settings that are controlled in Sparx EA via tagged values.

1. Once more, double click (or right click and choose “Properties…”) on your Application package in Project Browser and then click the UML button (near the top right corner). This will bring you back to your Package dialogue box.
2. In the central text box, fill in a description of your Application package. The text entered here will also show up as a documentation element in your schema file.
3. Next, click on the *Tags* tab (near the lower right hand corner) to add three tags used by Sparx EA when creating your schema files: attributeFormDefault, elementFormDefault, and version.
   1. FIXM standardly sets attributeFormDefault to “unqualified”.
   2. FIXM standardly sets elementFormDefault to “qualified”.
   3. Version should be set as appropriate for your Application (“1.0.0” in this example).

To add these tags, click on the third icon from the left [] in the *Tags* tab and fill in the *Tag* and *Value* fields similar to what is shown below.

1. When finished, your Package dialogue box will look something like below. Click OK to save these settings.

## Create Application Content

Now it is time to create any content needed for your Application. For this example, we will create two classes: a complex type for the message itself and an enumeration for the alert type. The message will have three attributes added to it and the enumeration will be linked to the message via an association. We will also add an association to link the message to Core’s Flight class (the root class for Core). Finally, we will create a root element for the Application.

1. Double click on your Application’s diagram (or right click it and choose “Open”) to get started.

### Create a ComplexType Class

We will start by creating a class with the XSDcomplexType stereotype for our message.

1. In Toolbox, click on “Complex Type” and then click anywhere in the diagram section of the screen[22]. This will open up an XSD complexType Properties dialogue box.
2. Add an appropriate *Name* and *Annotation* for your new class.
3. Then click OK.

You can now begin to add content to your class.

### Add an Attribute to a Class

For this example, we will add three attributes to this class: sender, recipient, and timestamp. Because each of these is of a type defined in Core’s Base package, using attributes is in line with how FIXM is standardly modeled.

1. Right click on the class in the diagram and choose “Features and Properties” and then “Attributes…”. This opens the Features dialogue box. In the Features dialogue box, you can add the attributes needed for your Application to your class.
2. Begin by clicking where it says “New Attribute…” under *Name* and filling in your new field’s name. In this example, we will start with “sender”.
3. Then hit the tab key or otherwise click out of the *Name* field.
4. You should now have a new attribute added to your class with a default *Type* of int and *Scope* of Private.
5. Next, click on the *Type* field, select the down arrow on the right side, and then click “Select Type…”.
6. This opens the Select Type dialogue box. Navigate to the appropriate class in the Base package you want to use, select it, and then click OK. In this example, that will be Fixm -> Core -> Base -> Organization -> PersonOrOrganization.
7. Next change the *Scope* field to “Public”.
8. Adjust the *Multiplicity* in the *Attribute* section as needed (in this example, it should be set to 0..1).
9. Then add a description for the field in the *Notes* section. When finished, your Features dialogue box will look something like below.
10. Repeat the steps above to continue adding as many attributes as desired. For this example, we will also add:
    1. A recipient field:
       1. *Name* set to: “recipient”.
       2. *Type* set to: “Fixm -> Core -> Base -> Organization -> PersonOrOrganization”.
       3. *Scope* set to: “Public”.
       4. *Multiplicity* set to: “0..2000”.
    2. A timestamp field:
       1. *Name* set to: “timestamp”.
       2. *Type* set to: “Fixm -> Core -> Base -> Types -> Time”.
       3. *Scope* set to: “Public”.
       4. *Multiplicity* set to: “0..1”.
11. When finished, click Close on the Features dialogue box. The class diagram should display the name, type, and multiplicity of each attribute added to the class.

### Create an Enumeration Class

Next, we will create an enumeration class to represent the type of the alert.

1. In Toolbox, click on “Enum” and then click anywhere in the diagram section of the screen. This will open up an XSD enumeration Properties dialogue box.
2. Add an appropriate *Name*, comma-delimited set of *Values*, and *Annotation* for your new class. For the *Type* field, FIXM standardly leaves this blank. When generating schemas, the physical model will still derive this enumeration from xs:string but leaving the field blank avoids displaying the string inheritance in the model diagrams.
3. Then click OK.
4. Right click on the enumeration class in the diagram and choose “Features and Properties” and then “Attributes…”. This opens the Features dialogue box.
5. In the Features dialogue box, you can add descriptions to each enumeration in the *Notes* tab. When finished, click OK.

### Add an Association Between Classes

When converted to XSD schemas, there is no difference between representing a field as an attribute of a class versus representing a field as an association between two classes. However, FIXM standardly only uses attributes for fields of a type defined in Core’s Base package. Any other classes you want to use as a field under a class should be attached to the class via an association.

In this example, we will use an association to create the alert type field needed for our message because our AlertType enumeration was not defined under the Base package.

1. Click on the class you wish to add your field to. You will see an upward arrow icon [] appear near the upper right hand corner of your class.
2. Click on this arrow and drag it over the class you wish to use as the type of your field. Then release the mouse button and choose “Association” from the list of options that pops up.
3. This will create an association between the two classes.
4. Double click (or right click and choose “Properties…”) on the association to open the Association Properties dialogue box.
5. Begin by clicking on the far right side of the *Direction* field under the *Connector Properties* table of the *Main* tab on the right hand side of the dialogue box. This should open a dropdown menu. From this menu, choose “Source -> Destination”.
6. Next, select “Role(s)” from the tree of options in the upper left hand corner. This brings up a new section of the Association Properties dialogue box split between setting up details for the source and target sides of your new association.
7. In the text/dropdown box directly under TARGET in the upper right hand corner, type in the name you would like to use for the field represented by this association (“type” in this example).
8. In the next text box down, fill in a short description for this field.
9. Finally, choose an appropriate *Multiplicity* for this field (“0..1” in this example).
10. When finished, your Association Properties dialogue box will look something like below. Click OK to save these settings.

In this example, we will also add an association between ExampleMessage and the Flight class (the root class of FIXM Core).

1. To accomplish this, first navigate to the correct package in Project Browser (Fixm -> Core -> Flight -> FlightData) and then select the “<<XSDcomplexType>>Flight” class.
2. Now click and drag the Flight class from Project Browser to the ExampleMessage diagram. When you release the mouse button, a dialogue box will appear asking you how you would like to paste the class into the diagram. Select “Link” under the *Drop As* dropdown box and then click Okay.
3. Next, follow the same steps detailed above to create an association from the ExampleMessage class to the Flight class. When finished, your diagram should look something like below.

The final step in creating associations is to add “position” tags. Sparx EA will automatically alphabetize class attributes when creating schemas from the logical model but associations need to be handled explicitly. Once all associations have been created for a class, follow the steps below to make sure the fields derived from associations are properly ordered in the physical model.

1. In the *Start* ribbon at the top of the screen, select “Tagged Values” from the *Windows* section to open the Tagged Value window. Having this window open provides a shortcut to accessing tagged values and is very useful when adding position tags.
2. Next, determine the correct ordering for each association originating from your class. For this example, the ExampleMessage class will have three attributes and two associations. Ordering the group of fields as a whole alphabetically gives you: flight, recipient, sender, timestamp, type. The position of the field in this alphabetical list gives you the value (starting with “1” for the first field) needed when adding a position tag. So, for this example, “flight” will need a position tag with a value of “1” and “type” will need a position tag with a value of “5”[23].
3. Now, click on each association in turn. When clicked on, you will see all the tagged values associated with whatever you click on show up in the Tagged Values window. For these associations, this window will initially look like the image below.
4. Click on *Connector Source* and then click on the third icon from the left [] to add a new tagged value. The *tag* field should be set to “position” and the *value* field to the correct numeric value as determined above. In this example, the flight association, connecting ExampleMessage to Flight, should be given the following tag:

The type association, connecting ExampleMessage to AlertType, should be set up as below:

### Add a Root Element

Each XML document has exactly one single root element. It encloses all the other elements and is therefore the sole parent element to all the other elements. Applications create one or more elements that serve as an entry point to the model and can therefore be used as root elements. For this example, we will create a single such entry point named “ExampleMessage”.

1. In Toolbox, click on “Element” and then click anywhere in the diagram section of the screen. This will open up an XSD element Properties dialogue box.
2. Add an appropriate *Name* (in this example, “ExampleMessage”) and *Annotation* (in this example, “The ExampleMessage element is an entry point to the Example Message application.”) for your Application.
3. Next click on the “…” icon to the right of the *Type* field. This opens the Select Classifier dialogue box. In the *Browse* tab, navigate to and click on the class you would like to use as the entry point for your Application. In this example, Fixm -> Applications -> ExampleMessage -> MessageType. Click OK.
4. The *Type* field should now be updated to your selection. Click OK to create your root element. Note the generalization link formed between the element and the class it is based on.

Repeat the steps above to add elements for each intended entry point into your Application.

1. Continue adding as many classes, attributes, associations, and elements as needed for your particular Application. When finished, right click on the name at the top of the diagram and click “Save Changes to ‘[diagram name]’”.

The Application presented here is simple but keep in mind that the steps detailed in this guide can be used to create as many packages and as much content as needed to capture all the structure, headers, and metadata needed for your particular exchange.

## Create Templates

FIXM is a general-purpose standard meant to facilitate the exchange of any and all flight data. To accomplish this, it must be extremely flexible in terms of required data content. However, individual users of the FIXM standard may want to lockdown the expected format of their data to reflect the content requirements of their particular message exchanges. Message templates, created via XSD restrictions, is the current method recommended by FIXM to accomplish this goal. For this example, we will create two templates to illustrate the process: departure alerts and arrival alerts.

### Create an Overall Template Container

The steps for creating this package are largely identical to those outlined in [Create an Application Package](#create-an-application-package) above. Key differences are noted below.

1. Begin by right clicking on your Application package and choose “Add a Package…”.
2. In the New Package dialogue box, change the *Name* field to something appropriate to your templates (here we used “ExampleTemplates”), select the *Package Only* radio button, and then click OK.
3. Apply a schema stereotype and begin setting up schema properties as outlined in [Apply Schema Stereotype](#apply-schema-stereotype) and [Set Up Schema Properties](#set-up-schema-properties) above using the same *Target Namespace* and *Prefix* as was used for your Application package[24].
4. For the *Schema File* field, choose something appropriate for your templates (in this example, “.\schemas\applications\examplemessage\exampletemplates\ExampleTemplates.xsd”). Please note, you will want to create a new subdirectory under your Application’s directory to hold the templates (in this example, “exampletemplates” should be created below the “examplemessage” directory).
5. In some cases, Sparx EA will automatically add additional namespace prefixes for packages that need to be imported, but that does not happen in this instance. Address this by manually adding the namespaces of any packages you will import. This step is accomplished by:
6. Clicking “New” to open a Namespace Details dialogue box.
7. Filling in appropriate details for each namespace to be included.
8. Then clicking OK.

In this example, our templates will import Base and Flight from Core. See below for examples showing how each Namespace Details dialogue box should be filled in.

1. At this point, your XSD schema Properties dialogue box should look something like below. Click OK to save these settings.
2. Finally, follow the steps outlined in [Add Schema Description and Tags](#add-schema-description-and-tags) to complete the setup of your template container.

### Create a Container for each Type of Message Template

With the overall template container built, you must now create individual containers for each message template you wish to include in your exchange. In this example, we will create two such containers: ArrivalAlert and DepartureAlert. Due to their similarity in structure to the overall container, we can use a shortcut to build them rather than repeating all the steps from the previous section.

1. Right click on your newly created template container in Project Browser and choose “Copy / Paste” and then “Copy to Clipboard” and then “Full Structure of Duplication”.
2. Once again, right click on your template container in Project Browser and this time choose “Copy / Paste” and then “Paste Package from Clipboard”.
3. You should see a new copy of your container show up in Project Browser below the original.
4. Repeat the above “Paste Package from Clipboard” step for each message template you would like to create[25]. In this example, we will create two message templates. When finished, your Project Browser should display something similar to below.
5. Double click (or right click and choose “Properties…”) on each individual message template container and modify the *Schema Name* and *Schema File* fields to be something appropriate for your templates. Outside of Sparx EA, you will also want to create individual directories below the overall template directory to hold each message template. For this example, the following values should be used:
   1. ArrivalAlert template
      1. *Schema Name* set to: “ArrivalAlert”
      2. *Schema File* set to:
      * “.\schemas\applications\examplemessage\exampletemplates\arrivalalert\ArrivalAlert.xsd”
   2. DepartureAlert template
      1. *Schema Name* set to: “DepartureAlert”
      2. *Schema File* set to:
      * “.\schemas\applications\examplemessage\exampletemplates\departurealert\DepartureAlert.xsd”

And the schemas directory structure should be modified as follows:

Don’t forget to modify your schema descriptions to something appropriate as well. In this example, ArrivalAlert should use “An example Arrival Alert template.” and DepartureAlert should use “An example Departure Alert template”.

1. When finished, your Project Browser should display something similar to below.

### Create a Message Template

The individual message templates themselves recreate the structure of the portions of the models that they draw fields from and reuse the same namespaces, prefixes, and package names. They use different diagram names, schema file names, schema descriptions, and, of course, the content itself is modified using XSD restrictions.

The convention used so far in FIXM when naming template material is to prefix the existing names with text to identify the Application followed by text to identify the particular message. So, for this example, fields under ArrivalAlert will be prefixed with “ExampleAA\_” while fields under DepartureAlert will be prefixed with “ExampleDA\_”.

#### Create Template Packages

One template package will need to be created for each Core or Application package you wish to restrict. So, for this example, the ArrivalAlert template will need to create an ExampleMessage package, a Flight package, an Arrival package, and a FlightData package because these are the packages that will be restricted to create an ArrivalAlert message.

These template packages should replicate all the settings of the package they correspond to except for their diagram’s *Name* field (if they have an associated diagram – some packages won’t), the schema description, and the *Schema File* field of their schema properties.

1. Follow the steps outlined in [Create an Application Package](#create-an-application-package) to recreate, under your message template container, the packages you wish to restrict in your template. All names, properties, descriptions, tags, etc., should be the same[26] except:
   1. If your package has an associated diagram, change the diagram *Name* to include your template’s prefix. While not strictly necessary, this helps prevent confusing template diagrams with the originals.
   2. Change the *Schema File* field to use a path corresponding to your message template and a filename beginning with your template’s prefix. Template schema files should be located under the directory of your message container but their paths should otherwise include creating any intermediate directories, etc., so that they are structured similar to the paths to the files they are restricting.
   3. Indicate that the schema is a template in the description text.

For this example, we will first create a template package corresponding to ExampleMessage under ArrivalAlert. When creating the package, we will change its diagram name to be “ExampleAA\_ExampleMessage”.

When configuring the package, use all the same values as ExampleMessage except change the schema description to “An example Arrival Alert message template for the ExampleMessage package.” and the *Schema File* field to “.\schemas\applications\examplemessage\exampletemplates\arrivalalert\examplemessage\ExampleAA\_ExampleMessage.xsd”.

The same steps will be used to create template packages for Core’s Flight package and its Arrival and FlightData sub-packages. When finished, Package Browser should appear as follows.

And the directory structure for the Application should be organized as shown below.

#### Create Template Content

The root mechanism for creating template content is making use of the “XSDrestriction” stereotype built into Sparx EA. When applied to a Generalization connector between two classes, this results in a schema restriction on the simpleType or complexContent definition, depending on the nature of the classes, of the derived class in the physical model. Put simply, these XSD restrictions allow you to create derived classes with fewer fields and/or more restricted content than the classes they are derived from.

We will illustrate in more detail how these restrictions are applied by looking first at how to constructe the ArrivalAlert template. When finished, this template should provide the following fields:

* 1 ExampleMessage with:
  + 1 sender
  + 1..99 recipients
  + 1 timestamp
  + 1 type (set to ARRIVAL)
  + 1 flight with:
    - 1 arrival with:
      * 1 actualTimeOfArrival
      * 1 arrivalAerodrome
    - 1 gufi
    - 1 flightIdentification with:
      * 1 aircraftIdentification
    - 0..1 flightType

When constructing a template, we found it easiest to start at the leaf (outermost) fields and work backwards to the root (innermost) fields.

##### Create a Restricted Class (Complex Type)

1. Double click on your message template’s diagram (or right click it and choose “Open”) to get started. In this example, we began with the “ExampeAA\_Arrival” diagram.
2. Locate the class you wish to restrict in Project Browser and then left click and drag the class into your diagram (in this example, Fixm -> Core -> Flight -> Arrival -> <<XSDcomplexType>>Arrival). Select “Link” under the *Drop As* dropdown box and then click Okay.
3. Create a new class of the same type as the class you wish to restrict. In this example, that would be a complexType class, created as outlined in [Create a ComplexType Class](#create-a-complextype-class). The *Name* and *Annotation* should be the same as the class you are restricting except the *Name* should begin with your template’s prefix.
4. Click on your new restricted class in the diagram. You will see an upward arrow icon [] appear near the upper right hand corner of the class. Click on this arrow and drag it over the class you are restricting. Then release the mouse button and choose “Generalization” from the list of options that pops up.
5. Double click (or right click and choose “Properties…”) on the generalization connector to open the Generalization Properties dialogue box.
6. Click on the far right side of the *Stereotype* field under the *Connector Properties* table of the *Main* tab on the right hand side of the dialogue box. This should open a Stereotype dialogue box. Check the box next to “XSDrestriction” and click Okay.
7. Then click Okay in the Generalization Properties dialogue box to apply the stereotype.

Any class attributes and associations that resolve to XML elements in the physical model that are not included in a restricted class will be removed[27]. At this point, our example restricted class is completely empty. The next step is to add back in the attributes and associations you wish to retain.

##### Set Up Restricted Class Attributes (Complex Type)

We will begin with class attributes. If desired, attributes can be added by following the steps listed in [Add an Attribute to a Class](#add-an-attribute-to-a-class), being sure to replicate the values used by the attributes in the class you are restricting. However, the steps listed below can be used as a shortcut that should make adding the attributes both easier and less error prone.

1. Once again, locate the class you wish to restrict in Project Browser (in this example, Fixm -> Core -> Flight -> Arrival -> <<XSDcomplexType>>Arrival). Click on the arrow [] to the left of the class to display its associated attributes.
2. Control-left-click on each attribute you wish to retain in your restriction. In this example, “actualTimeOfArrival” and “arrivalAerodrome”.
3. Now drag the selected attributes onto your restricted class in the diagram. In the example, the attributes will be dragged onto the ExampleAA\_Arrival class in the ExampleAA\_Arrival diagram. This will create exact replicas of the attributes within your restricted class.

Next, adjust the *Multiplicity* of your attributes to suit your template’s needs.

1. Right click on the restricted class in the diagram and choose “Features and Properties” and then “Attributes…”. Select the attribute you wish to adjust and then click on *Multiplicity* in the *Attribute* section to change its value. In this example, both “actualTimeOfArrival” and “arrivalAerodrome” should be given an upper and lower bound of “1”.

Most attributes in FIXM Core are marked as nillable. This is done via adding a tag to the field with *Tag* set to “nillable” and *Value* set to “true”.

1. If you do not wish attributes in your template to be nillable, navigate to the *Tagged Values* tab, and then erase the nillable tag by clicking on it and then clicking the fifth icon from the left []. In this example, the nillable tags should be erased from all attributes with a multiplicity lower bound of “1” or higher.
2. Click Close when finished. You should see the new multiplicities reflected in the restricted class diagram.

##### Set Up Restricted Class Associations

Next, we will focus on setting up associations. Like attributes, any associations that resolve to XML elements not added back to your restricted class will be removed. For example, the “reclearanceInFlight” association attached to Core’s Arrival class (shown below) was removed from ExampleAA\_Arrival because the association was never recreated.

To find an example of retained associations, let us move on to the restricted FlightData package.

Following the steps outlined so far in [Create Template Content](#create-template-content), create restricted versions of the “Flight” and “FlightIdentification” classes from Fixm -> Core -> Flight -> FlightData in the ExampleAA\_FlightData diagram. For our new “ExampleAA\_Flight” class, the “gufi” attribute should be retained and made required (*Multiplicity* of 1..1). For “ExampleAA\_FlightIdentification”, the “aircraftIdentification” attribute should be retained and made required. Below are examples of how Project Brower and the diagram will appear after this is done.

Much like “reclearanceInFlight” from the Arrival package, all of the associations attached to the Flight class in Core have been implicitly restricted away for ExampleAA\_Flight. To retain them, they must be added back in by hand.

In this example, the first association we will add back in is “flightType”.

1. If the class you wish to add an association to is not already present in the diagram, locate it in Project Brower and drag it into the diagram. In this example, “Fixm -> Core -> Flight -> FlightData -> TypeOfFlight” should be added to the ExampleAA\_FlightData diagram.
2. Use the steps detailed in [Add an Association Between Classes](#add-an-association-between-classes) to create the association desired for your restricted class. Like with class attributes, you will want to reuse all of the same values as the original association with the possible exception of multiplicity and removing the nillable tag. You will also likely need to modify the “position” tag to make sure your elements are ordered correctly in the physical model. In this example, use all of the same values used in the original “flightType” association except position (which will end up being set to “3” in the final version of this template).

Sparx EA automatically displays any associations between classes you add to a diagram. Above you will notice this for the existing associations “flightType” and “flightIdentification”. To improve the readability of your diagrams, it is recommended you hide (not delete[28]) these associations.

1. Right click on the any existing associations you do not want to show in your diagram and choose “Visibility” and then “Hide Connector”. In this example, the pre-existing “flightType” and “flightIdentification” associations were hidden.

##### Use a Restricted Class to Enforce the Use of Another Restricted Class

The next association that needs to be created for the example will be used to add back in the “flightIdentification” field to the restricted ExampleAA\_Flight class. However, we only want to allow the use of the restricted ExampleAA\_FlightIdentification class, not the original unrestricted FlightIdentification class from Core.

While doing so is mechanically no different than the steps outlined above in [Set Up Restricted Class Associations](#set-up-restricted-class-associations), this chaining together of restricted classes (ultimately all the way back to the root class of your Application) is the means by which templates are formed and enforced[29].

Below are a series of screenshots capturing key steps taken along the way to completing the restricted FlightData package. Each of these steps involves enforcing the use of a restricted class.

With the FlightData package complete, the only package left to finish for the ArrivalAlert template is ExampleMessage.

##### Create a Restricted Class (Enumeration)

When creating the final ArrivalAlert restricted package (ExampleMessage), we run into a new use of XSD restriction: limiting an enumeration to only allow a specific value.

1. Similar to [Create a Restricted Class (Complex Type)](#create-a-restricted-class-complex-type), begin by dragging the enumeration you wish to restrict into your diagram. In this example, “AlertType” from Fixm -> Applications -> ExampleMessage should be added to the “ExampleAA\_ExampleMessage” diagram.
2. Next, add a new enumeration to the diagram as outlined in [Create an Enumeration Class](#create-an-enumeration-class). However, the *Name* field should begin with your message template’s prefix (in this example, “MessageAA\_”) followed by the same name as the enumeration you are restricting, the *Annotation* should be the same as the enumeration you are restricting, and the *Values* should only include the value you wish to enforce in this template (in this example, “ARRIVAL”).
3. Next click on the “…” icon to the right of the *Type* field. This opens the Select Classifier dialogue box. In the *Browse* tab, navigate to and click on the class you would like your restriction to be derived from. In this example, Fixm -> Applications -> ExampleMessage -> AlertType. Click OK.
4. You will see the selected class show up in the *Type* field. Click OK to create the enumeration.
5. Finally, as done in [Create a Restricted Class (Complex Type)](#create-a-restricted-class-complex-type), apply the XSDrestriction stereotype to your generalization connector to finish constructing your restricted enumeration class.

The rest of the restricted ExampleMessage package should be created using the techniques covered above in [Create Template Content](#create-template-content). Below are a series of screenshots capturing key steps taken along the way.

At this point in the example Application, the ArrivalAlert message template should be complete and it is time to create the DepartureAlert template. The intended content of the DepartureAlert template is very close to the structure and content of ArrivalAlert:

* 1 ExampleMessage with:
  + 1 sender
  + 1..99 recipients
  + 1 timestamp
  + 1 type (set to DEPARTURE)
  + 1 flight with:
    - 1 departure with:
      * 1 actualTimeOfDeparture
      * 1 aerodrome
    - 1 gufi
    - 1 flightIdentification with:
      * 1 aircraftIdentification
    - 0..1 flightType

Only the value of the “type” field and the replacement of “arrival” with “departure” (and associated sub-fields) differ between the two, making this a good candidate for copying and pasting template content (see [Copying and Pasting a Template Package](#copying-and-pasting-a-template-package) below).

Before employing this technique, the portions of DepartureAlert that are not a good candidate for copy and paste (specifically the Flight package and the Departure package) should be created as described in [Create Template Content](#create-template-content) above. Do so now. Below are screenshots of showing how Project Browser and the restricted Departure package diagram should look after these packages have been completed.

### Copying and Pasting a Template Package

Copying and pasting content in EA can be dangerous if proper care is not taken. It is easy to accidentally create hidden relationships or cause other unexpected issues. However, it can also be very helpful in terms of saving effort and avoiding errors that sometimes creep in when creating content by hand. The restricted FlightData and ExampleMessage packages from the ArrivalAlert template are good candidates for using this technique to create corresponding packages under DepartureAlert. In this example, we will begin with FlightData.

1. Right click on package you would like to copy in Project Browser and choose “Copy / Paste” and then “Copy to Clipboard” and then “Full Structure of Duplication”.
2. Next, right click on the package under which you want to paste your copied package and choose “Copy / Paste” and then “Paste Package from Clipboard”. In this example, that is the “Flight” package under “DepartureAlert”.

At this point, your copied package should be replicated in the new location. You will now need to go through it and adjust settings as needed for this version of the package.

1. Double click (or right click and choose “Properties…”) on your copied package in Project Browser to access and adjust the schema properties. In this example, only the *Schema Location* field needs to be adjusted. It should be modified to use: “.\schemas\applications\examplemessage\exampletemplates\departurealert\flight\flightdata\ExampleDA\_FlightData.xsd”[30]. Click OK to save the new settings.
2. Again, double click (or right click and choose “Properties…”) on your copied package in Project Browser and this time click the UML button in the upper right hand corner. In the central text box, adjust the schema description as needed. In this example, the text should be changed to read “An example Departure Alert message template for the FlightData package.”. Click OK to save the change.
3. Right click and choose “Properties…” on your diagram name in Project Brower. In this example, that will be “ExampleAA\_FlightData” within the DepartureAlert template.
4. This opens the Class Diagram dialogue box. Change the *Name* field to something appropriate for your package. In this example, “ExampleDA\_FlightData”.
5. In Project Browser, delete any unwanted classes from your package by right clicking on them and choosing “Delete ‘[class name]’”. In this example, no classes need to be deleted.
6. In Project Brower, double click (or right click and choose “Properties…”) on each class in your package and update the *Name* field as needed. In this example, each *Name* should be modified to use a prefix of “ExampleDA\_” rather than “ExampleAA\_”.
7. Now review your class diagram by double clicking on its name (or right click it and choose “Open”).
8. Note any connectors that are no longer desired. Delete these by right clicking on them and choosing “Delete Connector”[31]. In this example, the “arrival” connector between ExampleDA\_Flight and ExampleAA\_Arrival is no longer desired and should be removed.

When prompted, select the “Delete the connector from the model” radio button and then click OK.

1. Note any orphaned classes left behind when the unwanted connectors have been removed. Click on each in turn and press the delete key (or right click on them and choose “Delete ‘[class name]’”). Unlike connectors, this will only remove the class from the diagram, not erase it entirely from the model. In this example, the ExampleAA\_Arrival class should be deleted from the diagram.
2. Next, use the techniques outlined in [Create Template Content](#create-template-content) to finish adjusting your package with any more changes needed. In this example, that involves creating a “departure” association between ExampleDA\_Flight and ExampleDA\_Departure (defined in the already created restricted Departure package under DepartureAlert). Don’t forget to adjust position tags as needed to ensure correct ordering of elements in the physical model.

As a final check when performing any copy and paste in Sparx EA, you will want to review the relationships of each class you copied to ensure no undesired connections exist.

1. In the *Start* ribbon at the top of the screen, select “Relationships” from the *Windows* section to open the Relationships window.
2. Click on each class in your package and review the displayed relationships for any unexpected connections. In this example, the Relationships window will reveal an unwanted connection between ExampleDA\_Flight and ExampleAA\_ExampleMessage that resulted from the copy and paste!
3. Right click and choose “Delete Connection” for any unwanted relationships. Click Yes when asked to confirm the deletion.

The steps outlined above should be repeated for the ExampleMessage package to complete the DepartureAlert template. Below are screenshots of how Project Browser and the restricted ExampleMessage package diagram should appear after this package is completed.

The templates created for this example were relatively simple and contained very few fields when compared to FIXM’s entire structure. However, the techniques outlined in [Create Templates](#create-templates) were the same ones employed to create the much larger set of FF-ICE Message templates and should provide the framework needed to create larger, more complicated templates as needed.

### Create the Includes Package

The final step in creating Templates is to add an “Includes” package and appropriate sub-packages. FIXM makes use of “package-wide include files” to increase its usability with a number of XML tools. The packages contained under the Includes package facilitate this and will be modified by the post-processing script to contain the needed content (see [Set Up and Use Package-Wide Include Files](#X6da751ea0cf07df598b956f29ffb35c931bf943) below for more details).

1. Right click on your overall templates container (in this example, the “ExampleTemplates” package) and choose “Add a Package…”.
2. In the New Package dialogue box, change the *Name* field to “Includes”.
3. Select the “Package Only” radio button and click OK.
4. For each namespace your templates both import and restrict (in this example, only “<http://www.fixm.aero/flight/4.2”>), create one sub-package under Includes. Like Includes itself, these sub-packages will be “Package Only”. Choose a *Name* appropriate to both your templates and the schemas they will import. In this example, one sub-package will be created and it should be named “ExampleFlight”.
5. Follow the steps detailed in [Apply Schema Stereotype](#apply-schema-stereotype), [Set Up Schema Properties](#set-up-schema-properties), and [Add Schema Description and Tags](#add-schema-description-and-tags) to configure this package. The *Target Namespace* and *Prefix* fields should match those of the schemas you will be importing, as should the “version” tag. The *Schema File* field should be set to something appropriate for your templates. In this example, use:
6. *Target Namespace* set to: “<http://www.fixm.aero/flight/4.2”>.
7. *Prefix* set to: “fx”.
8. *Schema File* set to: “.\schemas\applications\examplemessage\exampletemplates\ExampleFlight.xsd”.

With the Includes package in place, the example Applications Library is complete. Below are screenshots showing the final composition of Project Browser and the schemas directory structure.

## Generate the Application Schemas

When the Application is complete, it is time to generate the physical model. The general steps for generating schemas from the logical model are outlined in APPENDIX C.

1. Begin the process by right clicking on your Application package and selecting “Code Engineering” and then “Generate XML Schema…”.
2. Then follow the same steps listed in APPENDIX C to set up and generate your schemas.

## Post-Process the Application Schemas

After the schemas have been generated they will need to be post-processed. The post-processing script referenced in APPENDIX C will need to be modified in in order to accommodate your Application.

### Include Application Namespace Prefix

FIXM standardly adds "Type" to the end of the name of all of the types declared in its schemas. In order to avoid doing this to built-in XSD types, the post-processing script uses namespace prefixes to determine which types it should modify. To accommodate this, your Application’s namespace prefix needs to be added to the script for proper processing.

1. First, locate the else clause near the end of the script that contains the comment:

# Add "Type" to the end of all FIXM type names (both declaration and use).

1. In this section, you will see a series of search and replace commands. The bottom three of these have a portion that lists out all the namespace prefix options used by the FIXM schemas as part of the search field. These lines need to be modified to include the namespace prefix used by your Application.

For the example created here, this change should be implemented as shown below (modification in red):

$line =~ s/<xs:(restriction|extension).\*base="(fb|fx|xmg):[^"]\*/$&Type/;

$line =~ s/<xs:(attribute|element).\*type="(fb|fx|xmg):[^"]\*/$&Type/;

$line =~ s/<xs:list.\*itemType="(fb|fx|xmg):[^"]\*/$&Type/;

### Set Up and Use Package-Wide Include Files

When parsing schemas, some XML tools only recognize and process the first “import” element encountered for any given namespace. If a schema were split across two or more files and only one file were imported due to this behavior, this would cause obvious problems, as half or more of your schema could be ignored as a result.

FIXM combats this issue by creating schema files that contain “include” elements for every single file in a given namespace and then uses these wide reaching schema files anytime it uses an “import” element. This way, FIXM only needs one “import” to be processed in order to deliver the whole schema as intended. FIXM also creates schema files with a mix of imports and includes to serve as entry points into the standard, serving the same purpose as the “include” only schema files but also gathering together everything that will be needed in one place for tools set up to only process single file schemas.

The term FIXM generally uses when referring to these sorts of files is “package-wide include files”, or just “include files”, and FIXM makes broad use of these files to increase its usability with a number of XML tools. The schema files that will be used for this are generated by Sparx EA, but it is the post-processing script that creates the actual import and/or include elements within these files. Depending on the complexity and structure of your Application, you may need many of these.

1. Directly above the section of the post-processing script referenced in the previous step (for adding “Type” to type names), locate the series of elsif clauses with conditionals along the lines of:

elsif ($schema =~ /\/[filename].xsd/)

1. At the end of this section, directly after the closing brace of the last elsif clause in the series, add your own elsif clauses for setting up your Application’s include files. The conditionals need to be set up to match your include files’ names.
2. In the body of the clause, add import and/or include elements for the schema files as needed.

The following two subsections discuss how to these clauses should be constructed in more detail.

#### Include Only Files

Anytime your schema is split across multiple files, you will need to create an include file to gather all the pieces together in a single file. For the example Application created here, this happens for the schemas using the “<http://www.fixm.aero/flight/4.2”> namespace in three locations. Once each under the two message templates (ArrivalAlert and DepartureAlert) and once for the “Includes” file used to gather together all instances of the flight namespace used throughout the templates in one place. Below are the elsif clauses that can be used to add the content needed for these three include only files:

# Add package-wide includes to ExampleAA\_Flight.xsd

elsif ($schema =~ /\/ExampleAA\_Flight.xsd/)

{

$line = "\t<xs:include schemaLocation=\"./arrival/ExampleAA\_Arrival.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./flightdata/ExampleAA\_FlightData.xsd\"/>\n" .

$line;

}

# Add package-wide includes to ExampleDA\_Flight.xsd

elsif ($schema =~ /\/ExampleDA\_Flight.xsd/)

{

$line = "\t<xs:include schemaLocation=\"./departure/ExampleDA\_Departure.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./flightdata/ExampleDA\_FlightData.xsd\"/>\n" .

$line;

}

# Add package-wide includes to ExampleFlight.xsd

elsif ($schema =~ /\/ExampleFlight.xsd/)

{

$line = "\t<xs:include schemaLocation=\"./arrivalalert/flight/ExampleAA\_Flight.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./departurealert/flight/ExampleDA\_Flight.xsd\"/>\n" .

$line;

}

#### Import and Include Files

Import and include files serve a similar purpose to the include only files mentioned above but also act as general entry points, gathering together everything that would be needed to process a certain part of FIXM in a single file. There are three such entry points in the example Application created here. One each for the two message templates (ArrivalAlert and DepartureAlert) as well as one for the entry point used to bind these two templates together under the Application (ExampleTemplates). Below are the elsif clauses that can be used to create the content needed for these three import and include files:

# Add package imports/includes to ArrivalAlert.xsd

elsif ($schema =~ /\/ArrivalAlert.xsd/)

{

$line = "\t<xs:import namespace=\"<http://www.fixm.aero/flight/4.2>\" schemaLocation=\"./flight/ExampleAA\_Flight.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./examplemessage/ExampleAA\_ExampleMessage.xsd\"/>\n" .

$line;

}

# Add package imports/includes to DepartureAlert.xsd

elsif ($schema =~ /\/DepartureAlert.xsd/)

{

$line = "\t<xs:import namespace=\"<http://www.fixm.aero/flight/4.2>\" schemaLocation=\"./flight/ExampleDA\_Flight.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./examplemessage/ExampleDA\_ExampleMessage.xsd\"/>\n" .

$line;

}

# Add package imports/includes to ExampleTemplates.xsd

elsif ($schema =~ /\/ExampleTemplates.xsd/)

{

$line = "\t<xs:import namespace=\"<http://www.fixm.aero/base/4.2>\" schemaLocation=\"../../../core/base/Base.xsd\"/>\n" .

"\t<xs:import namespace=\"<http://www.fixm.aero/flight/4.2>\" schemaLocation=\"./ExampleFlight.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./arrivalalert/examplemessage/ExampleAA\_ExampleMessage.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./departurealert/examplemessage/ExampleDA\_ExampleMessage.xsd\"/>\n" .

$line;

}

### Modify Import Elements to Use Include Files

You will also need to replace “import” elements generated by Sparx EA with elements that make use of the files you just set up. The post-processing script handles this task as well, replacing all import elements for a given namespace with a single import using the include file. In the example Application created here, only imports using the “<http://www.fixm.aero/flight/4.2”> namespace need to be modified.

1. Locate the else clause near the beginning of the script that contains the comment:

# Replace Flight package imports with a single import of appropriate Flight.xsd

1. Locate the series of elsif clauses in this section with conditionals along the lines of:

elsif ($schema =~ /\/[template directory name]\//)

1. At the end of this section, directly between the final elsif and the closing else clause, add your own elsif clauses for replacing existing import statements with ones that make use of the broader include files.

In this example, these new clauses should be added to address this issue:

elsif ($schema =~ /\/arrivalalert\//)

{

$line = $match . "ExampleAA\_Flight.xsd\"/>\n";

}

elsif ($schema =~ /\/departurealert\//)

{

$line = $match . "ExampleDA\_Flight.xsd\"/>\n";

}

Finally, as noted in APPENDIX C, be careful not to run your modified post-processing script against any schema files that have already been post-processed. To avoid this issue, either regenerate the Core schemas you need before post-processing the entire schemas directory or only run the script exclusively against the generated Applications schemas.

## Sample XML

Below is a sample XML document that validates against the ExampleMessage Application described in this Appendix (using the DepartureAlert template):

<?xml version="1.0" encoding="UTF-8"?>

<xmg:ExampleMessage xsi:type="xmg:ExampleDA\_ExampleMessageType" xmlns:xmg="<http://www.fixm.aero/app/example/1.0"> xmlns:fb="<http://www.fixm.aero/base/4.2"> xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance">>

<xmg:flight>

<fx:departure>

<fx:actualTimeOfDeparture>2020-01-15T00:00:00Z</fx:actualTimeOfDeparture>

<fx:aerodrome>

<fb:locationIndicator>KBOS</fb:locationIndicator>

</fx:aerodrome>

</fx:departure>

<fx:flightIdentification>

<fx:aircraftIdentification>ABC1234</fx:aircraftIdentification>

</fx:flightIdentification>

<fx:gufi codeSpace="urn:uuid">3e7f6a63-6c3b-4f0f-844b-4b84338ed103</fx:gufi>

</xmg:flight>

<xmg:recipient>

<fb:name>RECIPIENT 1</fb:name>

</xmg:recipient>

<xmg:recipient>

<fb:name>RECIPIENT 2</fb:name>

</xmg:recipient>

<xmg:sender>

<fb:name>SENDER</fb:name>

</xmg:sender>

<xmg:timestamp>2020-01-15T00:00:01Z</xmg:timestamp>

<xmg:type>DEPARTURE</xmg:type>

</xmg:ExampleMessage>

**Rulebook**

Versioning and namespace

* An Application Library shall have its own version number and namespace.
* An Application Library should employ semantic versioning (major.minor.micro) with version number starting at 1.0.0.

Message templates

* XSD complex type restrictions should be preferably used to pare down the FIXM standard to just those fields that are applicable to a particular message, as well as to enforce stricter optionality and content patterns based on the individual message. One set of these restrictions shall exist for each message.
* XSD profiles may be used as an alternative to XSD restrictions if technical issues with XSD restrictions are observed, such as incompatibility of XSD restrictions with a marshalling tool.

Copyright

* ...

**Creating an Application Library step by step**

* When creating a restriction, XML elements are eliminated by removing them from the model
* Optionality, cardinality, and pattern restrictions can all be further restricted by applying the desired changes to the restricted class.
* Because XSD complex type restrictions must use the same namespace as the types they restrict, it is necessary to change their names.

# How to create a FIXM Extension

## Initial Download and Setup

FIXM Extensions can be applied to Core, Applications, or both. Which FIXM product(s) you begin with is entirely based on the scope of your Extension.

1. Determine which FIXM product(s) you wish to target with your Extension.
2. Download the appropriate full release from <https://fixm.aero/download.pl>.
3. Unzip the full release and navigate to the “uml” directory.
4. Rename the .eap file to something appropriate for your Extension.
5. Open the model (the .eap file) with Sparx Enterprise Architect[32] (EA).

For this guided example, the extension will target both Core and BasicMessage (download available at <https://fixm.aero/release.pl?rel=Basic-Msg-1.0.0>). The .eap file should be renamed to “FIXM\_Example\_Extension\_v1.0.0\_with\_Basic\_Message\_v1.0.0\_and\_Core\_v4.2.0.eap”. The starting basic directory structure for the example environment is shown below.

## Create a Top-Level Extensions Container

Before creating the Extension itself, you should first create a top-level container under which to place your Extension (similar to Applications).

1. Right click on the Core package in Project Browser and choose “Add a Package…”.
2. In the New Package dialogue box, click on the “…” box next to Owner (near the top right corner) to open the Select Owner dialogue box.
3. In the Select Owner dialogue box, click on Fixm and then click OK.
4. In the New Package dialogue box, change the *Name* field to “Extensions”, select the *Package Only* radio button, and then click OK.

## Create an Extension Root Package

With your Extensions container in place, you can now create and configure the root package for your Extension.

1. Right click on the newly created Extensions container and once again choose “Add a Package…”.
2. In the New Package dialogue box, change the *Name* field to something appropriate to your Extension (here we used “Example”), select the *Package Only* radio button, and then click OK.

### Apply Schema Stereotype

In order to generate XSD schemas in Sparx EA, each package that corresponds to a schema file needs an XSDschema stereotype applied to it. To add an XSDschema stereotype to your package:

1. Double click (or right click and choose “Properties…”) on your Extension package (here called “Example”) in Project Browser to open the Package dialogue box.
2. In the newly opened Package dialogue box, click on the “…” box next to *Stereotype* (near the top right corner) to open the Stereotype dialogue box.
3. In the newly opened Stereotype dialogue box, select “XSDschema” and then click OK.
4. Then click okay in the Package dialogue box to apply the stereotype.

### Set Up Schema Properties

Once the stereotype is applied, you can configure your schema properties, setting up a number of XSD details such as schema file name, schema namespace, namespace prefixes, etc.

1. Once again, double click (or right click and choose “Properties…”) on your Extension package in Project Browser. With the XSDschema stereotype applied, this should now open an XSD schema Properties dialogue box.
2. Fill in values for your schema properties as appropriate for your Extension. In this example, we will use the following values:
   1. *Target Namespace* set to: “<http://www.fixm.aero/ext/example/1.0”>[33].
   2. *Prefix* set to: “xmp”[34].
3. You should also specify additional namespace prefixes for any packages your Extension will import. In this example, we will be importing Base and Flight from Core as well as BasicMessage. This step is accomplished by:
4. Clicking “New” to open a Namespace Details dialogue box.
5. Filling in appropriate details for each namespace to be included.
6. Then clicking OK.

See below for examples showing how each Namespace Details dialogue box should be filled in.

1. Finally, you must fill in the *Schema File* field with the path to your extension directory and an appropriate file name. In this example, the following *Schema File* entry should be used: .\schemas\extensions\example\Example.xsd.

IMPORTANT NOTE: Sparx EA will not automatically create any of the directories for the path to the file specified in the *Schema File* field. **They must be created outside of Sparx EA before generating the schemas.** FIXM is natively set up to generate its schema files under a directory named “schemas” that is located in the same directory as the Sparx EA file. To save time, rather than creating all of the directories needed by hand, you can instead:

1. Copy the “schemas” directory from the FIXM release you started with into the directory where your Sparx EA file is located (in this example, the “uml” directory from the downloaded release).
2. Under the “schemas” directory, add an “extensions” directory.
3. Under the new “extensions” directory, add another directory with a name appropriate for your extension (here “example” was used).

In this example, the new “schemas” directory should be added to the “uml” directory and should be structured as shown below.

1. With all of these fields filled out, your XSD schema Properties dialogue box should look something like below. Click OK to save these settings.

### Add Schema Description and Tags

The final step in setting up your schema details is to add a description for your package and adjust the schema configuration settings that are controlled in Sparx EA via tagged values.

1. Once more, double click (or right click and choose “Properties…”) on your Extension package in Project Browser and then click the UML button (near the top right corner). This will bring you back to your Package dialogue box.
2. In the central text box, fill in a description of your Extension package. The text entered here will also show up as a documentation element in your schema file.
3. Next, click on the *Tags* tab (near the lower right hand corner) to add three tags used by Sparx EA when creating your schema files: attributeFormDefault, elementFormDefault, and version.
   1. FIXM standardly sets attributeFormDefault to “unqualified”.
   2. FIXM standardly sets elementFormDefault to “qualified”.
   3. Version should be set as appropriate for your Extension (“1.0.0” in this example).

To add these tags, click on the third icon from the left [] in the *Tags* tab and fill in the *Tag* and *Value* fields similar to what is shown below.

1. When finished, your Package dialogue box will look something like below. Click OK to save these settings.

## Create Extension Content

The steps so far have centered on creating and setting up the Extension’s root package. The number and organization of additional Extension packages created below the root package (called “subcomponents” in this document to more easily distinguish them from the root package) will depend entirely on what the Extension is trying to accomplish. In this very simple example, the Extension will be used to:

* Add a new field called “position” to the EnRoute section of Core to report a flight’s current whereabouts.
* Add a new field called “sequenceNumber” to the BasicMessage section to help with message ordering.

The steps below describe how to create the subcomponents that will be used to contain these new fields.

### Create an Extension Subcomponent

This example will start with creating a subcomponent for the “position” field. Many of the steps outlined here are very similar to the steps listed in the [Create an Extension Root Package](#Xabb10ecdb993c67e46ee03da82474f39ff9752a) section above. The main difference is that Extension subcomponents contain diagrams and Extension classes.

1. Right click on the your Extension package and choose “Add a Package…”.
2. Change the *Name* field in the New Package dialogue box to something appropriate to your subcomponent (in this example, “ExampleEnRoute”).
3. Select the *Create Diagram* radio button.
4. Then click OK.
5. This will bring up a New Diagram dialogue box. In the *Type* section:
   1. Choose “UML Structural” under *Select From*.
   2. Choose “Class” under *Diagram Types*.

Click OK to create the subcomponent and its associated diagram.

1. Next, apply the XSDschema stereotype to your subcomponent as described in the [Apply Schema Stereotype](#Xaba45d1484d28e0e0b1e8bf0bf14b187b333fea) section above.
2. After the stereotype is applied, continue the schema setup as described in [Set Up Schema Properties](#set-up-schema-properties) above. The following property values should be used for this subcomponent in this example:
3. *Target Namespace* set to: “<http://www.fixm.aero/ext/example/1.0”>.
4. *Prefix* set to: “xmp”.
5. *Schema File* set to: “.\schemas\extensions\example\ExampleEnRoute.xsd”.

NOTE: For this and other subcomponents, you should be able to skip the step of setting up additional namespaces for imported packages. Sparx EA should automatically generate these as needed.

When finished, your XSD schema Properties dialogue box should look something like below.

1. Click OK to save these settings.
2. Reopen the subcomponent and finish the schema setup as described in [Add Schema Description and Tags](#add-schema-description-and-tags). As described in that section, the following tags should be added:
3. An attributeFormDefault tag set to: “unqualified”.
4. An elementFormDefault tag to: “qualified”.
5. A version tag set to an appropriate version for your Extension (“1.0.0” for this example).

### Create an Extension Class

Now it is time to create any Extension classes needed. This is typically done in FIXM by creating new classes that generalize the extension classes (the classes used throughout Core by its extension hooks) found in the Extension package located under Core’s Base package (not to be confused with the similarly referred to “Extension package” you are in the process of creating).

1. Double click on your subcomponent’s diagram (or right click it and choose “Open”) to get started.
2. Next, in Toolbox, click on “Complex Type” and then click anywhere in the diagram section of the screen. This will open up an XSD complexType Properties dialogue box.
3. Add an appropriate *Name* and *Annotation* for your new class.
4. Then click OK.
5. Next, right click on your newly created class and choose “Advanced” and then “Parent…” to bring up the Set Parents and Interfaces dialogue box.
6. In the *Add New Relation* section of the dialogue box, set *Type* to “Generalizes” and then click on “Choose…” to open up the Select Classifier dialogue box.
7. From there, navigate to Fixm -> Core -> Base -> Extension and then choose the Extension class you wish to target. In this example, it will be “EnRouteExtension”.
8. Then click OK.
9. Once this is done, you should see that EnRouteExtension is now listed in the *Type Details* section of the Set Parents and Interfaces dialogue box and is also shown in italics at the top of your class in the diagram. Your new class now extends the EnRouteExtension extension hook from Core and is ready to be used to hold your Extension fields. Click Close.

### Add Extension Class Content

You can now begin to add content to your new Extension class.

1. Right click on the class in the diagram and choose “Features and Properties” and then “Attributes…”. This opens the Features dialogue box. In the Features dialogue box, you can add any attributes[35] needed for your Extension to your class.
2. Begin by clicking where it says “New Attribute…” under *Name* and filling in your new field’s name. In this example, that will be “position”.
3. Then hit the tab key or otherwise click out of the *Name* field.
4. You should now have a new attribute added to your class with a default *Type* of “int” and *Scope* of “Private”.
5. Next, click on the *Type* field, select the down arrow on the right side, and then click “Select Type…”.
6. This opens the Select Type dialogue box. Navigate to the appropriate class in the Base package you want to use, select it, and then click OK. In this example, that will be Fixm -> Core -> Base -> AeronauticalReference -> GeographicalPosition.
7. Next change the *Scope* field to “Public”.
8. Adjust the *Multiplicity* in the *Attribute* section as needed (in this example, it should be set to 0..1).
9. Then add a description for the field in the *Notes* section. When finished, your Features dialogue box will look something like below.
10. Most optional fields in FIXM are also nillable. If you wish to make your field nillable:
11. Click on the *Tagged Values* tab.
12. Click on *Attribute*.
13. Click on the third icon from the left [] at the top of the tab to add a new tag.
14. In the *Tag* field type “nillable” and in *Value* type “true” and then click OK.
15. Continue adding as many attributes as desired. When finished, click Close on the Features dialogue box. The class diagram should display the name, type, and multiplicity of each attribute added to the class.
16. When satisfied with the subcomponent, right click on its name at the top of the diagram and click “Save Changes to ‘[diagram name]’”.

The class content presented here is simple but keep in mind that as many attributes and additional Extension-defined classes can be added to this extension hook as needed.

Along the same lines, the steps detailed under [Create Extension Content](#Xfc05112c7f9bfce48196d1bdd241a19501776b0) can be used to add as many additional subcomponents as needed to hold additional extension hooks or Extension-specific packages.

Below are a series of screenshots capturing key steps along the way to creating a second subcomponent for representing the “sequenceNumber” field needed for extending BasicMessage. The only notable difference between creating this subcomponent and the previous one is that the ExampleMessage class created generalizes an extension hook class from BasicMessage under Applications rather than a class from Extensions under Base.

## Generate the Extension Schemas

When all the desired Extension subcomponents are in place, it is time to generate the schemas for the Extension. The general steps for generating schemas from the logical model are outlined in APPENDIX C.

1. Begin the process by right clicking on your Extension root package and selecting “Code Engineering” and then “Generate XML Schema…”.
2. Then follow the same steps listed in APPENDIX C to set up and generate your schemas.

## Post-Process the Extension Schemas

After the schemas have been generated they will need to be post-processed. The post-processing script referenced in APPENDIX C will need to be modified in two ways in order to accommodate your Extension.

### Include Extension Namespace Prefix

FIXM standardly adds "Type" to the end of the name of all of the types declared in its schemas. In order to avoid doing this to built-in XSD types, the post-processing script uses namespace prefixes to determine which types it should modify. To accommodate this, your Extension’s namespace prefix needs to be added to the script for proper processing.

1. First, locate the else clause near the end of the script that contains the comment:

# Add "Type" to the end of all FIXM type names (both declaration and use).

1. In this section, you will see a series of search and replace commands. The bottom three of these have a portion that lists out all the namespace prefix options used by the FIXM schemas as part of the search field. These lines need to be modified to include the namespace prefix used by your Extension.

For the example created here, this change should be implemented as shown below (modification in red):

$line =~ s/<xs:(restriction|extension).\*base="(fb|fx|msg|xmp):[^"]\*/$&Type/;

$line =~ s/<xs:(attribute|element).\*type="(fb|fx|msg|xmp):[^"]\*/$&Type/;

$line =~ s/<xs:list.\*itemType="(fb|fx|msg|xmp):[^"]\*/$&Type/;

### Set Up and Use Package-Wide Include Files

FIXM makes use of “package-wide include files” that increase its usability with a number of XML tools. The schema files that will contain the needed include elements are generated by Sparx EA but it is the post-processing script that creates the actual include elements within these files.

Depending on the complexity and structure of your Extension, you may need many of these but you should always have at least one: the root schema file corresponding to your Extension’s root package. In this example, that is the “Example.xsd” file created by the Example package.

1. Directly above the section of the post-processing script referenced in the previous step (for adding “Type” to type names), locate the series of elsif clauses with conditionals along the lines of:

elsif ($schema =~ /\/[filename].xsd/)

1. At the end of this section, directly after the closing brace of the last elsif clause in the series, add your own elsif clause for setting up your Extension’s root schema file. The conditional needs to be set up to match your root schema file’s name (in this example, “Example.xsd”).
2. In the body of the clause, add include elements for the schema files of each subcomponent in your Extension as well as an import element for the package-wide include file of every set of schemas you use with a different namespace then your extension’s own.

For the example created here, the new clause should look like this:

# Add package-wide includes to Example.xsd

elsif ($schema =~ /\/Example.xsd/)

{

$line = "\t<xs:include schemaLocation=\"./ExampleEnRoute.xsd\"/>\n" .

"\t<xs:include schemaLocation=\"./ExampleMessage.xsd\"/>\n" .

"\t<xs:import namespace=\"<http://www.fixm.aero/base/4.2>\" schemaLocation=\"../../core/base/Base.xsd\"/>\n" .

"\t<xs:import namespace=\"<http://www.fixm.aero/flight/4.2>\" schemaLocation=\"../../core/flight/Flight.xsd\"/>\n" .

"\t<xs:import namespace=\"<http://www.fixm.aero/app/msg/1.0>\" schemaLocation=\"../../applications/basicmessage/BasicMessage.xsd\"/>\n" .

$line;

}

Finally, as noted in APPENDIX C, be careful not to run your modified post-processing script against any schema files that have already been post-processed. To avoid this issue, either regenerate the Core and/or Applications schemas you need before post-processing the entire schemas directory or run the script exclusively against the generated Extension schemas.

## Sample XML

Below is a sample XML document that validates against the Example Extension described in this Appendix:

<?xml version="1.0" encoding="UTF-8"?>

<msg:Message xmlns:xmp="<http://www.fixm.aero/ext/example/1.0"> xmlns:fx="<http://www.fixm.aero/flight/4.2"> xmlns:fb="<http://www.fixm.aero/base/4.2"> xmlns:msg="<http://www.fixm.aero/app/msg/1.0"> xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance">>

<msg:extension xsi:type="xmp:ExampleMessageType">

<xmp:sequenceNumber>1903</xmp:sequenceNumber>

</msg:extension>

<msg:flight>

<fx:enRoute>

<fx:extension xsi:type="xmp:ExampleEnRouteType">

<xmp:position srsName="urn:ogc:def:crs:EPSG::4326">

<fb:pos>36.019970 -75.668760</fb:pos>

</xmp:position>

</fx:extension>

</fx:enRoute>

</msg:flight>

</msg:Message>

**Rulebook**

Versioning & namespace

* An Extension should employ semantic versioning (major.minor.micro) with version number starting at 1.0.0.
* ...

Diagrams

* Include copyright notice in each model diagram

Naming

* Name characters limited to upper and lower case, digits and underscore
* Use InterCap notation (= “Camel Case”) for all model elements
* Use starting capital for packages and classes
* Use starting miniscule for attributes and roles
* Use all capitals for enumeration values
* Names should be expressive of data content or relationship
* Names should not be of unwieldy length

Inheritance

* Abstract classes are allowed
* Root classes are forbidden
* Leaf classes must be justified

Datatypes

* ...

Stereotypes

* ...

**Creating an Extension step by step**

# How to generate XML Schemas from a FIXM model using Sparx Enterprise Architect

## Generating Schemas from the Logical Model

The FIXM models are heavily annotated with stereotypes that allow Sparx Enterprise Architect (EA) to convert the logical model into XML schema files. This section describes how to invoke XSD generation in Sparx EA[36].

1. Open the .eap file that contains the Logical Model you would like to build.
2. Select the component of the model you would like to build in Project Browser[37]. For this example, we will be building FIXM Core (model available at <https://fixm.aero/releases/FIXM-4.2.0/FIXM_Core_v4.2.0.eap>).
3. Right click on the chosen component (in this example, “Core”) and select “Code Engineering” and then “Generate XML Schema…”.
4. This brings up the Generate XML Schema dialogue box. There are a number of configuration options available in this dialogue box that should be set as follows:
   1. *Encoding*: Set to Unicode (UTF-8)
   2. *XSD Style*: No options checked.
   3. *Referenced Package Options*: No options except “Use relative-path to reference XSDs” checked.
   4. *Child Package Options*: “Generate XSD for Child packages” checked and “Include <XSDschema> packages” radio button selected. Also, all packages you want to build (typically all packages listed) checked.

When finished, your dialogue box should look similar to below.

As you can see from the list of schema locations under the *Filename* heading, FIXM is natively set up to generate its schema files into a directory named “schemas” that is located in the same directory as the .eap file itself. Sparx EA will not automatically create any of the directories for the path to the files specified in the *Filename* fields. **They must be created outside of Sparx EA before generating the schemas.** The paths shown in the *Filename* fields should give you guidance as to the required directory structure.

1. Ensure proper directory structure is in place to hold the generated schema files.

For this example, the following directory structure must be placed in the same directory as the .eap file before attempting to generate the schemas.

This structure could be created by hand or, to save time, you could copy the “schemas” directory from an appropriate FIXM release, editing it as needed if you are building a logical model with custom content.

One final note on this topic: at times EA does not appear to recognize directories created during an open session. As such, it might be best to close and then reopen the logical model after the appropriate directory structure has been created to ensure Sparx EA will have access to it.

1. With the configuration options set and the schema directories in place, click the Generate button to produce the physical model.

The Progress section of the dialogue box will show each package being built and will eventually display “Schema Generation Complete” when finished. There should be no error or warning messages displayed during this process.

## Post-processing the FIXM Schemas

The schemas generated by Sparx EA require minor post-processing to align with FIXM best practices and to increase their usability with a number of XML tools. For FIXM Core, BasicMessage, and FficeMessage, this post-processing has been automated and is performed by a Perl script, which makes the following changes to the generated schema files:

* Adds Base and Flight package imports to Fixm.xsd.
* Adds package-wide includes to Base.xsd.
* Adds package-wide includes to Flight.xsd.
* Adds package-wide includes to Cargo.xsd.
* Adds package-wide includes to FlightRouteTrajectory.xsd.
* Replaces Base package imports with a single import of Base.xsd.
* Replaces Flight package imports with a single import of Flight.xsd.
* Repeats similar modifications for the Basic Message and FF-ICE Message Applications packages.
* Adds "Type" to the end of all FIXM type names (both declaration and use).
* Alphabetizes attribute declarations.
* Removes empty sequences (an artifact of Sparx EA schema generation)

The Perl script developed[38] for this task can be found at [insert location here].

When invoking the script, provide the name of the directory containing the generated schemas as a command line argument. The post-processing script iterates through all files in the directory as well as all files in all subdirectories below the supplied directory. The script prints out the names of all the schema files it detects and processes as it runs.

1. Invoke the Perl script to post-process your generated schemas.

In this example, navigate to the directory your “schemas” directory is located in and then type: “perl postprocess.pl schemas”.

Below is an example invocation of the script against the generated FIXM Core schemas along with the on-screen output it generates.

Three final notes about using the post-processing script:

* The supplied directory should contain only the schema files to be processed. Any other files present could be corrupted by this script.
* The script should only be run on the generated schemas once. Running the script multiple times will layer additional changes onto the schemas each time it is run, resulting in invalid schemas.
* The script was constructed specifically to run against FIXM Core 4.2.0, FF-ICE Message 1.0.0, and Basic Message 1.0.0. Running the script against any other schemas without modifying the script appropriately will likely produce either problematic or insufficient modifications to the schemas.

# FF-ICE/R1 Services Description Example – Details and Other Considerations

## Planning Service

|  |  |
| --- | --- |
| **Behaviour** |  |
| **Name** | Planning Service with REJ or non-concur preliminary flight plan |
| **Description** | This example illustrates the submission of a new preliminary flight plan that is rejected by the eASP for any reason, be it technical (such as syntax error in the request) or operational (such as penetrating a closed airspace). See XXX for details |

|  |  |
| --- | --- |
| **Behaviour** |  |
| **Name** | Planning Service with CONCUR (operationally acceptable) preliminary file plan |
| **Description** | This example illustrates the submission of a new preliminary flight plan that is accepted by the eASP. See XXX for details |

|  |  |
| --- | --- |
| **Behaviour** |  |
| **Name** | Planning Service with manually treated preliminary flight plan |
| **Description** | See Case c) Filing Service with manually treated flight plan. |

|  |  |
| --- | --- |
| **Behaviour** |  |
| **Name** | Planning Service with Preliminary Flight Plan requiring negotiation |
| **Description** | This example illustrates the submission of a new preliminary flight plan that requires negotiation. See XXX for details |

|  |  |  |
| --- | --- | --- |
| **Endpoints** |  |  |
| **Name** | **Description** | **Address** |
| **operational endpoint** | TBD | https… |

|  |  |
| --- | --- |
| **Service Description References (Implemented Standard)** |  |
| **Name** | ECTL SWIM TI YP |
| **Description** | This specification contains requirements for system interfaces (e.g. protocols) and for IT infrastructure capabilities |
| **Version** | v1.0 |
| **Reference** | public://2019-11/EUROCONTROL-SPEC-170 SWIM TIYP Ed 1.0.pdf |
| **Standard Type** | EUROCONTROL\_SPECIFICATION\_FOR\_SWIM\_TECHNICAL\_INFRASTRUCTURE |
| **Conformance Statement** | Implementation of service and network bindings |
| **Is conformant** | true |

|  |  |
| --- | --- |
| **Service Description References (Service Document)** |  |
| **Name** | PlanningServiceExample.wsdl |
| **Description** | TBD |
| **Version** | TBD |
| **Reference** | TBD |
| **Document Type** | Machine readable service description |

|  |  |
| --- | --- |
| **Service Description References (Service Document)** |  |
| **Name** | FficeTemplates.xsd |
| **Description** | TBD |
| **Version** | 1.0 |
| **Reference** | \schemas\applications\fficemessage\fficetemplates |
| **Document Type** | Machine readable message description |

### Example WSDL

For operation submitPreliminaryFlightPlan:

### Service Behaviour

To illustrate the planning service behavior and associated interactions, four cases are distinguished:

* The ***Preliminary Flight Plan Message*** submission is rejected (***REJ or NON CONCUR***). For details see [10] Page II-4-13 Section 4.4.5.4 and [10] Page II-5-29 Section 5.3.7 iii
* The ***Preliminary Flight Plan Message*** submission is accepted (***ACK & CONCUR***). For details see [10] Page II-4-13 Section 4.4.5.2/4.4.5.3 and [10] Page II-5-28 Section 5.3.7 i
* The ***Preliminary Flight Plan Message*** submission requires manual intervention (***MAN***). For details see [10] Page II-4-13 Section 4.4.5.5
* The ***Preliminary Flight Plan Message*** submission requires negotiation (**NEGOTIATE**) as described in [10] Page II-5-29 Section 5.3.7 ii

Case a) Planning Service with REJ or non-concur preliminary flight plan

\*\*Description  
\*\*This example illustrates the submission of a new preliminary flight plan that is rejected by the eASP for any reason, be it technical (such as syntax error in the request) or operational (such as penetrating a closed airspace).

**Expected FF-ICE interaction**

Figure 13: FF-ICE interaction for REJ or non-concur flight plan submission

According to the FF-ICE/R1 Implementation Guidance Manual [10], the submission could be:

1. Rejected immediately for having failed data acceptability: in this case a ***Submission Response*** = ***REJ*** is used and no ***Planning Status Message*** is to follow.
2. Accepted with respect to data acceptability but not accepted due to operational acceptability: if the data acceptability is performed at a later stage this should result in a ***Submission Response*** = ***ACK*** followed by a ***Planning Status*** ***Message*** with the errors.

**Example implementation**

Figure 14: Preliminary Flight plan submission example with preliminary flight plan rejected for having failed data acceptability (REJ)

* The PreliminaryFlightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The PreliminaryFlightPlanSubmissionReply contains the status ***REJ*** and the reason.

Figure 15: Preliminary Flight plan submission example with preliminary flight plan rejected for having failed operational acceptability (NON CONCUR)

* The preliminaryFlightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The preliminaryFlightPlanSubmissionReply contains the status ***ACK***.
* The planningStatusMaessage contains the ***Planning Status*** with status ***NON CONCUR.***

Whatever the submission operation, the FF-ICE/R1 Implementation Guidance Manual [10] always makes a distinction between data acceptability and operational acceptability. However, some eASP may want to perform the two together as part of the same transaction. This would translate into the following figure.

Figure 16: Preliminary Flight plan submission example with rejected (REJ or ACK+NON CONCUR) preliminary flight plan

In this example, the data acceptability and operational acceptability are combined into the same transaction. The resulting synchronous reply conveys all necessary information: the ***REJ*** status and the operational reason that led to the ***NON CONCUR***. As outlined by the diagram, the interaction only takes place as a R/R and does not need to generate any asynchronous messages.

* The PreliminaryFlightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The PreliminaryFlightPlanSubmissionReply contains the status ***REJ*** and the reason or ***ACK*** and ***NON CONCUR***. In FF-ICE/R1 terminology this reply implements both the ***Submission Response*** and the ***Planning Status***.

Case b) Planning Service with CONCUR (operationally acceptable) preliminary file plan

**Description**  
This example illustrates the submission of a new preliminary flight plan that is accepted by the eASP.

**Expected FF-ICE interaction**

Figure 17: FF-ICE interaction with accepted (CONCUR) preliminary flight plan

**Example implementation**

Figure 18: Preliminary Flight plan submission example with accepted (CONCUR) preliminary flight plan

* The preliminaryFlightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The preliminaryFlightPlanSubmissionReply contains the status ***ACK***.
* The planningStatusMaessage contains the ***Planning Status*** with status ***CONCUR.***

Figure 18 mirrors the distinction between data acceptability and operational acceptability that is introduced by the FF-ICE/R1 Implementation Guidance. However, both aspects may be addressed technically as part of the same transaction, as illustrated by Figure 19.

Figure 19: Preliminary Flight plan submission example with accepted (ACK and CONCUR) preliminary flight plan

In this example, the data acceptability and operational acceptability are combined into the same transaction. The request is processed successfully and the preliminary flight plan is ***CONCUR*** (operationally acceptable).

* The preliminaryFlightPlanSubmissionRequest contains the preliminary flight plan encoded in FIXM 4.1.0.
* The preliminaryFlightPlanSubmissionReply contains the status ***ACK*** and the accepted preliminary flight plan encoded in FIXM 4.1.0. In FF-ICE terminology this reply contains both the ***Submission Response*** and the ***Planning Status***.

Case c) Planning Service with manually treated preliminary flight plan

See Case c) Filing Service with manually treated flight plan.

Case d) Planning Service with Preliminary Flight Plan requiring negotiation

**Description**  
This example illustrates the submission of a new preliminary flight plan that requires negotiation.

**Expected FF-ICE interaction**

Figure 20: FF-ICE interaction with preliminary flight plan that requires negotiation (NEGOTIATE)

**Example implementation**

Figure 21: Preliminary Flight plan submission example with preliminary flight plan that requires negotiation (NEGOTIATE)

* The preliminaryFlightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The preliminaryFlightPlanSubmissionReply contains the status ***ACK***.
* The planningStatusMaessage contains the ***Planning Status*** with status ***NEGOTIATE.***

Figure 21 mirrors the distinction between data acceptability and operational acceptability that is introduced by the FF-ICE/R1 Implementation Guidance. However, both aspects may be addressed technically as part of the same transaction:

Figure 22: Preliminary Flight plan submission example with preliminary flight plan that requires negotiation (ACK and NEGOTIATE)

In this example, the data acceptability and operational acceptability are again combined into the same transaction. The request is processed successfully and the preliminary flight plan is ***NEGOTIATE*** (requires negotiation).

* The preliminaryFlightPlanSubmissionRequest contains the preliminary flight plan encoded in FIXM 4.1.0.
* The preliminaryFlightPlanSubmissionReply contains the status ***ACK*** and the proposed for negotiation preliminary flight plan encoded in FIXM 4.1.0. In FF-ICE terminology this reply contains both the ***Submission Response*** and the ***Planning Status***.

#### Data Implementation Description

This part will be addressed in a future version of the document.

## Filing Service

### Service Behaviour

To illustrate the filing service behaviour and associated interactions, three cases are distinguished:

1. The ***Filed Flight Plan*** is rejected (***REJ or NOT ACCEPTABLE***), as described in (resp.) [10] Page II-4-13 Section 4.4.5.4 and Page II-6-2 Section 6.3.4 ii.
2. The ***Filed Flight Plan*** is accepted (***ACK and ACCEPTABLE***), as described in (resp.) [10] Page II-4-13 Section 4.4.5.2/4.4.5.3and Page II-6-1 Section 6.3.4 i
3. The ***Filed Flight Plan*** requires manual intervention (***MAN***), as described in [10] Page II-4-13 Section 4.4.5.5

Case a) Filing Service with non-acceptable flight plan

**Description**  
This example illustrates the submission of a new flight plan that is rejected by the eASP for any reason, be it technical (such as syntax error in the request) or operational (such as penetrating a closed airspace).

\*\*  
\*\*

**Expected FF-ICE interaction**

Figure 23: FF-ICE interaction for invalid flight plan submission (REJ)

Figure 24: FF-ICE interaction for non-acceptable flight plan submission

According to the FF-ICE conceptual document the submission could be:

1. Rejected immediately for having failed data acceptability: in this case a ***Submission Response* = *REJ*** is used and no ***Filing Status Message*** is to follow.
2. Accepted with respect to data acceptability but not accepted due to operational acceptability: if the data acceptability is performed at a later stage this should result in a ***Submission Response*** = ***ACK*** followed by a ***Filing Status*** with the errors.

**Example implementation**

Figure 25: Flight plan filing submission example with filed flight plan rejected for having failed data acceptability (REJ)

* The flightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0
* The flightPlanSubmissionReply contains the status ***REJ*** and the reason.

Figure 26: Flight plan filing submission example with filed flight plan rejected for having failed operational acceptability (NOT ACCEPTABLE)

* The flightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The flightPlanSubmissionReply contains the status ***ACK***.
* The filingStatusMaessage contains the ***Filing Status*** with status ***NON ACCEPTABLE.***

As noted already for the Planning Service, both data acceptability and operational acceptability may be addressed technically as part of the same transaction.

Figure 27: Flight plan filing submission example with rejected (REJ or ACT+NON ACCEPTABLE) filed flight plan

By doing so, the synchronous reply conveys all necessary information: the ***REJ*** status and the operational reason that led to the ***REJ***. As shown in Figure 27, the interaction only takes place as a R/R and does not need to generate any asynchronous messages.

The flightPlanSubmissionReply contains the status ***REJ*** and the reason. In FF-ICE terminology this reply implements both the ***Submission Response*** and the ***Filing Status.***

Case b) Filing Service with acceptable file plan

\*\*Description  
\*\*This example illustrates the submission of a new flight plan that is accepted by the eASP. Having combined data acceptability and operational acceptability into the same transaction, this means the request was processed successfully, and the flight plan is operationally acceptable.

\*\*  
\*\*

**Expected FF-ICE interaction**

Figure 28: FF-ICE interaction with accepted flight plan

**Example implementation**

Figure 29: Flight plan filing example with accepted flight plan

* The flightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The flightPlanSubmissionReply contains the status ***ACK***.
* The FilingStatusMessage contains the ***Filing Status*** with status ***ACCEPTABLE.***

Once again, both data acceptability and operational acceptability may be addressed technically as part of the same transaction.

Figure 30: Flight plan filing submission example with accepted (ACK and ACCEPTABLE) filed flight plan

By doing so, the synchronous reply conveys all necessary information: the ***ACK*** status and the accepted flight plan with the accepted trajectory.

* The flightPlanSubmissionReply contains the status ***ACK*** and the accepted flight plan encoded in FIXM 4.1.0. In FF-ICE terminology this reply contains both the ***Submission Response*** and the ***Filing Status***.

\*\*  
\*\*

Case c) Filing Service with manually treated flight plan

\*\*Description  
\*\*This example illustrates the submission of a new flight plan that is queued for manual intervention (***MAN***).

**Expected FF-ICE interaction**

Figure 31: FF-ICE interaction for manually treated flight plan

In case a flight plan submission requires manual treatment, FF-ICE foresees a ***Submission Response***=***MAN*** followed by a subsequent asynchronous ***Submission Response*** which could be ***ACK*** or ***REJ***, followed by an optional ***Filing Status*** that would contain the accepted flight plan or the reasons why it was not accepted (***NOT ACCEPTABLE***).

**Example implementation**

Figure 32: Flight plan filing example with manually treated flight plan

* The flightPlanSubmissionRequest contains the flight plan encoded in FIXM 4.1.0.
* The flightPlanSubmissionReply contains the status ***MAN*** and the accepted flight plan encoded in FIXM 4.1.0. In FF-ICE terminology this reply contains both the ***Submission Response*** and the ***Filing Status***.
* The FlightFilingResultMessage contains the result of the processing (***ACK*** or ***REJ***). In case of rejection it contains the reason why the flight plan was not accepted, in case of acceptance it contains the accepted flight plan. In practice it contains both the ***Submission Response*** and the ***Filing Status***.
* The FilingStatusMessage is sent only in case the submitted flight plan is accepted (***ACK***) and it contains the accepted flight plan encoded in FIXM 4.1.0.

#### Data Implementation Description

This part will be addressed in a future version of the document.

## Technical assumptions

The FF-ICE/R1 Implementation Guidance [10] mentions synchronous (e.g. ***Submission Response Message***) and asynchronous replies (e.g. ***Filing Status Message***) but does not provide definitions for these terms. This complicates the interpretation of the FF-ICE/R1 interactions. For example does synchronous mean the client blocks until the server replies or does it simply mean “timely”?

The chapter assumes the following:

1. The request submission and the synchronous response are realized as SOAP request/reply.
2. The asynchronous response is realized via an AMQP notification (which can be thought of as a message queue).

This is shown in the following diagram.

Figure 11: Synchronous and asynchronous communication between eAU and eASP

In all the following examples, the diagrams will use the colour scheme introduced by Figure 11: orange for synchronous communication, purple for asynchronous.

### Setting up for asynchronous notifications

In order to setup the AMQP endpoint from which to receive the asynchronous notifications the eAU needs to express such an interest to the eASP.

The initiative to establish a connection could be on either side (client or server). For example, the eAU could provide its own AMQP end-point to which the eASP shall connect to publish the asynchronous notifications. Conversely, an AMQP end-point could be created on the server side and the client could connect to it to fetch the notifications. The latter has the clear advantage of leveraging the server (i.e. the eASP) from having to deal with all security-related issues (such as VPNs, firewall rules, authentication, etc) for each eASP.

In the examples provided below, the eAU can invoke a specific web service operation createSubscription by which it requests the eASP to create an AMQP end-point on the server side to which it can then connect to fetch the asynchronous notifications.

This is shown in the following diagram:

Figure 12: SubscriptionCreationRequest to setup the AMQP end-point

The content of the SubscriptionCreationRequest should contain enough information to allow the eASP to decide which notifications shall be sent on this AMQP end-point (i.e. for which flights). Typically, the subscription should at least cover all the flights filed or operated by the eAU.

For example the content of such a request could be:

1. Create a subscription for all flight plans operated by *[operator name/id]*
2. Create a subscription for all flight plans submitted by *[eAU name/id]*
3. Create a subscription for all flight plans flown by the following aircrafts *[aircraft id]*
4. Create a subscription for all flight plans departing/arriving from/to the given airports *[aerodrome id]*
5. Create a subscription for all flight plans that concern a given control centre *[control centre id]*

The content of the SubscriptionCreationReply contains the URL of the AMQP end-point to which the eAU can connect.

This operation is to be invoked only once. Note that this AMQP channel could also be created manually by the eASP following an agreement between the eAU and the eASP.

All the examples in this chapter foresee the existence of this AMQP channel. There will be four types of asynchronous messages:

1. the FlightPlanningResultMessage,
2. the PlanningStatusMessage,
3. the FlightFilingResultMessage
4. the FilingStatusMessage,

These messages are further explained in the following chapters.

## Message classification and technology selection

FF-ICE/R1 Messages can be classified from the perspective of the participant who provides the service. Two types of messages can be identified: input messages and output messages. The input messages are those received by the service provider while the output messages are sent by the service provider.

According to the classification above, the FF-ICE/R1 Messages can be classified as following:

1. ***Planning Service***
   1. Input messages:
      1. ***Preliminary Flight Plan Message***, as specified in [10] Page II-E-3 Section C-2
      2. ***Flight Plan Update Message****,* as specified in [10] Page II-E-19 Section C-9
      3. ***Flight Cancellation Message***[39] as specified in [10] Page II-E-17 Section C-8
   2. Output messages:
      1. ***Submission Response Message****,* as specified in [10] Page II-C-1 Section C-1
      2. ***Planning Status Message****,* as specified in [10] Page II-E-8 Section C-3
2. ***Filing Service***
   1. Input messages:
      1. ***Filed Flight Plan Message****,* as specified in [10] Page II-E-10 Section C-4
      2. ***Flight Plan Update Message****,* as specified in [10] Page II-E-19 Section C-9
      3. ***Flight Cancellation Message8****,* as specified in [10] Page II-E-17 Section C-8
   2. Output messages:
      1. ***Submission Response Message****,* as specified in [10] Page II-C-1 Section C-1
      2. ***Filing Status Message****,* as specified in [10] Page II-E-14 Section C-5
3. ***Flight Data Request Service***
   1. Input messages:
      1. ***Flight Data Request Message****,* as specified in [10] Page II-E-24 Section C-10
   2. Output messages:
      1. ***Flight Data Response Message****,* as specified in [10] Page II-E-26 Section C-11
      2. ***Submission Response Message****,* as specified in [10] Page II-C-1 Section C-1

Messages can be arranged according to the source that sends the message; however, the initiative of the communication is another aspect to be evaluated to select a suitable service implementation.

In all the input messages introduced above, the initiative of the communication is taken by the service consumer; these messages are called *requests*. Every time a service provider receives a *request* message, they will send back an output message. Those output messages, known as *reply* messages, are triggered by the reception of the *request* message and therefore the initiative of such communication is taken by the service consumer.[40]

In contrast, some output messages can be sent by the service provider on its own initiative. These messages are called push messages or *notification* messages.[41]

In order to document the FIXM-Based examples in this section, two sets of technologies have been selected that accommodate the need to exchange request, reply, and notification messages:

* **SOAP Web Service technologies** allow to exchange request and reply messages where the initiative is taken by the service consumer.[42]
* **AMQP** supports the exchange of notification messages where the initiative is taken by the service provider.

The following set of tables describes a possible design of FIXM-Based Services in support of the FF-ICE/R1 Services described in [10].

# Use of Schematron

There is some validation functionality that an XML schema alone cannot provide. For example, there is no way for an XSD to make a particular XML element required or optional based on the content of another element. However, some message exchange business rules require exactly these sorts of checks.

Schematron is a validation language capable of handling business rules of this nature. As such, the use of Schematron can supplement the limitations of XSDs and provide enforcement of any business rules outside the scope of what XML schemas can offer.

**Error! Reference source not found.** provides examples of FIXM business rules that could be encoded and checked using Schematron technology. Schematron encodings are however not provided in this version of the document. Future versions may revisit the overall formulation and description method for FIXM business rules, in particular in the light of the related AIXM experience.

# FIXM Development Tool Compatibility

Research Goal

What do you want to know, prove, demonstrate, analyze, test, investigate or examine? List your project goals…for example…

•The goal of this research is to:

oIllustrate a new methodology/architecture/product/invention that has never been built before

oDetermine the efficacy of your new method/architecture/product/invention

oCreation of a research roadmap as it pertains to my new method/architecture/product/invention

Background and/or Theories

What is already known or unknown? What past research are you building upon?

Hypotheses (optional)

Depending on the nature of you research, hypothesis might not be stated up front. For example, if you choose a qualitative research

method that leverages grounded theory, then a the “theory” is an emergent property at the END of your research (i.e., a theory is

developed inductively not deductively when using grounded theory). However, if your method is Quantitative or Design Science

oriented, then you should include verbiage upfront in your proposal that discusses what your hypothesis (or hypotheses will be).

Methodology

Research Goal

What do you want to know, prove, demonstrate, analyze, test, investigate or examine? List your project goals…for example…

•The goal of this research is to:

oIllustrate a new methodology/architecture/product/invention that has never been built before

oDetermine the efficacy of your new method/architecture/product/invention

oCreation of a research roadmap as it pertains to my new method/architecture/product/invention

Background and/or Theories

What is already known or unknown? What past research are you building upon?

Hypotheses (optional)

Depending on the nature of you research, hypothesis might not be stated up front. For example, if you choose a qualitative research

method that leverages grounded theory, then a the “theory” is an emergent property at the END of your research (i.e., a theory is

developed inductively not deductively when using grounded theory). However, if your method is Quantitative or Design Science

oriented, then you should include verbiage upfront in your proposal that discusses what your hypothesis (or hypotheses will be).

## Introduction

Typically, the development of data exchange standards are based on Logical and Physical Model best practices. Logical Models use UML best practices to show the relationships between key concepts whereas the Physical Model use schema best practices to develop the data exchange standard. The development of different standards deviate from best practices to accommodate unique use cases required by stakeholders. Although it is not the responsibility of exchange models to be “compatible” with various development tools, compatibility is indeed critical for stakeholder and industry adoption making an analysis of Development Tool Compatibility essential.

To this end, a compatibility analysis was run against the FIXM schemas and

* a variety of technologies (e.g., SOAP, REST, JMS)
* several common development tools.

The result of this compatibility analysis was the creation of the FIXM support matrix. The section below titled *Platform Support Matrix* supplies a list of the supported tools and technologies. This list of currently supported software versions is also located on the FIXM work area.

## Evaluation Environment

The evaluation process included the following components:

1. **FIXM Schemas**
   1. FIXM Core 4.2.0
   2. FF-ICE Message 1.0.0 (with restrictions)
2. **WSDL file**

Two WSDL files were tested

1. The first file contained FIXM schema details that contained no restrictions.
2. The second filed contained FIXM schema details that contained restrictions.
3. **FlightPlanningService Web Service**

The FIXM test web service being evaluated here is called *FlightPlanningService*, which supports one operation called *submitFlightPlan*. Developer can issue a submitFlightPlan remote request, as either a REST or SOAP call, to the FIXM *FlightPlanningService* and receive a submission response from the Service.

This is a very basic web service to test the sending of minimal flight plan XML via SOAP to a server in an attempt to get a web service SubmissionResponse. Testing focused mainly on Java based client and server.

The below sections will outline the approach and findings associated with the evaluation of various tools tested in interpreting the Fixm WSDL file to produce the Fixm server and client Java code.

A number of tools were tested but the Apache Axis library and the WSDL2Java tool were found to provide the most success.

**Example WSDL (FIXM Schema with Restrictions)**

## Apache Axis library and the WSDL2Java tool

The Apache Axis library provides the WSDL2java tool, which can interpret WSDL files. Axis1 and Axis2 versions were tested.

Before running the WSDL2Java tool required two questions needed to be answered:

**Which data binding parameter option to pass to it?**

Two data binding options, ADP and JAXBRI, were tested. These options provide two different approaches when interpreting schemas to create data models in JAVA.

ADP uses its own Axis1 based parsing tool. Supports limited validation but no range checking

JAXBRI uses JAXB as a parsing tool. Utilize JAXB for schema validation in the code (big timesaving). It does not support parameter testing but does support limited field checking i.e. ‘Required’ versus ‘Optional’

**Where to Run It?**

***Option 1 –Through an IDE using an Axis plugin***

The IDE used did not have a deciding impact on the results. However, they did determine the Data Binding option selected. Developers cannot specify the data binding through the IDE as the IDE interacts with the WSDL2Java tool at the backend. IDEs are Axis1 based and run WSDL2Java with the default data binding option ADP.

For this option, the only variant is the WSDL2Java tool version (i.e. Axis1 or Axis2) as the data binding option remains the same therefore

***Option2 – From the command-line***

For this approach, testing focused on the Axis2 version only as Axis1 failed to generate any code through the IDEs.

Developers can specify the data binding option from the command-line when executing WSDL2Java command.

Both options were tested i.e. ADP and JAXBRI.

For this option, the only variant was the data binding option, as the WSDL2Java tool version remains the same.

**Note**: Testing of the JAXBRI data binding was carried out for the command-line option only.

## Evaluation Results

### Definitions

**Pass Outcome**

Succeeded in generating compilable and runnable Fixm server and client code. The process to achieve this may or may not have required additional coding.

**Failed Outcome**

Failed in generating compilable and runnable Fixm server and client code. No additional workarounds were available or the amount of coding needed made the solution impractical.

### Pass Outcomes

1. Axis2 WSDL2Java tool executed from the IntelliJ IDE

* With IDE’s default data binding ADP
  + FIXM schema with no restrictions – passed with no additional work.
  + FIXM schema with restrictions - required additional coded fixups to produce compilable code.

1. Axis2 WSDL2Java tool executed from the command-line

* With data binding ADP
  + FIXM schema with no restrictions - passed with no additional work.
  + FIXM schema with restrictions - required additional coded fixups to produce compilable code.
* With data binding JAXBRI
  + FIXM schema with no restrictions - passed with no additional work.
  + FIXM schema with restrictions - passed with no additional work.

The results indicate that executing the Axis2 version of the WSDL2Java tool from the command-line and supplying the JAXBRI binding option resulted in the greatest success, as it required no additional coding fix-ups to generate a compilable server and client.

### Failed Outcomes

1. Axis1 WSDL2Java tool executed from the IntelliJ IDE with ADP data binding

* Failed for FIXM schema with and without restrictions

1. Axis1 WSDL2Java tool executed from the Eclipse IDE with ADP data binding

* Failed for FIXM schema with and without restrictions

1. Apache CXF in Eclipse

* CXF failed completely for FIXM schema with and without restrictions.

1. JAX-WS in Eclipse (using NetBeans)

* Passed for FIXM schema with no restrictions. Failed for schema with restrictions.

1. Glassfish/JAX-WS on IntelliJ IDE

* Passed for FIXM schema with no restrictions. Failed for schema with restrictions.

## Future Testing

The growth of the FIXM Support Matrix needs to be the responsibility of the entire FIXM community.  Coordinated feedback and testing is strongly encouraged between all FIXM community members.

Examples of potential areas for testing include:

* ASP.net Web Client
  + Visual Studio/.NET (for RESTful WS) on Windows 10
  + Visual Studio/.NET (for SOAP WS) on Windows 10
* ASP.net Web Services
* WSDL to Java Tools - JAX-RS in Eclipse (using NetBeans)
* Linux Operating system

## Platform Support Matrix

FIXM does not manage the software listed below .This list of tools is provided for convenience.

# Appendix F. Developing a Basic Web Service Using FIXM (Server and Client)

The following outlines the steps to build a basic FIXM web service.

1. **Generate JAXB-RI-Bound Source Code**

Execute the WSDL2Java command (from the command-line)

Successful execution results in the generation of:

* A receiving stub, which contains a code skeleton for receiving a
* submitted flight plan. All business logic for your project should be within the calling scope of the skeleton.
* A well-defined Flight Plan response model that the skeleton code
* returns.
* The file Build.xml (ANT tool uses this).

1. **Add additional logic**

Add additional client code and server-side business logic to implement the services, which the web service needs to function. **Note**: The generated-code will compile but will fail with Exceptions. Additional programming is required.

1. **Build and Deploy Server Code**

Issue the following command to compile the entire source code. Ant will package the service as an ‘aar’ file and deploy it to the Axis service folder running under tomcat.

1. **Start Tomcat Server**

To start Tomcat server run the following command that is located in the %CATALINA\_HOME %/bin directory.

1. **List Services**

List the services provided by the web service. The expected services will be listed under “Available services”.

<http://localhost:8080/axis2/services/listServices>.

1. **Verify Running Services**

Obtain the WSDL from the service to verify that the service is running and able to accept requests

<http://localhost:8080/axis2/services/FlightPlanningService?wsdl>

# ATS Message to FIXM Mapping

#### Mapping of ATS Fields to FIXM

This section provides a mapping from fields in PANS-ATM ATS messages to the FIXM Logical Model, one ATS message field per subsection. The columns in the mapping tables are defined in Table 8.

Table 8: Column Definitions

|  |  |
| --- | --- |
| **Column** | **Description** |
| PANS-ATM Field | The field number as defined in ICAO Doc 4444 [PANS-ATM]. |
| Package | The package that contains the definition of the PANS-ATM field in the logical model. |
| Class | The class (in the specified package) that models the PANS-ATM field. |
| Path from Flight | Starting from class *Flight* in package *Flight.FlightData*, this defines the path to the location in the logical model where the field is encoded. |

Table 9 provides an explanation of an entry in the map using the flight identifier recorded in field 7a of an ICAO ATS message (section Field 7).

Table 9: Example

|  |  |  |
| --- | --- | --- |
| Column | Value | Description |
| PANS-ATM Field | 7a | This is the field number from PANS-ATM that represents the flight identifier, which is being mapped to the logical model. |
| Package | Base.Types | The flight identifier is modelled in the *Base.Types* package. |
| Class | AircraftIdentification | The name of the class that models a flight identifier is *AircraftIdentification*. |
| Path from Flight | flightIdentification.aircraftIdentification | Starting at class *Flight*, follow the *flightIdentification* association to class *FlightIdentification*, then to the *aircraftIdentification* attribute of that class (which is of type *AircraftIdentification*). |

A PANS-ATM ATS message field may be mapped to different FIXM elements depending on context. A simple constraint notation based on logic and set theory is employed to specify these conditions. The notation is described in Table 10.

Table 10: Constraint Notation

|  |  |
| --- | --- |
| Notation | Description |
| [ . . . . ] | A constraint. The field in question is only encoded in the specified FIXM element if the constraint is satisfied. |
| A ∧ B | Logical conjunction: both A and B are true. |
| A ∨ B | Logical disjunction: A is true or B is true. |
| A = B | Equality: A and B are equal. |
| A ≠ B | Inequality: A and B are not equal. |
| A ∈ B | Set membership: the item A is contained in the set/list B. |
| A ∉ B | Set exclusion: the item A is not contained in the set/list B. |
| Free text | If the constraint is not amenable to formal specification, it is described in text. |

The term ‘〈kind〉’ in the subsequent tables (fields 8, 13, 15, 16 and 18) is a reference to the kind of route/trajectory information to which a field is mapped. That route information is dependent on the message type. Refer to section Varieties of Route for a mapping between the message type and the kind of route information.

##### Field 3

Field 3 in an ATS message denotes the message type. FIXM is concerned with modelling information that may be included in a message, but FIXM itself does not define messages (section **Error! Reference source not found.**). As such, there is no equivalent of ATS field 3 in FIXM.

##### Field 5

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 5a | Flight.Emergency | EmergencyPhase | emergency.phase |
| 5b | Base.Types | TextName | emergency.originator.atcUnitNameOrAlternate |
| 5c | Base.Types | CharacterString | emergency.emergencyDescription |

##### Field 7

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 7a | Base.Types | AircraftIdentification | flightIdentification.aircraftIdentification |
| 7b/c | Base.Types | ModeACode | enRoute.currentModeACode |

##### Field 8

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 8a | Flight.FlightRouteTrajectory.RouteTrajectory | FlightRulesCategory | routeTrajectoryGroup.〈kind〉.routeInformation.flightRulesCategory |
| 8b | Flight.FlightData | TypeOfFlight | flightType |

##### Field 9

##### Field 10

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 10a | Flight.Capability | StandardCapabilitiesIndicator | aircraft.capabilities.standardCapabilities |
|  |  | CommunicationCapabilityCode | aircraft.capabilities.communication.communicationCapabilityCode |
|  |  | DatalinkCommunicationCapabilityCode | aircraft.capabilities.communication.datalinkCommunicationCapabilityCode |
|  |  | NavigationCapabilityCode | aircraft.capabilities.navigation.navigationCapabilityCode |
| 10b | Flight.Capability | SurveillanceCapabilityCode | aircraft.capabilities.surveillance.surveillanceCapabilityCode |

##### Field 13

##### Field 14

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 14a | Base.AeronauticalReference | SignificantPointChoice | enroute.boundaryCrossingCoordination.crossingPoint |
| 14b | Base.Types | Time | enroute.boundaryCrossingCoordination.crossingTime |
| 14c | Base.RangesAndChoices | FlightLevelOrAltitudeChoice | enroute.boundaryCrossingCoordination.clearedLevel |
| 14d | Flight.EnRoute | FlightLevelOrAltitudeChoice | enroute.boundaryCrossingCoordination.altitudeInTransition.level |
| 14e | Flight.EnRoute | BoundaryCrossingCondition | enroute.boundaryCrossingCoordination.altitudeInTransition.crossingCondition |

##### Field 15

##### Field 16

##### Field 17

##### Field 18

##### Field 19

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 19a | Base.Types | Duration | supplementaryData.fuelEndurance |
| 19b | Base.Types | Count | supplementaryData.personsOnBoard |
| 19c | Flight.Capability | EmergencyRadioCapabilityType | aircraft.capabilities.survival.emergencyRadioCapabilityType |
| 19d | Flight.Capability | SurvivalEquipmentType | aircraft.capabilities.survival.survivalEquipmentType |
| 19e | Flight.Capability | LifeJacketType | aircraft.capabilities.survival.lifeJacketType |
| 19f | Base.Types | Count | aircraft.capabilities.survival.dinghyInformation.number |
|  | Base.Types | Count | aircraft.capabilities.survival.dinghyInformation.totalCapacity |
|  | Flight.Capability | DinghyCoverIndicator | aircraft.capabilities.survival.dinghyInformation.covered |
|  | Base.Types | CharacterString | aircraft.capabilities.survival.dinghyInformation.colour |
| 19g | Base.Types | CharacterString | aircraft.coloursAndMarkings |
|  | ~~Base.Types~~ | ~~CharacterString~~ | ~~aircraft.significantMarkings~~ |
| 19h | Base.Types | CharacterString | aircraft.capabilities.survival.survivalEquipmentRemarks |
| 19i | Base.Types | TextName | supplementaryData.pilotInCommand.name |

##### Field 20

##### Field 21

|  |  |  |  |
| --- | --- | --- | --- |
| ICAO 4444 Field | Package | Class | Path from Flight |
| 21a | Base.Types | Time | radioCommunicationFailure.contact.lastContactTime |
| 21b | Base.Measures | Frequency | radioCommunicationFailure.contact.lastContactFrequency |
| 21c | Base.AeronauticalReference | SignificantPointChoice | radioCommunicationFailure.contact.position.position |
| 21d | Base.Types | Time | radioCommunicationFailure.contact.position.timeAtPosition |
| 21e | Base.Types | CharacterString | radioCommunicationFailure.remainingComCapability |
| 21f | Base.Types | CharacterString | radioCommunicationFailure.radioFailureRemarks |

##### Field 22

In an ATS message, field 22 specifies a change to the information associated with a flight. It does not define new information elements, just a modification to elements that appear in other fields. As such, there are no mapping rules for field 22. The mapping of the information that can be specified in field 22 is captured in the other fields. For example, the entry *-7/NEWACID* in field 22 has the same mapping as if *–NEWACID* appeared in field 7 (on page 110).

[1](https://www.fixm.aero/fixm_410.pl) To do so, FIXM has transitioned from attributes to allowing for the model-wide support of “nillable” properties.

[2] They should not, however, include any additional flight data structures needed to support the specific data exchange. If such fields are required, they should be supplied via Extensions.

[3](https://www.fixm.aero/documents/FIXM%20Strategy.pdf) When newer versions of FIXM products are released, upgrading the restrictions only requires updating the reference to the newer versions and implementing the ad-hoc adaptations only for the parts that have changed.

[4](https://www.fixm.aero/releases/FIXM-4.1.0/FIXM_Core_v4_1_0_Modelling_Best_Practices.pdf) FIXM does not use GML but mimics it for geographic positions. GML encodes geographic locations as sequences of values since it employs the same construct to represent polygons.

[5](http://www.FIXM.aero) This design is intentional. It saves FIXM from being tied to an external standard for such a small use case and also aims to avoid potential issues or difficulties when marshalling / unmarshalling the standard xlink:href attribute.

[6](https://ost.eurocontrol.int/sites/FIXM/SitePages/Home.aspx) From draft Volume I of the ICAO Manual on SWIM (ICAO Doc 10039)

[7](https://www.fixm.aero/fixm_nas_extension_420.pl) See chapter 2.2.2 for a general introduction to the concept of Application Library.

[8] ***Publication Service***, ***Trial Service and Notification Service*** are not covered by this example.

[9] See in particular [10], Section 2.3.2, Figure 1

[10] Web Service Operations are used to couple a request and a reply. Service operations indicate the intent or the results of the information exchange.

[11] It is assumed that validation of the flight plan ensures when code ‘N’ is included in field 10a, no other code is included, but such validation is not part of the translation rules.

[12] It is assumed that validation of the flight plan ensures the field 10a code ‘R’ is always paired with field 18 PBN, but such validation is not part of the translation rules.

[13] It is assumed that validation of the flight plan ensures the field 10a code ‘Z’ is always paired with at least one of field 18 NAV, COM or DAT, but such validation is not part of the translation rules.

[14] It is assumed that validation of the flight plan ensures when code ‘N’ is included in field 10b, no other code is included, but such validation is not part of the translation rules.

[15] If field 18 DOF is omitted it is necessary to apply business rules to calculate the date of flight. Such business rules are outside the scope of this chapter. The responsibility lies with individual stakeholders.

[16] Note that each Extension must target the same version of FIXM Core and/or Application Library in order to be used together. For example, you cannot combine one Extension that uses FIXM Core 4.2.0 with another Extension that uses FIXM Core 4.1.0.

[17] Starting with the Basic Message v1.0.0 model and then deleting the BasicMessage package allows you to skip creating and setting up the Applications container and associated schema directory.

[18] The FIXM development team uses Sparx Systems Enterprise Architect version 13.5, build 1352 for all development work.

[19] Note that a package in Sparx EA can have more than one associated diagram. To date, FIXM products have only created one diagram per package, though.

[20] Though FIXM tends to use namespaces that look like URLs, the namespace value do not need to resolve to an actual location on the Internet. The namespace is just a field used to resolve naming collisions between schemas and should, therefore, be distinctive enough to ensure a reasonably high chance it is not used by another schema.

[21] This example used “xmg” but the prefix can be set to any value that makes sense in the context of your Application. Because it will be used throughout your generated schemas, a short prefix is typically preferred.

[22] If your toolbox is not showing XML Schema options, make sure you’ve applied the XSDschema stereotype to your package and close and then reopen the diagram.

[23] Note that position tags may need to be updated appropriately any time you add, remove, or rename an attribute or association. Be sure to check your position tags anytime you edit your model.

[24] This is important because XSD complex type restrictions, the mechanism used to create the templates, must use the same namespace as the types they restrict. Because of this, different versions of templates are not able to use distinct namespaces to distinguish themselves from each other. Tying the versioning of the templates to the versioning of the Application package and changing that versioning each time the templates change solves this issue.

[25] Note that these pastes must be performed back to back. If you go back and re-copy the package after the first paste, any additional sub-packages added will be copied as well and would then need to be manually deleted.

[26] Note, this only applies to the package itself – not the contents of the package. Creating the contents of the message template is covered in [Create Template Contents](#create-template-content).

[27] XML attributes, rarely used in this version of FIXM, are handled differently. If you wish to remove an XML attribute from your restricted class, the field must be retained but the “use” tag associated with the field must be set to a value of “prohibited”.

[28] If deleted, these associations will be removed entirely from the model, including Core! It is very important that you do not do this.

[29] It is important to note that this substitution of a restricted class for the class it is derived from can also be done for attributes. If your templates add restrictions to any classes defined in Core’s Base package, use them when specifying the *Type* field of your attributes as needed. The FficeMessage templates provide examples of this for the PersonOrOrganization class.

[30] Don’t forget to create any needed directories outside of Sparx EA to accommodate your schema structure.

[31] Note that it is very important to explicitly delete any unwanted connectors. If you instead delete the class in the diagram that the connector is attached to, this will remove the connector from the diagram but it will still exist within the model. This can cause undesired elements to be created when generating the physical model.

[32] The FIXM development team uses Sparx Systems Enterprise Architect version 13.5, build 1352 for all development work.

[33] Though FIXM tends to use namespaces that look like URLs, the namespace value do not need to resolve to an actual location on the Internet. The namespace is just a field used to resolve naming collisions between schemas and should, therefore, be distinctive enough to ensure a reasonably high chance it is not used by another schema.

[34] This example used “xmp” but the prefix can be set to any value that makes sense in the context of your extension. Because it will be used throughout your generated schemas, a short prefix is typically preferred.

[35] In FIXM, attributes are standardly used when the field you are adding is of a Type defined in the Core’s Base package. When defining your own types, they are standardly attached to a class by using an association instead. In this example, we will be adding a new field of type GeographicalPosition from the AeronauticalReference package under Base so using an attribute is the appropriate choice.

[36] The FIXM development team uses Sparx Systems Enterprise Architect version 13.5, build 1352 for all development work.

[37] This could be the root package of an entire FIXM product (for example, the Core package or the FficeMessage package under Applications) or a particular sub-section or individual sub-package (for example, the Base package under Core or the EnRoute package under Flight) of a FIXM product.

[38] This Perl script was developed using perl 5, version 16, subversion 3 (v5.16.3) built for x86\_64-linux-thread-multi and originally tested in a Linux environment (CentOS Linux 7 (Core)). It was also tested using perl 5, version 28, subversion 1 (v5.28.1) built for MSWin32-x64-multi-thread in a Windows environment (Windows 7 Enterprise (Service Pack 1)).

[39] Sometimes named ***Flight Plan Cancellation Message***

[40] This is commonly known as Request-Reply Message Exchange Pattern.

[41] This is commonly known as Publish-Subscribe Message Exchange Pattern.

[42] Web Service Operations are used to couple a request and a reply. Service operations indicate the intent or the results of the information exchange.