

Feature Identities, Descriptors and Handles

Philip Sargent

Laser-Scan Ltd.

Cambridge CB4 4FY, UK

Philip.Sargent@computer.org

<http://www.sargents.demon.co.uk/Philip/>

Abstract. Finding the “right” geographic feature is a common source of interoperability difficulties. This paper reviews the issues and discusses how *persistent feature identifiers* can be used to support relationships and incremental updating in dispersed inter-operating information systems. Using such identifiers requires common definitions for concepts such as “scope” of datasets and identifier namespaces. This work extends current understanding in the Features Special Interest Group of the Open GIS Consortium (OGC).¹

1 Introduction

Compared with the requirements of an individual analyst, operational use of geographic information in a multi-user, multi-organisation application, adds significant new requirements in data maintenance, data transformation, lineage tracking, schema maintenance and metadata update.

These additional functions tend to involve several datasets with specific relationships, e.g. one dataset may be a prior version of another. We then require some way of tracking individual features across those datasets. The standard example is where some attributes of a feature are under the update authority of a different organisation from other attributes; but somehow all parties must agree that they are referring to the correct feature even if many (or all) the attribute values change.

A study of feature identity presupposes that we will be dealing with moderately persistent real world objects which are observable as distinct entities (at least for a while): entities that exist long enough to be worth naming and talking about. Thus this paper is firmly placed in the “object” rather than the “field” tradition of GIS, with the proviso that some of these objects may have indistinct boundaries [1] and may be temporary, e.g. sandbanks, storms and forest fires.

This paper attempts to review conceptual structures which may underpin future interoperability standards. File data formats have a relatively short useful life compared to the life of the data they transport and “standard” function interfaces have even shorter lives, but the data model has a much longer life: almost as long as that of the data itself.

¹ Submitted to Interop'99 – The 2nd International Conference on Interoperating Geographic Information Systems, Zurich, March 10-12, 1999

2 Background

2.1 Conceptual Data Model

It is necessary to outline a conceptual data model in which to frame the spaces and domains under discussion. The following is a simplified version of the Open GIS Consortium (OGC) conceptual data model [2]:

Real World: the entire world in objective reality

Conceptual World: the observed subset of the real world

Geospatial World: a categorisation and classification of that subset

Dimensional World: the classified entities with metric representations and spatial reference systems, but not yet represented in any software system

Project World: the entities in a logical schema defined by a particular information community

Software Worlds: a set of representations of the entities in an overlapping set of increasingly capable software systems with defined schemas

In this paper a “feature” will be taken to be a *software representation* of a real-world object [2], e.g. a lake, road or city, which can have associated with it a number of attributes, some of which are *geometric representations* (“geometries”), i.e. shapes with locations. (Note that this definition differs from a commonly understood meaning where a feature *is* the geometric representation in a spatial reference system.) Thus a school is represented as a feature with one associated complex geometry which is the set of polygons representing the floor plans of the buildings and another that is the boundary of the site.

If we examine the conceptual data model sketched out above, we will see that unique labelling can only be done for discrete objects which are already a categorisation of a subset of the real world.

2.2 Labelling the Real World

The first hurdle to overcome is the disbelief of those who think that suggesting the use of feature identifiers means that we must label and index everything in the real world. That is clearly infeasible and for almost all purposes we cannot assume that such an index exists.

However there are organisations which do maintain unique identifiers for a great many types of real world objects, e.g. road bridges are numbered within the jurisdiction of a local government’s civil works department, telegraph poles are labelled by the local telecommunications organisation, and every computer Ethernet card has a unique number burned in to it during manufacture (we know they are unique even if we don’t know where they are). These labelled real world objects are almost invariably man-made or even entirely man-imagined, e.g. land parcel identifications, for the simple reason that natural objects usually permit less precise delineation, e.g. is an estuary part of the coast or the riverbank? This indeterminacy is of several distinct types and has been discussed in detail in a recent conference [1].

The types of feature identity that will be introduced later in this paper must be able to interwork with these pre-existing labelling schemes that are maintained by a variety of organisations with very different construction grammars and quality control standards. A key point is that the labels have to be maintained: mistakes happen and must be corrected, incorrect numbers are applied to real objects and real labels are misrecorded in software systems. Thus the real world label must be related to, but distinct from, any geographic information system's *feature identity*. For these reasons it is clear that real world labels can never substitute completely for some purposes, even though such labels probably provide more added value than other types of feature identity.

2.3 Practicalities

Practical interoperability has to take account of pre-existing data which may have been constructed using entirely different semantic principles. In the case of feature identity, the task in hand is not so much to provide support for those existing dataset collections which provide sophisticated persistent identifiers as to provide mechanisms whereby datasets without persistent identifiers can nevertheless offer useful services as if they did. For example, a good standard should not preclude the possibility that identity is constructed using a key based on one or more attribute values (as used in many RDBMSs), but neither should it mandate such an approach since many other GISs have a concept of identity which cannot be adequately represented by that approach.

The process of standardisation introduces its own oddities and restrictions which are not present in either commercial or academic research software. In addition, the goals themselves are different from most academic work in GIS. University research tends to look for proof that an innovative technique is feasible, elegant and efficient, irrespective of its compatibility with existing systems, whereas standardization research has to bring the bulk of the existing commercial implementations with it. Thus introducing new concepts has to be done with great care:

- find the *minimum* number of new concepts required
- consider whether these concepts need to be reified as software objects;
- if so reified, whether these objects should be named, and
- if so named, over what namespace the names should be unique.

A “namespace” is a concept which has been found increasingly useful in the design of computer languages. The C++ language recently elevated it to be an integral part of the language, and the Python language is fundamentally designed around it [3].

Thus while a concept may be universally agreed as being a useful aid in structuring a problem area, it can be nevertheless be productive avoid naming or reifying it because that can then avoid an entire harmonisation argument.

For those objects (concepts) we are going to name individually we then consider how we might “get one” and what we could do with it once we’ve got it,

i.e. what other object might be a factory for it, and what operations we might want to perform on it or with it.

Example The technique of reducing conflict by avoiding the reification of a concept is demonstrated if the OGC concept of ‘Feature Type’.

The OGC abstract specification defines the concept of *feature type* which specifies the attributes (properties) that a feature of that type can have, but the OGC Simple Features for SQL specification avoids making that concept into a manipulatable object and instead each *individual feature* can be queried as to what attributes it supports. This avoids introducing a new type into the function interface and avoids introducing a new namespace, the list of names of feature types, with its own uniqueness constraints. The savings in elapsed time to produce the standard and to test proposed implementations for conformance more than makes up for the slight inconvenience of not having a standard for *feature type* itself (which could be introduced into later standards if absolutely required).

2.4 Dataset

A “dataset” has an obvious meaning when a simple GIS organises its persistent storage as files. However, large and complex GIS applications involve databases and possibly a large number of files for import, distributed update, etc. Thus we need a tighter definition or a different concept entirely.

The OGC uses the term “Feature Collection” to mean an object which represents a collection of features but which also may support its own attributes. The fundamental operation on a feature collection is to make available (in some way) the features in the collection. Feature collections may be permanent, e.g. a dataset, or transient, e.g. the result of a query. The metadata of a dataset thus become attributes of a feature collection. Compatibility with other standards for metadata *content* can then cause problems because some which come from bibliographic communities allow “repeated fields” with different data, whereas most GIS data models for features (and an OGC feature collection is a variety of feature) allows only “name = value” semantics.

The existing OGC “Simple Features” specifications do not need to go into details such as defining the *feature type* of a *feature collection* [4] but the OGC Abstract Specification suggests that any feature could belong to several *feature collections* at once.

2.5 Schema

Many of the queries that one wishes to perform on a dataset logically should be queries addressed to the dataset’s *schema*, e.g. questions as to what *feature types* the dataset holds, what attribute names and types are defined for each *feature type*, etc. If we consider a set of related datasets, e.g. a set of versions, then

we can see that the schema itself has a broader existence than each individual dataset and might be better identified with a “scope” (Sect. 3.2).

Clearly if versions of the “same feature” may be found in several different datasets, it should have some of the schema in common between them (but datasets may not necessarily have the same internal structure: versioning should be able to cope with restructuring, e.g. of a directory tree [5]).

Consider for a moment the case where we have a real world *feature description*, in this case the different feature representations of the real entity may have *nothing* in common: for example, the London suburb “Richmond” appears as a feature in both the London Tube map and as part of the UK postal code coverage, but these have nothing in common and it is sensible to consider them as two different features (software representations).

Initially it would be simpler to just assert that the schemas must be identical in all datasets in the same scope where a feature identifier may be used. However we must bear in mind that serious geographic information applications are approaching 24 hour – 365 day operation, so some allowance for dynamic schema evolution will certainly be needed in the near future.

2.6 Lineage and Metadata

An important relationship between datasets which are intended to share feature identity scopes is “lineage”: the history of data. Lineage should be described in the metadata of a dataset: “the currency, accuracy, data content and attributes, sources, prices, coverage, and suitability for a particular use” [6].

For feature identity management we need something more precise than the rather loose semantics and grammar, defined only in natural language descriptions, which are commonly used in metadata descriptions. If a dataset is composed of several persistent *feature collections* then each could contain its own metadata, and in the limit of granularity, every individual feature could contain metadata on how it was constructed and under what conditions its attribute values were originally measured.

There are geographic-specific metadata protocols and systems [7] as well as names and types [6], but the recent enormous growth in non-geographic metadata protocols such as RDF [8] implies that the GIS-specific protocols will have a short life before they are merged into the mainstream.

3 How Many Varieties of Identity?

3.1 Descriptors and Handles

We have the concepts *feature* and *dataset* (*feature collection*). We think we need the concept of *feature identifier*, but some thought will show that we need two such concepts: a *feature descriptor* and a *feature handle*. A *descriptor* is some way of specifying a feature from “outside the system”, by listing some sufficiently unique combination of attribute values, where the list of attributes will be highly

application dependent. If an external agency maintains real world labels, then a single attribute value may be sufficient: but from the software designer's point of view the uniqueness of such labels cannot be entirely relied upon. A *feature handle*, however, is a concept that is "inside" the software system and which is required to have quite tightly defined uniqueness properties which are enforced by the software itself.

Feature handles should generally be considered "opaque" and it might not even be sensible to think of them as "values" at all. Some handles might be string of text which is a query (in any language) which is sufficient to retrieve the feature itself.

3.2 Scopes

Scope is a dataset management and metadata issue. A scope is a unit within which updating management can occur and probably the level at which schemas are defined. A scope is a collection of software and data in which a *feature handle* has meaning. If we are going to insist on uniqueness, we must define scope.

There are two interpretations for scope, the first broader than the second:

Meaning Within a scope, a *feature handle* has meaning.

Reachability A scope is a "domain of reachability" in that a reference (a *feature handle*) from a feature to another feature can "reach" another, e.g. to imply a relationship.

A scope to be used for update could be larger than a domain of reachability, i.e. a system may allow references *from* a feature to another only within a part of the scope in which that reference (feature handle) has meaning, but might allow corrections to be made to anything it knows about. (Existing examples are systems which allow topological relationships only between features in the same thematic layer.)

Even at our limited current state of knowledge, we can probably suggest that our lives will be simpler if we propose that a feature handle has meaning only within *one* scope, and that the handle itself contains the means to uniquely identify that scope.

How we identify scopes and how we define the function which evaluates a *feature handle* to produce (on demand) that feature or how we evaluate the handle to produce some kind of *scope handle*, is another matter.

3.3 Universal Identifiers: URIs, Monikers, GUIDs etc.

The problem of unique object identifiers has been tackled by several other software industries before geographical information users became interested or aware of the issue.

Existing distributed computing platforms all offer their own solutions to the problem, but the objects in these cases are responsive "live" software processes, not "dead" geographic features hidden inside proprietary software systems and

not accessible to dynamic enquiries. These “live” objects include Microsoft COM objects, CORBA nameservices, Inter-Language Unification (ILU) “String Binding Handles” (SBHs) and on-going work to develop Internet Service Location protocols [9,10]. Some of these object identifiers include type fingerprints and version information as well as providing uniqueness and persistence.

The object identifiers from the bibliographic and World Wide Web communities [8,11–13] are more what we require for geographic features though these too are evolving towards a “live” web-object way of operating.

3.4 Names or Addresses?

It is important to understand that names are not the same as addresses. They are conceptually distinct and some schemes implement them distinctly.

- An identifier is a name with particular persistence and uniqueness properties.
- A *naming scheme* is a system for creating these names.
- A name is *resolved* to an address by a *resolution service*.
- An *object server* uses the address to retrieve the named object.
- If you have a name, you need a *registry service* to tell you what resolution services are appropriate for it.

A practical advantage of the separation of names and addresses is that an address can be ephemeral even if the name is permanent. By separating resolution as a separate service, a name can outlast its originating organisation [14].

3.5 Mechanisms

When discussing any kind of object identifier problem, many people want to get straight in to discussing whether their favourite implementation mechanism will do what is required. There are basically two mechanisms for creating unique identifiers and the second comes in two main types:

1. Pseudo random generation
2. Federated hierarchical organisations
 - (a) where the same grammar is used by the subsidiary authorities
 - (b) where each subsidiary authority defines its own subsidiary addressing scheme

Pseudo-random generation is the surprisingly effective technique of generating a large random number from some local source of entropy, e.g. timing of keystrokes on a keyboard or a short analysis of network traffic coupled with an absolute clock value. The probability of two separately generated identifiers being the same can be reduced to arbitrarily low levels by ensuring that the number is large enough and the entropy unbiased enough [15]. Microsoft’s COM GUIDs use this method.

Federated naming schemes use a unique central authority which administers some top-level prefixes which it distributes to a number of other authorities, each of which adds something sequentially to the prefix and distributes further.

Conceptually, the identifier in a federated system gets longer and longer as the depth of the subsidiary tree gets deeper, but in practice it is quite possible to work within a maximum defined length so long as this is increased universally every few decades. The Internet Protocol (IP), Domain Name Service (DNS) and Ethernet card numbering systems work like this. Some naming protocols put an explicit depth on the tree.

The subsidiary naming systems may be subject to separate international standards, e.g. the *service type names* of the Service Location Protocol (SLP) are registered with the Internet Assigned Numbers Authority (IANA) [10].

The World-Wide Web architecture assumes that resource identifiers (URIs) are identifiable by their *scheme name* which determines the subsidiary naming scheme. This applies to all URNs and URIs (including URLs) [13]. A URN in absolute form consists of:

`<scheme> : <scheme-specific-encoding>`

where the *scheme name* usually contains usually only lowercase letters and digits. The scheme name identifies the naming service and, implicitly, the resolution service, which would be used to resolve the identifier [12, 13]. A subset of schemes use a common generic syntax:

`<scheme> ://<authority><path>?<query>`

The familiar “`http:`” URL uses a DNS *machine-name* (and optional port number) as the authority. Note that the *query* option means even the `http:` scheme can be extended to arbitrary encodings using cgi scripts and “?” parameter separators. The *Handle System* architecture can be defined with a scheme name “`hdl:`”. Thus a particular document has the persistent name `hdl:cnri.dlib/february96-urn_implementors` which (currently) resolves to the address: `http://www.dlib.org/dlib/february96/02arms.html` [14].

Given the existence of one universal naming system for organisations (DNS), there is no great reason to invent and maintain another. In the past, individual industries have had to organise their own hierarchical organisation naming systems, e.g. the International Article Numbering Association (EAN) which assigns manufacturer identification numbers for barcodes for retail goods and much else besides (`http://www.ean.be`).

Today the naming systems for extensions to multimedia email formats, Java packages, URIs and no doubt much else are “piggy-backed” onto DNS. Thus if an individual has a personal website `http://www.sargents.demon.co.uk` he can be sure that he can create a unique name for his collection of Java utilities by calling the package `uk.co.demon.sargents.utils`.

It has been suggested that feature identifiers should use their geographic location and feature type as part of their unique identity. Unfortunately this runs into so many problems with different resolutions, projections, datums, accuracies and Feature Class Codes that this approach is not now being seriously followed.

4 What do we need Identifiers For?

Having reviewed a bewildering array of identification mechanisms, nearly all under active development, we can see that we really do need some clearer idea of what we want identifiers for and how we want to use them:

1. When doing updates, we need to be able to determine if the supplied identifier in the update refers to a feature in the original dataset to decide whether to update it or to create a new one.
2. When comparing versions, we want to be able to find the previous version of a feature from the current version and vice versa.
3. When doing some work outside a GIS, we want to be able to make a reference to specific features in a published dataset which persists for some indefinite time into the future.
4. Within some defined scope (usually the “same” dataset), we want to assert that a relationship of a certain type exists between two features and we want this relationship to persist when any feature collection containing the related features is copied to another dataset.
5. When we copy a feature collection in its entirety, it would be useful if there were some simple relationship between the feature handles in the two copies, e.g. differing only in some prefix, so that access in both directions were quick and easy.

We introduced feature descriptors and feature handles earlier; but how can we use them ?

- A feature *descriptor* is useful only for finding and then acquiring a specific feature. It has no other purpose and since feature descriptors come from “outside” the system expressed in a variety of types of language there can be no guarantee that two different descriptors will not produce (resolve to, evaluate to, return) the same feature. In which case the two (different) descriptors have “equality by reference”.
- A feature *handle* will also produce a feature, but some *scopes* will *also* define that only identical feature handles can return the same feature. This means that “equality by value” implies “equality by reference”.
- There is also the intermediate case where a feature descriptor may be defined by a global scope which provides a “re-phrasing” service such that two different descriptors of specified types could be cast to a canonical form in which “equality by value” does then imply “equality by reference”.

What do we mean when we say a feature *descriptor* “returns a feature”? There are two possibilities:

1. we get a lump of binary data encoded in a “well-known format” which contains all the feature’s attribute data and sufficient references to a schema to be able to decode the names and types of the attributes,

2. we get a feature *handle* which incorporates the identification of the scope in which it is valid.

Whereas when we ask what happens when a feature *handle* “returns a feature”, we must mean that we get the binary lump.

Assuming that we used the descriptor to make a query on some third-party indexing service, we get a handle but we must then find the actual dataset repository. We require some naming scheme so that we can use the handle to then obtain information about the scope object itself (the dataset).

Each scope may have its own coding function which it uses to evaluate the handle and to return the feature itself (in a well-known binary format).

Some scopes may publish their coding functions and allow independent access into the dataset, e.g. a directory tree of files in well-known formats, others may maintain their own integrity and require access using opaque handles using a private coding function.

4.1 Mechanisms to Declare Scopes

The URI and RDF mechanisms described earlier (Sect. 3.5) are specifically designed to be applied to old software systems inherited from a previous age (“legacy” systems, though strictly speaking they should be called “heritage” systems). Thus so long as an existing geographic data repository is not still being updated, it can be *annotated* by setting up a URI which contains a subsidiary naming scheme and RDF descriptions of the metadata and schemas, and a subsidiary URI offering a unique identifier system for the individual features.

5 Putting it all Together

We have clearly seen that we need some type of “repository” which manages multiple feature collections (datasets), which maintains scopes, which can respond to queries about schemas, which can evaluate feature handles to return features and which can be identified and located from part of a feature handle’s observable value. We need such a repository for each “project” on which several people are working; incorporating several “lines of effort” at the same time. We have also seen that we need something (else?) which can evaluate feature descriptors and, if valid, return a feature handle. This latter service could make use of RDF and other metadata services and protocols.

It is suggested that the repository manager be identified with a URN and that feature handles also be legal URNs but where the initial part of the scheme-specific encoding is the URN of its manager.

It is not necessary that the repository manager actually be “on line” at any time: the important characteristics are solely the *persistence* and *uniqueness* of the identifiers. If desired, these could be maintained by an entirely manual process consisting of paper forms and authorised signatures as currently used in

many file-and-directory-based GIS archives. (The probability that an organisation will want to participate and yet refuses to register with DNS is assumed to be negligible.)

There are places for randomly generated identifiers, e.g. when generating new feature handles in disconnected remote sites, or generating short locally unique strings for storage as feature attributes [15]. However, since we need some kind of federated system for relating repositories which are going to exchange data anyway, it makes sense to use that for the primary architecture. We should also remember that not every feature necessarily needs to be issued an identifier, especially in inherited systems, and that identifiers do not necessarily need to be held “in” the dataset as attributes on the features.

The types of relationships, e.g. version relationships, between the different feature collections making up a collectively-managed “scope” could usefully be partially standardised [5] to encourage a software component market. The types of relationships which already exist in ad hoc, manually managed systems are varieties of “source” data related to “working” datasets and eventually “published” data. Those GISs which provide version services have better defined semantics for the narrower domain of version control.

6 Conclusions

Doing a decent job of a “simple” matter of proposing standard ways of constructing feature identifiers thus turns out to involve interrelated aspects of dynamic schema discovery, metadata granularity, formal version control semantics, distributed/replicated unique naming systems and a lot more besides. Despite the temptation to throw up our hands in horror, there does seem hope that very tightly-defined and narrowly focused *feature handles* may yet provide some usable functionality which is worth the effort of implementation.

Acknowledgements

The paper is a personal viewpoint written while the author was a Visiting Scientist at the European Commission Joint Research Centre at Ispra, Italy. Notable contributions were made by Adam Gawne-Cain (cadcorp), Grant Ruwoldt (Bentley Systems, Inc.), Adrian Cuthbert (Laser-Scan Ltd.) and David Arctur (Laser-Scan, Inc.) in the Features Special Interest Group of the Open GIS Consortium Technical Committee’s Core Task Force. Thanks are also due to the OGC staff who facilitated our work and showed that they understood what we were doing and why.

No part of this paper should be construed as any intention by OGC to produce proposals for standards in this area.

References

1. Burrough, P.A., Frank, A.U., (eds.): Geographic Objects with Indeterminate Boundaries. GISDATA2. Taylor and Francis, London (1996) ISBN 0-7484-0386-8 (series editors Ian Masser and François Salgé)
2. Open GIS Consortium: The OpenGIS[®] Abstract Specification Topic 5, The OpenGIS[®] Feature, <http://www.opengis.org/public/abstract.html>.
3. Lutz, M.: Programming Python. O'Reilly & Associates, Inc., Bonn Cambridge Paris Sebastopol Tokyo (1996). ISBN 1-56592-197-6. <http://www.python.org>.
4. The Open GIS Consortium: The OpenGIS[®] Implementation Specification, OpenGIS[®] Simple Features Specifications for OLE/COM, CORBA and SQL, <http://www.opengis.org/techno/specs.htm>.
5. Kaler, C.: Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning), Internet Engineering Task Force (IETF), Work in progress — Internet Draft August 6, 1998, Document `draft-kaler-webdav-versioning-00.txt` in <http://www.ietf.org/internet-drafts/>, see also <http://www.ics.uci.edu/~ejw/authoring/>
6. ISOTC 211WG 3, Geospatial data administration, Part 15: Metadata, <http://www.statkart.no/isotc211/wg3/wg3welc.htm>.
7. Committee on Earth Observation Satellites: CIP - Catalogue Interoperability Protocol, <http://lcweb.loc.gov/z3950/agency/profiles/cip.html>.
8. Manola, F.: Towards a Web Object Model, <http://www.objs.com/OSA/wom.htm>.
9. Inter-Language Unification (ILU) Concepts, ftp://ftp.parc.xerox.com/pub/ilu/2.0a12/manual-html/manual_1.html.
10. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service Location Protocol, Internet Engineering Task Force (IETF), Work in progress — Internet Draft, July 30, 1998 Document `draft-ietf-svrloc-protocol-v2-08.txt` in <http://www.ietf.org/internet-drafts/>, see also <http://www.ietf.org/ids.by.wg/svrloc.html>.
11. Bray, T., Hollander, D., Layman, A.: Namespaces in XML, World Wide Web Consortium (W3C) Working Draft 18-May-1998 <http://www.w3.org/TR/WD-xml-names>.
12. Berners-Lee, T, Fielding, R., Irvine, U.C., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax, Internet Engineering Task Force (IETF) Request For Comment (RFC) 2396, August 1998, <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2396.txt>.
13. World Wide Web Consortium: Naming and Addressing: URIs, <http://www.w3.org/Addressing/Addressing.html>.
14. Sun, S.X.: Handle System: A Persistent Global Name Service — Overview and Syntax, Internet Engineering Task Force (IETF), Work in progress — Internet Draft July 16, 1998, Document `draft-sun-handle-system-01.txt` in <http://www.ietf.org/internet-drafts/>, see also <http://www.handle.net>.
15. Leach, P.J., Salz, R.: UUIDs and GUIDs, Internet Engineering Task Force (IETF), Work in progress — Internet Draft February 4, 1998, Document `draft-leach-uuids-guids-01.txt` in <http://www.ietf.org/internet-drafts/>, see also <http://www.ics.uci.edu/~ejw/authoring/>