AIXM 5

Feature Identification and Reference

- use of xlink:href and UUID -

Aeronautical Information Exchange Model (AIXM)

For all inquiries, please contact:

Deborah COWELL - deborah.cowell@faa.gov

Eduard POROSNICU - eduard.porosnicu@eurocontrol.int

Edition No.		Edition Issue Date	Author	Reason for Change
0.1	First	2009	Design Team	First Edition
0.2	Proposed	2010	Design Team	Update
0.3	Proposed	06 Oct 2010	Design Team	Update Abstract references
0.4	Updated proposal	22 Nov 2010	Eurocontrol-FAA AIXM Design Team with the contribution of AIXM Forum members	Included guidelines for UUID generation. Updated based on comments and recommendations received from the AIXM Forum members and during the AIXM-XML Seminars of 2010.
0.5	Updated Proposal	21 Dec 2010		Updated following latest comments on the AIXM Forum. Use of “uuid” URN. Use of URN only for abstract references.
0.6	Updated	21 FEB 2011		Updated following discussions off-line with the most active (on this subject) AIXM Forum members
1.0	Released	29 APR 2011		Final release. Added a sentence in section 2.2 stressing the importance for UUID to be allocated by the real authoritative source for a feature’s data.

Table of Contents

1 Scope

1.1 Introduction

The Aeronautical Information Exchange Model (AIXM) is a GML 3.2 application schema meant to allow for the machine-to-machine exchange of aeronautical information in a structured format. As services that disseminate information in AIXM 5.1 to consumers are developed, the ability to manage the linkages between aeronautical features is key. This encompasses the concepts of feature identification and feature reference.

The AIXM 5.1 schema uses the XLink schema bundled with GML 3.2 for representing a reference between two features. In concert with the XLink standard, the XPointer standard may be used to address individual XML elements of messages. This document defines a number of standard cases for how XLinks should be used within an AIXM 5.1 message and how those XLinks should be resolved by applications.

The AIXM 5.1 schema also relies on the use of universal unique identifiers (UUID) as artificial identifiers for AIXM features. In fact, they do not identify the feature itself, but the data that represents that feature in digital aeronautical information management systems. This document provides guidance with regard to the algorithms that can be used for the generation of these UUID values.

1.2 References

[XLINK]	XLink v1.0 Specification http://www.w3.org/TR/xlink
[XPTR]	XPointer Specification http://www.w3.org/TR/xptr
[XPTH]	XPath Specification http://www.w3.org/TR/xpath
[UUID]	Universally unique identifier (theory) http://en.wikipedia.org/wiki/Universally_unique_identifier
[UUID-AIXM]	Universally Unique Identifier (UUID) Analysis, by Robert DeBlanc, MITRE, in support to FAA
[UUID-ISO]	ISO/IEC 9834-8, which is also ITU-T Rec x.667 http://www.itu.int/ITU-T/studygroups/com17/oid.html

1.3 Assumptions and Dependencies

This document assumes that the systems in question are communicating using messages or data sets that comply with the AIXM 5.0, 5.1 or later schema version.

2 Feature identification (UUID)

2.1 The gml:identifier property

Each AIXM Feature is identified through the use of the ***identifier*** property, which is inherited from the abstract AIXMFeature. In the AIXM XML Schema, this is mapped to the gml:identifier property, which all AIXM features inherit from the gml:DynamicFeatureType.

According to the AIXM Temporality Concept, the *identifier* property is the only time-invariant property; therefore it is situated outside the TimeSlice complex object, which encapsulates all the feature properties that can change in time. The gml:identifier property can be transmitted along with any Feature TimeSlice, allowing to identify the feature to which the TimeSlice belongs.

There are two essential requirements for the identifier property:

1. ***to be unique*** – there should exist a reasonable confidence that an identifier will never be unintentionally used by anyone for anything else;
2. ***to be universal*** – the same identifier should be used in all systems to identify a given AIXM Feature.

2.2 Use of UUID

The first requirement can be satisfied through the use of Universal Unique Identifiers (UUID). UUID generation algorithms can guarantee that the risk for the same UUID value to be generated by another system, for another feature, is extremely low. Information about such algorithms is provided in Appendix 1 of this document.

Concerning the second requirement, it is important to note that the identifier does not identify a feature. It identifies the data that someone has about a feature! In order to get maximum benefit from UUID, they should be generated by the primary originator (authoritative source) for that feature data.

Ideally, all stakeholders should have the same data about a given feature. However, as multiple “pseudo-primary” information sources may exist for the same data item or because the digital data transmission chain may be broken or duplicated, this cannot be guaranteed, at least on short term. Ensuring that the same gml:identifier is used in all systems for a given AIXM feature is a requirement for the information management process; therefore, it needs to be taken care through the process rules. From this point of view, UUIDs can indicate the continuity and the coherence of the data chain. If two systems use the same UUID for a feature, this indicates that either:

- they have the data from the same source (could be one of the two system, or a third one), or
- there are processes in place that ensure the consistency of the data between the two systems.

It is therefore possible that two or more information sets (list of TimeSlices) exists for the same AIXM feature, in two different systems, with different gml:identifier values. When data from different sources is merged in a single system, the owner of that system might be confronted with the need to identify and merge duplicate feature data, based on actual properties of the feature, not on the gml:identifier.

Overall, the most important advantage of using UUID as gml:identifier in AIXM is for software development. It is much simpler and less error prone to write code that relies on

UUID for feature identification and reference, as compared the use any kind of “natural key” combination.

2.3 UUID version and codeSpace

On the basis of the analysis presented in Appendix 1, ***the use of version 4 UUID based on random number generation, is recommended for AIXM***. An example of a gml:identifier using a UUID value is provided below.

```
<gml:identifier
     codeSpace="urn:uuid:">a82b3fc9-4aa4-4e67-8def-
aaea1ac595j</gml:identifier>
```

Information about UUID generation capabilities in common software is provided in Appendix 1, section A.1.9.

Note that a Uniform Resource Name (URN) is used as codeSpace for the gml:identifier. The ISO/IEC 9834-8 or RFC 4122 (http://www.ietf.org/rfc/rfc4122.txt) provide the UUID codeSpace value “urn:uuid:”.

2.4 The gml:id property

Every GML object is required to have a gml:id value, which is intended as a local unique identifier, within the XML data set. AIXM features also being GML objects, they must have a gml:id value as well. In addition, all other GML objects inside the feature (TimeSlice, gml:TimePeriod, gml:Point, aixm:SurfaceCharacteristics, aixm:AirspaceVolume, etc.) are also required to have a gml:id values, as shown in the example below:

```
<aixm:Airspace gml:id="…">
  <gml:identifier
     codeSpace="urn:uuid:">a82b3fc9-4aa4-4e67-8def-
aaea1ac595j</gml:identifier>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="…">
      <gml:validTime>
        <gml:TimePeriod gml:id="…">
          <gml:beginPosition>2010-06-29T17:31:00</gml:beginPosition>
          <gml:endPosition>2010-06-29T19:00:00</gml:endPosition>
        </gml:TimePeriod>
      </gml:validTime>
      <aixm:interpretation>BASELINE</aixm:interpretation>
      <aixm:sequenceNumber>1</aixm:sequenceNumber>
      <aixm:type>D</aixm:type>
      <aixm:geometryComponent>
        <aixm:AirspaceGeometryComponent gml:id="…">
          <aixm:theAirspaceVolume>
            <aixm:AirspaceVolume gml:id="…">
              <aixm:upperLimit uom="FT">500</aixm:upperLimit>
              <aixm:upperLimitReference>MSL</aixm:upperLimitReference>
              <aixm:lowerLimit uom="FT">GND</aixm:lowerLimit>
              <aixm:lowerLimitReference>MSL</aixm:lowerLimitReference>
              <aixm:horizontalProjection>
              <aixm:Surface gml:id="…">
                <gml:patches>
                  <gml:PolygonPatch>
                    <gml:exterior>
              ...
```

```
</aixm:Airspace>
```

The gml:id value has to comply with the same rules as any other XML ID attribute: to be unique within the XML file, to start with a letter, etc.

It is recommended that the gml:id of the AIXM features (such as aixm:Airspace, aixm:Runway, etc.) also make use of the UUID value, prefixed with “uuid.”. Since UUID are globally unique, they are also locally unique and thus perfect candidates for gml:id. Attention, this recommendation concerns only the AIXM Feature level, not the lower levels, such as aixm:AirspaceTimeSlice, etc. The use of the feature UUID as gml:id will facilitate the implementation of concrete xlink:href references using the direct ‘#ID’ syntax, as explained in 3.1.

Applied to the previous example, these recommendations give the following gml:id values:

```
<aixm:Airspace gml:id="uuid.a82b3fc9-4aa4-4e67-8def-aaea1ac595j">
  <gml:identifier
     codeSpace="urn:uuid:">a82b3fc9-4aa4-4e67-8def-
aaea1ac595j</gml:identifier>
  <aixm:timeSlice>
    <aixm:AirspaceTimeSlice gml:id="ID00001">
      <gml:validTime>
        <gml:TimePeriod gml:id="ID00002">
          <gml:beginPosition>2010-06-29T17:31:00</gml:beginPosition>
          <gml:endPosition>2010-06-29T19:00:00</gml:endPosition>
        </gml:TimePeriod>
      </gml:validTime>
      <aixm:interpretation>BASELINE</aixm:interpretation>
      <aixm:sequenceNumber>1</aixm:sequenceNumber>
      <aixm:type>D</aixm:type>
      <aixm:geometryComponent>
        <aixm:AirspaceGeometryComponent gml:id="ID00003">
          <aixm:theAirspaceVolume>
            <aixm:AirspaceVolume gml:id="ID00004">
              <aixm:upperLimit uom="FT">500</aixm:upperLimit>
              <aixm:upperLimitReference>MSL</aixm:upperLimitReference>
              <aixm:lowerLimit uom="FT">GND</aixm:lowerLimit>
              <aixm:lowerLimitReference>MSL</aixm:lowerLimitReference>
              <aixm:horizontalProjection>
              <aixm:Surface gml:id="ID00005">
                <gml:patches>
                  <gml:PolygonPatch>
                    <gml:exterior>
              ...

</aixm:Airspace>
```

3 Feature Reference (xlink:href)

3.1 Introduction

Associations between AIXM Features are implemented in the AIXM XML Schema through the use of XLinks [XLINK].

The general recommendation is to use the gml:identifier (UUID) of the referenced AIXM Feature. This supports many commercial solutions out-of-the-box, as it relies entirely on the XLink, XPointer and XPath standards. It is critical that every XLink value points to the correct feature of interest. For example, that an xlink value identifying an Airspace holds the identifier of an Airspace feature, not the one of a Runway or another feature, message, etc. However, this cannot be ensured by the XML Schema and it needs to be dealt with as part of the data validation rules.

If the UUID is not available, a natural key search can be used. In general, the verbosity of such a request may increase as finding the key will require querying into a TimeSlice of the AIXM Feature.

From the point of view of the Xlink target, there exist three cases of interest:

1. Concrete local references within a data set;
2. Concrete external references resolved via web services;
3. Abstract references to be resolved by the consuming application.

These are described in more detail in the following sub-sections.

3.2 Concrete local references within a message

In some cases, services producing AIXM 5.1 data will provide data sets in which all the referenced features are included. Instead of a reference by gml:identifier, a simpler local reference to the gml:id attribute could be used in this case. The gml:id is, by definition, unique within an XML file. Therefore, when used for local references, it is unambiguous. The gml:id attributes are defined as identifiers in the schema and can be indexed during parsing.

An example is provided below. Note that the recommendation made in 2.4 is applied here, which means the gml:id of the Airspace feature is actually based on the UUID value of the gml:Identifier.

```
<aixm:Airspace gml:id="uuid.:a82b3fc9-4aa4-4e67-8def-aaea1ac595j">
  <gml:identifier
     codeSpace="urn:uuid:">a82b3fc9-4aa4-4e67-8def-
aaea1ac595j</gml:identifier>

    ...

</aixm:Airspace>
...
<aixm:AirTrafficControlService gml:id="uuid.d4d33081-54ad-4c1a-9519-
b5b67de561ae">
  <aixm:timeSlice>
    <aixm:AirTrafficControlServiceTimeSlice
gml:id="AirTrafficControlService01_TS1">
      <gml:validTime>
        <gml:TimePeriod gml:id="AirTrafficControlService01_TS1_TP1">
          <gml:beginPosition>2008-01-01T00:00:00</gml:beginPosition>
          <gml:endPosition indeterminatePosition="unknown"/>
```

```
        </gml:TimePeriod>
      </gml:validTime>
      <aixm:interpretation>BASELINE</aixm:interpretation>
      <aixm:type>ACS</aixm:type>
      <aixm:clientAirspace xlink:href="#uuid.a82b3fc9-4aa4-4e67-8def-
aaea1ac595j"/>
    </aixm:AirTrafficControlServiceTimeSlice>
  </aixm:timeSlice>
</aixm:AirTrafficControlService>
```

3.3 Concrete external references

When information about features is exposed via web services, a method by which XLinks can be resolved is via a Universal Resource Locator (URL).

In this case, the expectation is that the consumer of the message can follow the URL provided in the XLink directly and the hashtag will identify the feature within the resultant resource to which this reference resolves. If the recommendation made in 2.4 was applied and the target feature has a UUID based gml:id, then the concrete reference may be encoded using simply the '#ID' reference syntax, as in the example below:

```
      <aixm:clientAirspace
xlink:href="http://aim.faa.gov/services/AirspaceService#uuid.a82b3fc9-
4aa4-4e67-8def-aaea1ac595j"/>
```

A more general, but also much more complex solution is to use xpointer. Again, the combination of the URL and the XPointer should result in the tag which resolves to the feature which is referenced.

```
      <aixm:clientAirspace
xlink:href="http://aim.faa.gov/services/AirspaceService?get=a82b3fc9-4aa4-
4e67-8def-
aaea1ac595j#xmlns(ns1=http://www.opengis.net/gml/3.2)xmlns(ns2=http://www.
aixm.aero/schema/5.1)xpointer(//ns2:Airspace[ns1:identifier='a82b3fc9-
4aa4-4e67-8def-aaea1ac595j'])"/>
```

Note that the above URL is independent of implementation. It may identify a (Web Feature Server) WFS server, but it could also be an implementation of a simple web service which returns a particular set of features based on the user's query. As long as the resolution of the URL produces an XML document which contains the AIXM Feature, the above is a valid reference.

The approach for the first two cases (concrete local or external references) is based purely on the XLink and XPointer standard: there is no application-specific logic required to resolve the reference. However, implementers should take care to take advantage of caching strategies to avoid continually resolving features for which they already have definitions.

3.4 Abstract references

Xlink assumes a "resource centered approach", in which the XML document or fragment is a resource which can be referenced from anywhere. It assumes that the resource is available on the web in a single, definitive copy. In the aeronautical information domain, the feature is an entity which can be globally referenced but there are many representations of that feature in circulation. Many of those representations are in AIXM messages; others are in databases or applications. This is why the use of concrete references is rare and the use of abstract references needs to be considered as well.

Abstract references fit very well to the GML paradigm since they provide a reference to the abstract idea of the feature, rather than one of its representations. Resolving the abstract

reference to a physical one is the problem of the application and it can deal with issues of multiple copies in all the messages, databases etc. that it knows about.

Leaving the resolution with the application also allows the application to adapt to its context. For example, if the application is off-line it could choose to use a locally cached copy of the feature. An on-line application could go to the definitive web-service for the feature.

In those cases, the xlink:href should use a universal resource name (URN) rather than a URL; note that URNs, unlike URLs, cannot be directly used to find a resource. In the case where a URN is presented to the consumer, it is the responsibility of the consuming application to resolve the reference.

Nothing in the use of a URN implies the availability of the referenced feature or its location; it may be that the feature is defined locally within the message, accessible remotely by a web service, or directly through a database access.

In general, the following three steps will be followed:

1. The recipient of the data will use the identifier of the referenced feature to search its local database.
2. If no such feature exists in its local data set, the incoming data set would be searched for the referenced feature.
3. If the above does not find the feature, the system would search known data sources to resolve the reference.

This type of reference is limiting in its openness, as it requires application logic to be resolved. As such, its use is expected to be reduced over time, as the aeronautical information domain moves towards Web service solutions. By using the concrete standards mentioned above, any standards-complete system can resolve AIXM references without additional application logic.

3.4.1 Using UUID

It is recommended that the URN is based on the UUID of the referenced feature, as in the following example

```
    <aixm:clientAirspace xlink:href="urn:uuid:a82b3fc9-4aa4-4e67-8def-aaea1ac595j"/>
```

In this case, the application will consume the URN-based locater and internally discover the definition of the referenced airspace through data registries, manual coding, or other methods specific to the systems involved. The beginning of the URN should match the codeSpace used for gml:identifier.

3.4.2 Using natural keys

When the UUIDs are not available, an AIXM specific URN composed with natural keys could be used, as in the example below:

```
<aixm:clientAirspace xlink:href="urn:aixm:Airspace(gml:timePosition=2010-04-07T09:00;aixm:type=D;aixm:designator=EBD25A)"/>
```

Apart from being much more complex than those based on UUID, the disadvantage of the URN based on natural keys is that they requires specific code for decoding and identifying the target feature. Therefore, URN with natural keys should be used only during transition periods and on a limited scale. For example, it might be a solution to use natural key based URN between systems that store the data in legacy formats and which do not have the possibility to work with UUID values..

The following rule shall be applied in the composition of the "aixm" URN:

urn:aixm:***Feature***(timePosition=***time_value***;***property_name*=*property_value***;…)

where:

- ***Feature*** is the name of an AIXM Feature as defined in the AIXM XML Schema; for example: AirportHeliport, Airspace, Runway, etc.
- ***time_value*** is a UTC date and time value in the format yyyy-mm-ddThh:mm and it indicates the moment in time at which the BASELINE TimeSlice of the AIXM Feature referred had the ***property_value*(s)** that appear in the URN composition;
- ***property_name*** is the name of property of an AIXM Feature that composes a natural key for that feature; the property name shall be spelled according to the AIXM XML Schema, exactly as it appears as child element of the AIXM Feature TimeSlice.

  It is assumed that all natural key properties are either direct child elements of the Feature TimeSlice or that they exist only once in the XML tree (such as the gml:pos of a Navaid).
- ***property_value*** is the value of the of the property identified by ***property_name*** as defined in the BASELINE TimeSlice of that feature that is valid at the date and time specific by the ***time_value***;

  In some situations, this includes properties that do not directly have a value, but have an xlink:href attribute that points to another Feature. In this case, the URN of the referenced feature shall be used as property_value. A typical example is the Runway feature, for which the natural key includes the AirportHeliport where it is located. The URN will look like in the example below:

```
urn:aixm:Runway(gml:timePosition=2010-12-
20T16:32;aixm:designator=02%2F20;aixm:associatedAirportHelipor
t=urn:aixm:AirportHeliport(gml:timePosition=          2010-12-
20T16:32;aixm:designator=EBBR))
```

  Note that the value of an AIXM feature natural key property could contain characters that are not allowed in the composition of the URN, as explained in the URN Syntax (RFC 2141) and have to be replaced with their hex code, prefixed by the character "%". This is the case for the "/" character, which is typically used in runway designators. Therefore, in the example above, the runway designator "02/20" was encoded as "02%2F20", where "%2F" is the hex representation of "/".

In order to be valid URN in terms of the RFC 2141 (URN Syntax), the "aixm" URN would have to be registered with the Internet Assigned Numbers Authority. Until then, it shall be considered as a non-standard (experimental) URN.

3.5 Use of xlink:title

The value of the attribute xlink:title on a xlink is a human-readable description of the referenced value. In this case, it would be a human-friendly description of the referenced aeronautical feature.

It is suggested that the xlink:title be used, especially in cases in which the referenced feature is defined remotely. The title should be a human-friendly name of the feature which can be used internally by applications for display purposes. It is discouraged to use the xlink:title for automatic feature identification.

```
      <aixm:clientAirspace xlink:href="urn:uuid:a82b3fc9-4aa4-4e67-8def-
aaea1ac595j" xlink:title="Gabbs North MOA"/>
```

Appendix 1

A.1. UUID algorithms

A.1.1 Introduction

This Appendix examines the four main versions of the UUID; how they are generated, their efficiency and their computational overhead. On the basis of this comparison it is found that the version 4 UUIDI, based on random number generation, is the most efficient. The tendency in the industry is to move toward the use of version 4, with the notable exception of Oracle that remains on version 1. A survey of the UUID/Globally Unique Identifier (GUID) versions supported by the main software products and libraries is summarized in Figure 7.

A.1.2 Definition and Uniqueness

A UUID is a 128-bit number that is encoded either with a random number, the output of a cryptographic hash function or a combination of a random number and the time of generation. UUIDs are conventionally written or displayed in their 'canonical' form, which is a sequence of 32 hexadecimal digits grouped into a sequence of 8, 4, 4, 4 and 12 digits; an example is given below.

550e8400-e29b-41d4-a716-446655440000

The left-most digit of this string represents the most significant four bits of the UUID.

The purpose of the UUID is best described by Wikipedia http://en.wikipedia.org/wiki/UUID as: "*The intent of UUIDs is to enable distributed systems to uniquely identify information without significant central coordination. ..... …..Anyone can create a UUID and use it to identify something with reasonable confidence that the identifier will never be unintentionally used by anyone for anything else.*" The UUID is defined in three compatible standards: ISO/IEC 11578:1996, ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122.

The reason for the use of a number field as large as [1-2^{122}] is that the UUID is ***universal***, i.e. the probability of a duplicate UUID being encountered within the IT universe for the foreseeable future must be a very low.

A.1.3 Format and Versions

The UUID format is depicted in Figure 1. Six of the 128 bits are used to specify the type of UUID leaving 122 to carry a random number, the output of a cryptographic hash function or a combination of a Network Interface Card (NIC) MAC address and the time of generation, depending on the Version of the UUID. The six control bits are very awkwardly placed, and are not even contiguous, making the generation and interpretation of UUIDs more complicated than it need have been. The digits are numbered from the left-most digit of the canonical (string) form. The type of UUID is defined first by the value of the Variant field (YY) which occupies the most significant two bits of the 17th hex digit.

Note: *RFC 4122, paragraph 4.1.1 shows the variant field as having three bits. For all standard UUIDs the least significant bit of these three is irrelevant; it is used as the top bit of the clock sequence in version 1 UUIDs, or is part of the random or hash fields in version 3,4 or 5 UUIDs.*

A value of 2 in the variant field indicates that the UUID is one of the standard versions. A zero value of the most significant bit of the variant field indicates that the UUID was generated by a workstation in the Apollo Network Computing System. A value 3 of the

variant field indicates a UUID that was used by .COM in versions of Microsoft Windows before Windows 2000. The V*ersion* field (VVVV) is coded into the 13th digit allowing up to 15 UUID versions. Five versions, numbered 1 – 5 currently exist.

Hex Digits	1	2	3	4	5	6	7	8
1 to 8	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
9 to 16	XXXX	XXXX	XXXX	XXXX	**VVVV**	XXXX	XXXX	XXXX
17 to 24	**YY**XX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
25 to 31	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

Figure 1 - General Format of UUID

The UUID standards do not require that the UUIDs generated by a given system must all be of the same version. A closed (i.e. not universal) distributed system, could exploit this by allowing any of the four main versions to be generated. The degree of uniqueness of the UUIDs would then be quadrupled, from a field of [1-2^{122}] an effective field of [1-2^{124}].

A.1.4 Version 1 UUID

The version 1 format is the oldest and most complicated, but is still widely used; the encoding in shown in Figure 2. The 60 bits labeled 't' are the time at which the UUID was created, in increments of 100 nano-seconds ($1x10^{-7}$) seconds. The 48 bits labeled 'm' are the (universally unique) MAC address of one of the Network Interface Cards (NIC) on the system; when this is not available a 48-bit random or pseudo-random number is generated. The 14 clock-sequence bits labeled 'c' are used to reduce the possibility of a duplicate UUID value being generated following the clock being set backwards (after a power outage) or the NIC card being changed. The field allows for 2^{14} or >16,000 resets in the lifetime of the system. In the unlikely event that the clock sequence just before the event is known, it can be used on recovery and just incremented; otherwise the clock sequence is re-initialized with a random number.

Hex Digits	1	2	3	4	5	6	7	8
1 to 8	tttt	tttt	tttt	tttt	tttt	tttt	tttt	tttt
9 to 16	tttt	tttt	tttt	tttt	**0001**	tttt	tttt	tttt
17 to 24	**10**cc	cccc	cccc	cccc	mmmm	mmmm	mmmm	mmmm
25 to 31	mmmm	mmmm	mmmm	mmmm	mmmm	mmmm	mmmm	mmmm

Figure 2 - Format of Version 1 UUID – Time and Address

The time field in a version 1 UUID provides the elapsed time, in 100 nanosecond increments, since the start of the Gregorian calendar in October 1582. The maximum

(unsigned) value of 2^{60} nanoseconds is equal to about 3663 years. Currently the highest order bit set in a version 1 UUID time field is the 57th bit. The 58th and higher bits will not be used until after 2444.

A.1.5 Version 2 UUID

Version 2 UUID is used in the IEEE Portable Operating System Interface POSIX Distributed Computing Environment. The format is very similar to version 1.

A.1.6 Version 4 UUID

The encoding of a version 4 UUID is shown in Figure 3. The 122 bits labeled 'r' comprise a pseudo-random number, or preferably a cryptographic quality random number.

Hex Digits	1	2	3	4	5	6	7	8
1 to 8	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
9 to 16	rrrr	rrrr	rrrr	rrrr	**0100**	rrrr	rrrr	rrrr
17 to 24	**10**rr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr
25 to 31	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr	rrrr

Figure 3 - Format of Version 4 UUID – Random Number

After a distributed system starts generating version 4 UUIDs, if a source of cryptographic quality random numbers is used, the probability that it will generate a duplicate UUID of one already generated p(n;d) is given by the expression $1 - e^{-nxn/2d}$ where the number already generated is 2^n and d is the number of bits occupied by the random number. The probability of a collision will be higher if pseudo-random numbers generated by the system are used.

As an example, after 243 (≈8.8 trillion) UUIDs have been generated, the probability of the next UUID being a duplicate of one already generated is ≈ $7.276x10^{-12}$. Figure 4 gives p(n;d) for values of n: 36, 41, 43 and 46 and values of d: 90, 106 and 122. {calculated with R Statistical Language}. The fourth column of the table shows that taking 4 octets of the random number field for another use would introduce a significant probability of collisions occurring.

	d = 122	d = 106	d = 90
n = 36	$4.44x10^{-16}$	$2.91x10^{-11}$	$1.19x10^{-6}$
n = 41	$4.55x10^{-13}$	$2.98x10^{-8}$	$1.95x10^{-3}$
n = 43	$7.28x10^{-12}$	$4.77x10^{-7}$	$3.08x10^{-2}$
n = 46	$4.66x10^{-10}$	$3.05x10^{-5}$	$8.65x10^{-1}$

Figure 4 - Probability of Collision for Version 4 (Random Number) UUIDs

A.1.7 Versions 3 and 5 UUID

The encoding of a version 3 or version 5 UUID is shown in Figure 5. The bits labeled 'h' are the output of an MD5 cryptographic hash function in version 3 UUIDs and are the output of a SHA-1 cryptographic hash function in version 5 UUIDs. The hash function values of 128 bits (MD5) and 160 bits (SHA-1) are truncated to the 122 bits available in the UUID.

Hex Digits	1	2	3	4	5	6	7	8
1 to 8	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh
9 to 16	hhhh	hhhh	hhhh	hhhh	**0011/0101**	hhhh	hhhh	hhhh
17 to 24	**10**rr	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh
25 to 31	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh	hhhh

Figure 5 - Format of Version 3 or Version 5 UUID

Information about the MD5 algorithm can be found at http://www.w3.org/TR/1998/REC-DSig-label/MD5-1_0, and about the SHA-1 algorithm at http://www.itl.nist.gov/fipspubs/fip180-1.htm. Both of these algorithms have been 'broken' by systematic attacks; in the case of MD5 within 2^{43} hash operations and in the case of SHA-1 within 2^{63} hash operations. These are well below the number of operation that can be guaranteed to produce a collision, 2^{64} and 2^{80} respectively. However this has no implications for the use of hash numbers in UUIDs, where they are used more for convenience than for security, other than to suggest that the uniqueness of a UUID formed with either of these algorithms will be less than that of a version 4 UUID formed with a random number of cryptographic quality.

A.1.8 Comparisons of UUID Versions

Efficiency of bit usage

The time algorithm used in version 1 UUIDs is inefficient in its granularity of 100 nanoseconds. The 56th bit of the 60-bit time field was not used until some time in 1986, and the 57th bit will not be used until 2444. This means that the top 4 bits are effectively not being used. Also, the 14 bit clock sequence field in version 1 UUIDs is only used each time the system is powered up, allowing for 2^{14} or >16,000 resets in the lifetime of the system. A lower number of bits would provide for a sufficient number of resets, given that a recycling of this field would not produce UUIDs with an ambiguous time of generation.

In contrast to version 1 UUIDs, in versions 3, 4 and 5 UUIDs all 122 available bits are used to hold either a random number or the output of a hash function.

Computational overhead

An April 2007 test to compare the times taken to generate version 1, version 3 and version 4 UUIDs { http://johannburkard.de/blog/programming/java/Java-UUID-generators-compared.html } showed the following average times to generate 1 million UUIDs with standard Java software:

Version 1 Time based	5,432 milliseconds
Version 3 MD5 based	40, 788 milliseconds
Version 4 Random number based	48, 900 milliseconds

Figure 6 - Average times for UUID generation

These results show that versions 3 and 4 UUIDs incur substantially higher processing overhead than version 1 UUIDs: by a factor of about 7.5 for version 4 and a factor of about 9 for version 3.

The SHA-1 algorithm generates a 160 bit hash compared to the 128 bits generated by the MD5 algorithm. When used in a UUID, both hashes are truncated to 122 bits, so any advantage in security or randomness that the SHA-1 might have over MD5 is obviated when it is used in a UUID.

Uniqueness

The timestamp field in the version 1 UUID is not a random number. Each bit changes with half the frequency of the preceding, less significant bit; the 48^{th} and higher bits change less than once a year. More significant is the devotion of 14 bits to the clock sequence field, which is only incremented each time a system is switched back on. The Version 1 UUIDs are therefore substantially less unique than either Version 3 or Version 4 UUIDs.

The uniqueness of version 3 or 5 UUIDs can not be greater than the uniqueness of version 4 UUIDs and may be less, given the fact that they can both be compromised after a number of uses that is well below the number at which duplication could be expected ("brute force attack"): 2^{43} instead of 2^{64} in the case of MD5 and 2^{63} instead of 2^{80} in the case of SHA-1.

A.1.9 Support by Common Software

It is sometimes difficult to establish what versions of UUID are supported by a particular software application, as many users and even product announcements do not seem to be aware that five distinct versions exist. Typically the output of a UUID generator is only described as several groups of hexadecimal digits. Oracle is notable in this respect, simply defining its UUID as a 16 byte raw value. In the case of Microsoft, it was difficult to untangle because UUIDs are used extensively within Microsoft packages; only one of many interfaces in the Microsoft Component Object Model (COM), and now in the .NET framework, use the type of UUIDs included in this study. Microsoft switched from using version 1 to version 4 with the introduction of .NET and Windows 2000.

Although a particular application package may not support a given version of a UUID, in many cases an extension to support that version can be added from libraries such as the UNIX-based Open Source Software Project (OSSP).

	Version 1	Version 3	Version 4	Version 5	Variant
Microsoft .COM					3
Microsoft Windows 2000			X		
Oracle	X				
Java (JUG)	X	X	X	X	
Java J2SE5		X	X		

JavaScript **uuid.js**	**X**				
Linux (ossp)	**X**	**X**	**X**	**X**	
MySQL **Version 1**	**X**				
MySQL **current**			**X**		
PostgreSQL			**X**		
Apache **(Jakarta project)**	**X**	**X**	**X**	**X**	
OpenPKG	**X**	**X**	**X**		
Python	**X**	**X**	**X**	**X**	
Ruby	**X**	**X**	**X**	**X**	
C++ (ossp)	**X**	**X**	**X**	**X**	

Figure 7 - UUID Versions Supported by Common Software