W3C Recommendation

Architecture of the World Wide Web, Volume One

W3C Recommendation 15 December 2004

This version:
http://www.w3.org/TR/2004/REC-webarch-20041215/

Latest version:
http://www.w3.org/TR/webarch/

Previous version:
http://www.w3.org/TR/2004/PR-webarch-20041105/

Editors:
Ian Jacobs, W3C
Norman Walsh, Sun Microsystems, Inc.

Authors:
See acknowledgments (§8) (see page 58).

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

The World Wide Web uses relatively simple technologies with sufficient scalability, efficiency and utility that they have resulted in a remarkable information space of interrelated resources, growing across languages, cultures, and media. In an effort to preserve these properties of the information space as the technologies evolve, this architecture document discusses the core design components of the Web. They are identification of resources, representation of resource state, and the protocols that support the interaction between agents and resources in the space. We relate core design components, constraints, and good practices to the principles and properties they support.

W3C Recommendation

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is the 15 December 2004 Recommendation of “Architecture of the World Wide Web, Volume One.” This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was developed by W3C's Technical Architecture Group (TAG), which, by charter maintains a list of architectural issues. The scope of this document is a useful subset of those issues; it is not intended to address all of them. The TAG intends to address the remaining (and future) issues now that Volume One is published as a W3C Recommendation. A complete history of changes so this document is available. Please send comments on this document to public-webarch-comments@w3.org (public archive of public-webarch-comments). TAG technical discussion takes place on www-tag@w3.org (public archive of www-tag).

This document was produced under the W3C IPR policy of the July 2001 Process Document. The TAG maintains a public list of patent disclosures relevant to this document; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

List of Principles, Constraints, and Good Practice Notes

The following principles, constraints, and good practice notes are discussed in this document and listed here for convenience. There is also a free-standing summary.

1. Introduction

The ***World Wide Web*** (***WWW***, or simply ***Web***) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (***URI***).

Examples such as the following travel scenario are used throughout this document to illustrate typical behavior of ***Web agents***—people or software acting on this information space. A ***user agent*** acts on behalf of a user. Software agents include servers, proxies, spiders, browsers, and multimedia players.

Story

While planning a trip to Mexico, Nadia reads “Oaxaca weather information: 'http://weather.example.com/oaxaca'” in a glossy travel magazine. Nadia has enough experience with the Web to recognize that "http://weather.example.com/oaxaca" is a URI and that she is likely to be able to retrieve associated information with her Web browser. When Nadia enters the URI into her browser:

1. The browser recognizes that what Nadia typed is a URI.
2. The browser performs an information retrieval action in accordance with its configured behavior for resources identified via the "http" URI scheme.
3. The authority responsible for "weather.example.com" provides information in a response to the retrieval request.
4. The browser interprets the response, identified as XHTML by the server, and performs additional retrieval actions for inline graphics and other content as necessary.
5. The browser displays the retrieved information, which includes hypertext links to other information. Nadia can follow these hypertext links to retrieve additional information.

This scenario illustrates the three architectural bases of the Web that are discussed in this document:

1. Identification (§2) (see page 10). URIs are used to identify resources. In this travel scenario, the resource is a periodically updated report on the weather in Oaxaca, and the URI is “http://weather.example.com/oaxaca”.

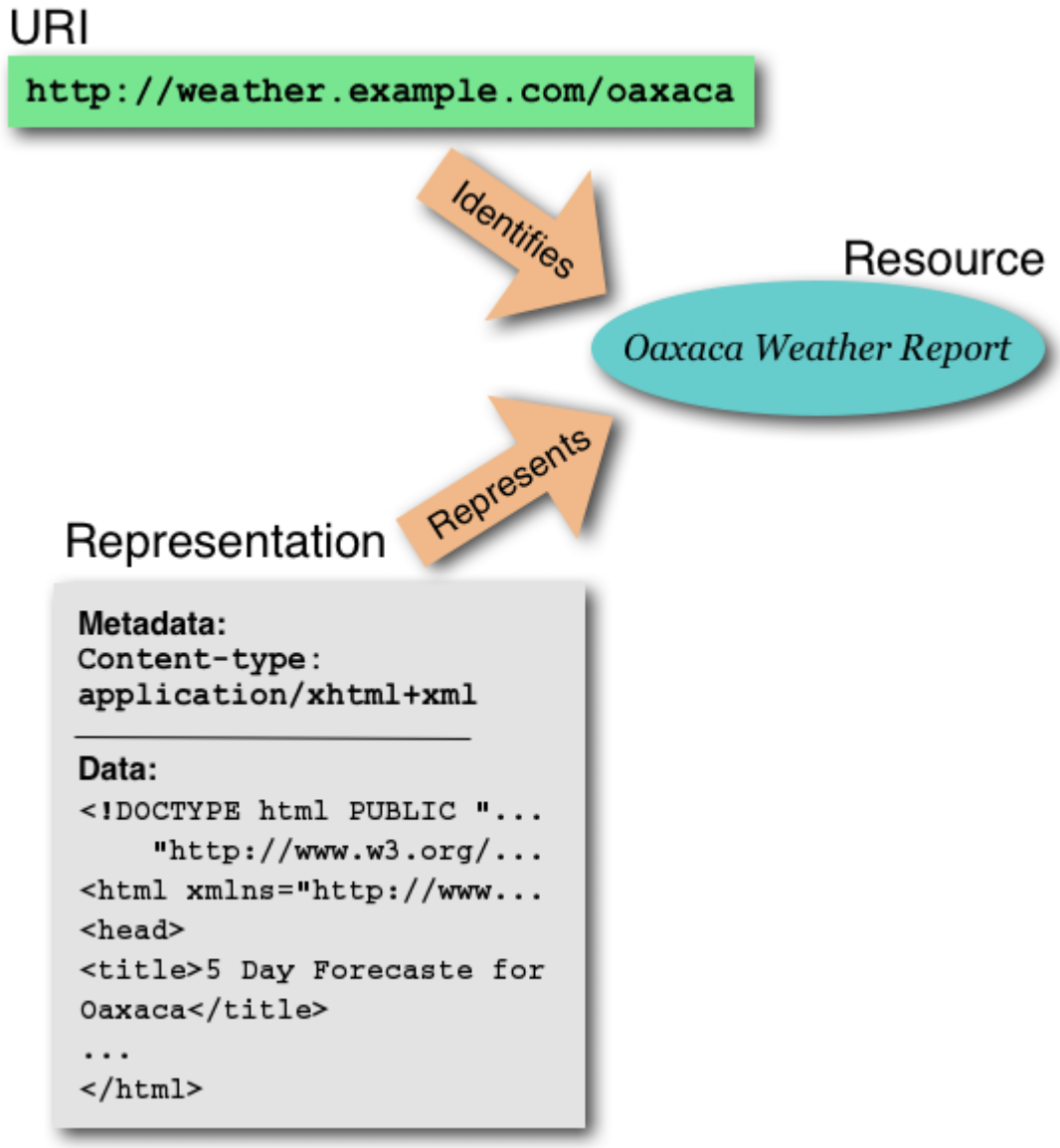
2. Interaction (§3) (see page 20). Web agents communicate using standardized protocols that enable interaction through the exchange of messages which adhere to a defined syntax and semantics. By entering a URI into a retrieval dialog or selecting a hypertext link, Nadia tells her browser to perform a retrieval action for the resource identified by the URI. In this example, the browser sends an HTTP GET request (part of the HTTP protocol) to the server at "weather.example.com", via TCP/IP port 80, and the server sends back a message containing what it determines to be a representation of the resource as of the time that representation was generated. Note that this example is specific to hypertext browsing of information—other kinds of interaction are possible, both within browsers and through the use of other types of Web agent; our example is intended to illustrate one common interaction, not define

the range of possible interactions or limit the ways in which agents might use the Web.

3. Formats (§4) (see page 32). Most protocols used for representation retrieval and/or submission make use of a sequence of one or more messages, which taken together contain a payload of representation data and metadata, to transfer the representation between agents. The choice of interaction protocol places limits on the formats of representation data and metadata that can be transmitted. HTTP, for example, typically transmits a single octet stream plus metadata, and uses the "Content-Type" and "Content-Encoding" header fields to further identify the format of the representation. In this scenario, the representation transferred is in XHTML, as identified by the "Content-type" HTTP header field containing the registered Internet media type name, "application/xhtml+xml". That Internet media type name indicates that the representation data can be processed according to the XHTML specification.

   Nadia's browser is configured and programmed to interpret the receipt of an "application/xhtml+xml" typed representation as an instruction to render the content of that representation according to the XHTML rendering model, including any subsidiary interactions (such as requests for external style sheets or in-line images) called for by the representation. In the scenario, the XHTML representation data received from the initial request instructs Nadia's browser to also retrieve and render in-line the weather maps, each identified by a URI and thus causing an additional retrieval action, resulting in additional representations that are processed by the browser according to their own data formats (e.g., "application/svg+xml" indicates the SVG data format), and this process continues until all of the data formats have been rendered. The result of all of this processing, once the browser has reached an application steady-state that completes Nadia's initial requested action, is commonly referred to as a "Web page".

The following illustration shows the relationship between identifier, resource, and representation.



In the remainder of this document, we highlight important architectural points regarding Web identifiers, protocols, and formats. We also discuss some important general architectural principles (§5) and how they apply to the Web.

1.1. About this Document

This document describes the properties we desire of the Web and the design choices that have been made to achieve them. It promotes the reuse of existing standards when suitable, and gives guidance on how to innovate in a manner consistent with Web architecture.

The terms MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY are used in the principles, constraints, and good practice notes in accordance with RFC 2119 [RFC2119 (see page 53)].

This document does not include conformance provisions for these reasons:

- Conforming software is expected to be so diverse that it would not be useful to be able to refer to the class of conforming software agents.
- Some of the good practice notes concern people; specifications generally define conformance for software, not people.
- We do not believe that the addition of a conformance section is likely to increase the utility of the document.

1.1.1. Audience of this Document

This document is intended to inform discussions about issues of Web architecture. The intended audience for this document includes:

1. Participants in W3C Activities
2. Other groups and individuals designing technologies to be integrated into the Web
3. Implementers of W3C specifications
4. Web content authors and publishers

Note: This document does not distinguish in any formal way the terms "language" and "format." Context determines which term is used. The phrase "specification designer" encompasses language, format, and protocol designers.

1.1.2. Scope of this Document

This document presents the general architecture of the Web. Other groups inside and outside W3C also address specialized aspects of Web architecture, including accessibility, quality assurance, internationalization, device independence, and Web Services. The section on Architectural Specifications (§7.1) (see page 56) includes references to these related specifications.

This document strives for a balance between brevity and precision while including illustrative examples. TAG findings are informational documents that complement the current document by providing more detail about selected topics. This document includes some excerpts from the findings. Since the findings evolve independently, this document includes references to approved TAG findings. For other TAG issues covered by this document but without an approved finding, references are to entries in the TAG issues list.

Many of the examples in this document that involve human activity suppose the familiar Web interaction model (illustrated at the beginning of the Introduction) where a person follows a link via a user agent, the user agent retrieves and presents data, the user follows another link, etc. This document does not discuss in any detail other interaction models such as voice browsing (see, for example, [VOICEXML2 (see page 55)]). The choice of interaction model may have an impact on expected agent behavior. For instance, when a graphical user agent running on a laptop computer or hand-held device encounters an error, the user agent can report errors directly to the user through visual and audio cues, and present the user with options for resolving the errors. On the other hand, when someone is browsing the Web through voice input and audio-only output, stopping the dialog to wait for user input may reduce usability since it is so easy to "lose one's place" when browsing with only audio-output. This document does not discuss how the

principles, constraints, and good practices identified here apply in all interaction contexts.

1.1.3. Principles, Constraints, and Good Practice Notes

The important points of this document are categorized as follows:

Principle
: An architectural principle is a fundamental rule that applies to a large number of situations and variables. Architectural principles include "separation of concerns", "generic interface", "self-descriptive syntax," "visible semantics," "network effect" (Metcalfe's Law), and Amdahl's Law: "The speed of a system is limited by its slowest component."

Constraint
: In the design of the Web, some choices, like the names of the `p` and `li` elements in HTML, the choice of the colon (:) character in URIs, or grouping bits into eight-bit units (octets), are somewhat arbitrary; if `paragraph` had been chosen instead of `p` or asterisk (*) instead of colon, the large-scale result would, most likely, have been the same. This document focuses on more fundamental design choices: design choices that lead to constraints, i.e., restrictions in behavior or interaction within the system. Constraints may be imposed for technical, policy, or other reasons to achieve desirable properties in the system, such as accessibility, global scope, relative ease of evolution, efficiency, and dynamic extensibility.

Good practice
: Good practice—by software developers, content authors, site managers, users, and specification designers—increases the value of the Web.

2. Identification

In order to communicate internally, a community agrees (to a reasonable extent) on a set of terms and their meanings. One goal of the Web, since its inception, has been to build a global community in which any party can share information with any other party. To achieve this goal, the Web makes use of a single global identification system: the URI. URIs are a cornerstone of Web architecture, providing identification that is common across the Web. The global scope of URIs promotes large-scale "network effects": the value of an identifier increases the more it is used consistently (for example, the more it is used in hypertext links (§4.4)).

> ***Principle: Global Identifiers***
>
> Global naming leads to global network effects.

This principle dates back at least as far as Douglas Engelbart's seminal work on open hypertext systems; see section Every Object Addressable in [Eng90 (see page 52)].

2.1. Benefits of URIs

The choice of syntax for global identifiers is somewhat arbitrary; it is their global scope that is important. The ***Uniform Resource Identifier***, [URI (see page 55)], has been successfully deployed since the creation of the Web. There are substantial benefits to participating in the existing network of URIs, including linking, bookmarking, caching, and indexing by search engines, and there are substantial costs to creating a new identification system that has the same properties as URIs.

> ***Good practice: Identify with URIs***
>
> To benefit from and increase the value of the World Wide Web, agents should provide URIs as identifiers for resources.

A resource should have an associated URI if another party might reasonably want to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it. Software developers should expect that sharing URIs across applications will be useful, even if that utility is not initially evident. The TAG finding *"URIs, Addressability, and the use of HTTP GET and POST"* discusses additional benefits and considerations of URI addressability.

Note: Some URI schemes (such as the "ftp" URI scheme specification) use the term "designate" where this document uses "identify."

2.2. URI/Resource Relationships

By design a URI identifies one resource. We do not limit the scope of what might be a ***resource***. The term "resource" is used in a general sense for whatever might be identified by a URI. It is conventional on the hypertext Web to describe Web pages, images, product catalogs, etc. as “resources”. The distinguishing characteristic of

these resources is that all of their essential characteristics can be conveyed in a message. We identify this set as "***information resources***."

This document is an example of an information resource. It consists of words and punctuation symbols and graphics and other artifacts that can be encoded, with varying degrees of fidelity, into a sequence of bits. There is nothing about the essential information content of this document that cannot in principle be transfered in a message. In the case of this document, the message payload is the representation (see page 22) of this document.

However, our use of the term resource is intentionally more broad. Other things, such as cars and dogs (and, if you've printed this document on physical sheets of paper, the artifact that you are holding in your hand), are resources too. They are not information resources, however, because their essence is not information. Although it is possible to describe a great many things about a car or a dog in a sequence of bits, the sum of those things will invariably be an approximation of the essential character of the resource.

We define the term "information resource" because we observe that it is useful in discussions of Web technology and may be useful in constructing specifications for facilities built for use on the Web.

Constraint: URIs Identify a Single Resource

Assign distinct URIs to distinct resources.

Since the scope of a URI is global, the resource identified by a URI does not depend on the context in which the URI appears (see also the section about indirect identification (§2.2.3)).

[URI (see page 55)] is an agreement about how the Internet community allocates names and associates them with the resources they identify. URIs are divided into schemes (§2.4) (see page 16) that define, via their scheme specification, the mechanism by which scheme-specific identifiers are associated with resources. For example, the "http" URI scheme ([RFC2616 (see page 54)]) uses DNS and TCP-based HTTP servers for the purpose of identifier allocation and resolution. As a result, identifiers such as "http://example.com/somepath#someFrag" often take on meaning through the community experience of performing an HTTP GET request on the identifier and, if given a successful response, interpreting the response as a representation of the identified resource. (See also Fragment Identifiers (§2.6) (see page 18).) Of course, a retrieval action like GET is not the only way to obtain information about a resource. One might also publish a document that purports to define the meaning of a particular URI. These other sources of information may suggest meanings for such identifiers, but it's a local policy decision whether those suggestions should be heeded.

Just as one might wish to refer to a person by different names (by full name, first name only, sports nickname, romantic nickname, and so forth), Web architecture allows the association of more than one URI with a resource. URIs that identify the same resource are called ***URI aliases***. The section on URI aliases (§2.3.1) (see page 14) discusses some of the potential costs of creating multiple URIs for the same resource.

Several sections of this document address questions about the relationship between URIs and resources, including:

- How much can I tell about a resource by inspection of a URI that identifies it? See the sections on URI schemes (§2.4) (see page 16) and URI opacity (§2.5) (see page 17).
- Who determines what resource a URI identifies? See the section on URI allocation (§2.2.2) (see page 12).
- Can the resource identified by a URI change over time? See the sections on URI persistence (§3.5.1) (see page 29) and representation management (§3.5) (see page 28).
- Since more than one URI can identify the same resource, how do I know which URIs identify the same resource? See the sections on URI comparison (§2.3) (see page 14) and assertions that two URIs identify the same resource (§2.7.2) (see page 19).

2.2.1. URI collision

By design, a URI identifies one resource. Using the same URI to directly identify different resources produces a ***URI collision***. Collision often imposes a cost in communication due to the effort required to resolve ambiguities.

Suppose, for example, that one organization makes use of a URI to refer to the movie *The Sting*, and another organization uses the same URI to refer to a discussion forum about *The Sting*. To a third party, aware of both organizations, this collision creates confusion about what the URI identifies, undermining the value of the URI. If one wanted to talk about the creation date of the resource identified by the URI, for instance, it would not be clear whether this meant "when the movie was created" or "when the discussion forum about the movie was created."

Social and technical solutions have been devised to help avoid URI collision. However, the success or failure of these different approaches depends on the extent to which there is consensus in the Internet community on abiding by the defining specifications.

The section on URI allocation (§2.2.2) examines approaches for establishing the authoritative source of information about what resource a URI identifies.

URIs are sometimes used for indirect identification (§2.2.3). This does not necessarily lead to collisions.

2.2.2. URI allocation

URI allocation is the process of associating a URI with a resource. Allocation can be performed both by resource owners and by other parties. It is important to avoid URI collision (§2.2.1) (see page 12).

2.2.2.1. URI ownership

URI ownership is a relation between a URI and a social entity, such as a person, organization, or specification. URI ownership gives the relevant social entity certain rights, including:

1. to pass on ownership of some or all owned URIs to another owner—delegation; and
2. to associate a resource with an owned URI—URI allocation.

By social convention, URI ownership is delegated from the IANA URI scheme registry [IANASchemes], itself a social entity, to IANA-registered URI scheme specifications. Some URI scheme specifications further delegate ownership to subordinate registries or to other nominated owners, who may further delegate ownership. In the case of a specification, ownership ultimately lies with the community that maintains the specification.

The approach taken for the "http" URI scheme, for example, follows the pattern whereby the Internet community delegates authority, via the IANA URI scheme registry and the DNS, over a set of URIs with a common prefix to one particular owner. One consequence of this approach is the Web's heavy reliance on the central DNS registry. A different approach is taken by the URN Syntax scheme [RFC2141 (see page 53)] which delegates ownership of portions of URN space to URN Namespace specifications which themselves are registered in an IANA-maintained registry of URN Namespace Identifiers.

URI owners are responsible for avoiding the assignment of equivalent URIs to multiple resources. Thus, if a URI scheme specification does provide for the delegation of individual or organized sets of URIs, it should take pains to ensure that ownership ultimately resides in the hands of a single social entity. Allowing multiple owners increases the likelihood of URI collisions.

URI owners may organize or deploy infrastruture to ensure that representations of associated resources are available and, where appropriate, interaction with the resource is possible through the exchange of representations. There are social expectations for responsible representation management (§3.5) by URI owners. Additional social implications of URI ownership are not discussed here.

See TAG issue siteData-36, which concerns the expropriation of naming authority.

2.2.2.2. Other allocation schemes

Some schemes use techniques other than delegated ownership to avoid collision. For example, the specification for the data URL (sic) scheme [RFC2397 (see page 53)] specifies that the resource identified by a data scheme URI has only one possible representation. The representation data makes up the URI that identifies that resource. Thus, the specification itself determines how data URIs are allocated; no delegation is possible.

Other schemes (such as "news:comp.text.xml") rely on a social process.

2.2.3. Indirect Identification

To say that the URI "mailto:nadia@example.com" identifies both an Internet mailbox and Nadia, the person, introduces a URI collision. However, we can use the URI to indirectly identify Nadia. Identifiers are commonly used in this way.

Listening to a news broadcast, one might hear a report on Britain that begins, "Today, 10 Downing Street announced a series of new economic measures." Generally, "10 Downing Street" identifies the official residence of Britain's Prime Minister. In this context, the news reporter is using it (as English rhetoric allows) to indirectly identify the British government. Similarly, URIs identify resources, but they can also be used in many constructs to indirectly identify other resources. Globally adopted assignment policies make some URIs appealing as general-purpose identifiers. Local policy establishes what they indirectly identify.

Suppose that `nadia@example.com` is Nadia's email address. The organizers of a conference Nadia attends might use "mailto:nadia@example.com" to refer indirectly to her (e.g., by using the URI as a database key in their database of conference participants). This does not introduce a URI collision.

2.3. URI Comparisons

URIs that are identical, character-by-character, refer to the same resource. Since Web Architecture allows the association of multiple URIs with a given resource, two URIs that are not character-by-character identical may still refer to the same resource. Different URIs do not necessarily refer to different resources but there is generally a higher computational cost to determine that different URIs refer to the same resource.

To reduce the risk of a false negative (i.e., an incorrect conclusion that two URIs do not refer to the same resource) or a false positive (i.e., an incorrect conclusion that two URIs do refer to the same resource), some specifications describe equivalence tests in addition to character-by-character comparison. Agents that reach conclusions based on comparisons that are not licensed by the relevant specifications take responsibility for any problems that result; see the section on error handling (§5.3) for more information about responsible behavior when reaching unlicensed conclusions. Section 6 of [URI (see page 55)] provides more information about comparing URIs and reducing the risk of false negatives and positives.

See also the assertion that two URIs identify the same resource (§2.7.2).

2.3.1. URI aliases

Although there are benefits (such as naming flexibility) to URI aliases, there are also costs. URI aliases are harmful when they divide the Web of related resources. A corollary of Metcalfe's Principle (the "network effect") is that the value of a given resource can be measured by the number and value of other resources in its network neighborhood, that is, the resources that link to it.

The problem with aliases is that if half of the neighborhood points to one URI for a given resource, and the other half points to a second, different URI for that same resource, the neighborhood is divided. Not only is the aliased resource undervalued because of this split, the entire neighborhood of resources loses value because of the missing second-order relationships that should have existed among the referring resources by virtue of their references to the aliased resource.

Good practice: Avoiding URI aliases

A URI owner SHOULD NOT associate arbitrarily different URIs with the same resource.

URI consumers also have a role in ensuring URI consistency. For instance, when transcribing a URI, agents should not gratuitously percent-encode characters. The term "character" refers to URI characters as defined in section 2 of [URI (see page 55)]; percent-encoding is discussed in section 2.1 of that specification.

Good practice: Consistent URI usage

An agent that receives a URI SHOULD refer to the associated resource using the same URI, character-by-character.

When a URI alias does become common currency, the URI owner (see page 12) should use protocol techniques such as server-side redirects to relate the two resources. The community benefits when the URI owner supports redirection of an aliased URI to the corresponding "official" URI. For more information on redirection, see section 10.3, Redirection, in [RFC2616 (see page 54)]. See also [CHIPS (see page 52)] for a discussion of some best practices for server administrators.

2.3.2. Representation reuse

URI aliasing only occurs when more than one URI is used to identify the same resource. The fact that different resources sometimes have the same representation does not make the URIs for those resources aliases.

Story

Dirk would like to add a link from his Web site to the Oaxaca weather site. He uses the URI http://weather.example.com/oaxaca and labels his link “report on weather in Oaxaca on 1 August 2004”. Nadia points out to Dirk that he is setting misleading expectations for the URI he has used. The Oaxaca weather site policy is that the URI in question identifies a report on the current weather in Oaxaca—on any given day—and not the weather on 1 August. Of course, on the first of August in 2004, Dirk's link will be correct, but the rest of the time he will be misleading readers. Nadia points out to Dirk that the managers of the Oaxaca weather site do make available a different URI permanently assigned to a resource reporting on the weather on 1 August 2004.

In this story, there are two resources: “a report on the current weather in Oaxaca” and “a report on the weather in Oaxaca on 1 August 2004”. The managers of the Oaxaca weather site assign two URIs to these two different resources. On 1 August 2004, the representations for these resources are identical. That fact that dereferencing two different URIs produces identical representations does not imply that the two URIs are aliases.

2.4. URI Schemes

In the URI "http://weather.example.com/", the "http" that appears before the colon (":") names a URI scheme. Each URI scheme has a specification that explains the scheme-specific details of how scheme identifiers are allocated and become associated with a resource. The URI syntax is thus a federated and extensible naming system wherein each scheme's specification may further restrict the syntax and semantics of identifiers within that scheme.

Examples of URIs from various schemes include:

- mailto:joe@example.org
- ftp://example.org/aDirectory/aFile
- news:comp.infosystems.www
- tel:+1-816-555-1212
- ldap://ldap.example.org/c=GB?objectClass?one
- urn:oasis:names:tc:entity:xmlns:xml:catalog

While Web architecture allows the definition of new schemes, introducing a new scheme is costly. Many aspects of URI processing are scheme-dependent, and a large amount of deployed software already processes URIs of well-known schemes. Introducing a new URI scheme requires the development and deployment not only of client software to handle the scheme, but also of ancillary agents such as gateways, proxies, and caches. See [RFC2718 (see page 54)] for other considerations and costs related to URI scheme design.

Because of these costs, if a URI scheme exists that meets the needs of an application, designers should use it rather than invent one.

Good practice: Reuse URI schemes

A specification SHOULD reuse an existing URI scheme (rather than create a new one) when it provides the desired properties of identifiers and their relation to resources.

Consider our travel scenario (see page 5): should the agent providing information about the weather in Oaxaca register a new URI scheme "weather" for the identification of resources related to the weather? They might then publish URIs such as "weather://travel.example.com/oaxaca". When a software agent dereferences such a URI, if what really happens is that HTTP GET is invoked to retrieve a representation of the resource, then an "http" URI would have sufficed.

2.4.1. URI Scheme Registration

The Internet Assigned Numbers Authority (IANA) maintains a registry [IANASchemes] of mappings between URI scheme names and scheme specifications. For instance, the IANA registry indicates that the "http" scheme is defined in [RFC2616]. The process for registering a new URI scheme is defined in [RFC2717 (see page 54)].

Unregistered URI schemes SHOULD NOT be used for a number of reasons:

- There is no generally accepted way to locate the scheme specification.
- Someone else may be using the scheme for other purposes.
- One should not expect that general-purpose software will do anything useful with URIs of this scheme beyond URI comparison.

One misguided motivation for registering a new URI scheme is to allow a software agent to launch a particular application when retrieving a representation. The same thing can be accomplished at lower expense by dispatching instead on the type of the representation, thereby allowing use of existing transfer protocols and implementations.

Even if an agent cannot process representation data in an unknown format, it can at least retrieve it. The data may contain enough information to allow a user or user agent to make some use of it. When an agent does not handle a new URI scheme, it cannot retrieve a representation.

When designing a new data format, the preferred mechanism to promote its deployment on the Web is the Internet media type (see Representation Types and Internet Media Types (§3.2) (see page 22)). Media types also provide a means for building new information applications, as described in future directions for data formats (§4.6) (see page 45).

2.5. URI Opacity

It is tempting to guess the nature of a resource by inspection of a URI that identifies it. However, the Web is designed so that agents communicate resource information state through representations (see page 22), not identifiers. In general, one cannot determine the type of a resource representation by inspecting a URI for that resource. For example, the ".html" at the end of "http://example.com/page.html" provides no guarantee that representations of the identified resource will be served with the Internet media type "text/html". The publisher is free to allocate identifiers and define how they are served. The HTTP protocol does not constrain the Internet media type based on the path component of the URI; the URI owner is free to configure the server to return a representation using PNG or any other data format.

Resource state may evolve over time. Requiring a URI owner to publish a new URI for each change in resource state would lead to a significant number of broken references. For robustness, Web architecture promotes independence between an identifier and the state of the identified resource.

Good practice: URI opacity

Agents making use of URIs SHOULD NOT attempt to infer properties of the referenced resource.

In practice, a small number of inferences can be made because they are explicitly licensed by the relevant specifications. Some of these inferences are discussed in the details of retrieving a representation (§3.1.1) (see page 21).

The example URI used in the travel scenario (see page 5) ("http://weather.example.com/oaxaca") suggests to a human reader that the identified resource has something to do with the weather in Oaxaca. A site reporting the

weather in Oaxaca could just as easily be identified by the URI "http://vjc.example.com/315". And the URI "http://weather.example.com/vancouver" might identify the resource "my photo album."

On the other hand, the URI "mailto:joe@example.com" indicates that the URI refers to a mailbox. The "mailto" URI scheme specification authorizes agents to infer that URIs of this form identify Internet mailboxes.

Some URI assignment authorities document and publish their URI assignment policies. For more information about URI opacity, see TAG issues metaDataInURI-31 and siteData-36.

2.6. Fragment Identifiers

Story

When browsing the XHTML document that Nadia receives as a representation of the resource identified by "http://weather.example.com/oaxaca", she finds that the URI "http://weather.example.com/oaxaca#weekend" refers to the part of the representation that conveys information about the weekend outlook. This URI includes the fragment identifier "weekend" (the string after the "#").

The ***fragment identifier*** component of a URI allows indirect identification of a ***secondary resource*** by reference to a primary resource and additional identifying information. The secondary resource may be some portion or subset of the primary resource, some view on representations of the primary resource, or some other resource defined or described by those representations. The terms "primary resource" and "secondary resource" are defined in section 3.5 of [URI (see page 55)].

The terms “primary” and “secondary” in this context do not limit the nature of the resource—they are not classes. In this context, primary and secondary simply indicate that there is a relationship between the resources for the purposes of one URI: the URI with a fragment identifier. Any resource can be identified as a secondary resource. It might also be identified using a URI without a fragment identifier, and a resource may be identified as a secondary resource via multiple URIs. The purpose of these terms is to enable discussion of the relationship between such resources, not to limit the nature of a resource.

The interpretation of fragment identifiers is discussed in the section on media types and fragment identifier semantics (§3.2.1).

See TAG issue abstractComponentRefs-37, which concerns the use of fragment identifiers with namespace names to identify abstract components.

2.7. Future Directions for Identifiers

There remain open questions regarding identifiers on the Web.

2.7.1. Internationalized identifiers

The integration of internationalized identifiers (i.e., composed of characters beyond those allowed by [URI (see page 55)]) into the Web architecture is an important and open issue. See TAG issue IRIEverywhere-27 for discussion about work going on in this area.

2.7.2. Assertion that two URIs identify the same resource

Emerging Semantic Web technologies, including the "Web Ontology Language (OWL)" [OWL10 (see page 53)], define RDF properties such as `sameAs` to assert that two URIs identify the same resource or `inverseFunctionalProperty` to imply it.

3. Interaction

Communication between agents over a network about resources involves URIs, messages, and data. The Web's protocols (including HTTP, FTP, SOAP, NNTP, and SMTP) are based on the exchange of messages. A ***message*** may include data as well as metadata about a resource (such as the "Alternates" and "Vary" HTTP headers), the message data, and the message itself (such as the "Transfer-encoding" HTTP header). A message may even include metadata about the message metadata (for message-integrity checks, for instance).

Story

Nadia follows a hypertext link labeled "satellite image" expecting to retrieve a satellite photo of the Oaxaca region. The link to the satellite image is an XHTML link encoded as `<a href="http://example.com/satimage/oaxaca">satellite image</a>`. Nadia's browser analyzes the URI and determines that its scheme (see page 16) is "http". The browser configuration determines how it locates the identified information, which might be via a cache of prior retrieval actions, by contacting an intermediary (such as a proxy server), or by direct access to the server identified by a portion of the URI. In this example, the browser opens a network connection to port 80 on the server at "example.com" and sends a "GET" message as specified by the HTTP protocol, requesting a representation of the resource.

The server sends a response message to the browser, once again according to the HTTP protocol. The message consists of several headers and a JPEG image. The browser reads the headers, learns from the "Content-Type" field that the Internet media type of the representation is "image/jpeg", reads the sequence of octets that make up the representation data, and renders the image.

This section describes the architectural principles and constraints regarding interactions between agents, including such topics as network protocols and interaction styles, along with interactions between the Web as a system and the people that make use of it. The fact that the Web is a highly distributed system affects architectural constraints and assumptions about interactions.

3.1. Using a URI to Access a Resource

Agents may use a URI to access the referenced resource; this is called ***dereferencing the URI***. Access may take many forms, including retrieving a representation of the resource (for instance, by using HTTP GET or HEAD), adding or modifying a representation of the resource (for instance, by using HTTP POST or PUT, which in some cases may change the actual state of the resource if the submitted representations are interpreted as instructions to that end), and deleting some or all representations of the resource (for instance, by using HTTP DELETE, which in some cases may result in the deletion of the resource itself).

There may be more than one way to access a resource for a given URI; application context determines which access method an agent uses. For instance, a browser

might use HTTP GET to retrieve a representation of a resource, whereas a hypertext link checker might use HTTP HEAD on the same URI simply to establish whether a representation is available. Some URI schemes set expectations about available access methods, others (such as the URN scheme [RFC 2141 (see page 53)]) do not. Section 1.2.2 of [URI (see page 55)] discusses the separation of identification and interaction in more detail. For more information about relationships between multiple access methods and URI addressability, see the TAG finding *"URIs, Addressability, and the use of HTTP GET and POST"*.

Although many URI schemes (§2.4) (see page 16) are named after protocols, this does not imply that use of such a URI will necessarily result in access to the resource via the named protocol. Even when an agent uses a URI to retrieve a representation, that access might be through gateways, proxies, caches, and name resolution services that are independent of the protocol associated with the scheme name.

Many URI schemes define a default interaction protocol for attempting access to the identified resource. That interaction protocol is often the basis for allocating identifiers within that scheme, just as "http" URIs are defined in terms of TCP-based HTTP servers. However, this does not imply that all interaction with such resources is limited to the default interaction protocol. For example, information retrieval systems often make use of proxies to interact with a multitude of URI schemes, such as HTTP proxies being used to access "ftp" and "wais" resources. Proxies can also to provide enhanced services, such as annotation proxies that combine normal information retrieval with additional metadata retrieval to provide a seamless, multidimensional view of resources using the same protocols and user agents as the non-annotated Web. Likewise, future protocols may be defined that encompass our current systems, using entirely different interaction mechanisms, without changing the existing identifier schemes. See also, principle of orthogonal specifications (§5.1) (see page 46).

3.1.1. Details of retrieving a representation

Dereferencing a URI generally involves a succession of steps as described in multiple specifications and implemented by the agent. The following example illustrates the series of specifications that governs the process when a user agent is instructed to follow a hypertext link (§4.4) (see page 38) that is part of an SVG document. In this example, the URI is "http://weather.example.com/oaxaca" and the application context calls for the user agent to retrieve and render a representation of the identified resource.

1. Since the URI is part of a hypertext link in an SVG document, the first relevant specification is the SVG 1.1 Recommendation [SVG11 (see page 54)]. Section 17.1 of this specification imports the link semantics defined in XLink 1.0 [XLink10 (see page 55)]: "The remote resource (the destination for the link) is defined by a URI specified by the XLink `href` attribute on the 'a' element." The SVG specification goes on to state that interpretation of an `a` element involves retrieving a representation of a resource, identified by the `href` attribute in the XLink namespace: "By activating these links (by clicking with the mouse, through keyboard input, voice commands, etc.), users may visit these resources."
2. The XLink 1.0 [XLink10 (see page 55)] specification, which defines the `href` attribute in section 5.4, states that "The value of the href attribute must be a

URI reference as defined in [IETF RFC 2396], or must result in a URI reference after the escaping procedure described below is applied."
3. The URI specification [URI (see page 55)] states that "Each URI begins with a scheme name that refers to a specification for assigning identifiers within that scheme." The URI scheme name in this example is "http".
4. [IANASchemes (see page 52)] states that the "http" scheme is defined by the HTTP/1.1 specification (RFC 2616 [RFC2616 (see page 54)], section 3.2.2).
5. In this SVG context, the agent constructs an HTTP GET request (per section 9.3 of [RFC2616 (see page 54)]) to retrieve the representation.
6. Section 6 of [RFC2616 (see page 54)] defines how the server constructs a corresponding response message, including the 'Content-Type' field.
7. Section 1.4 of [RFC2616 (see page 54)] states "HTTP communication usually takes place over TCP/IP connections." This example addresses neither that step in the process nor other steps such as Domain Name System (DNS) resolution.
8. The agent interprets the returned representation according to the data format specification that corresponds to the representation's Internet Media Type (§3.2) (see page 22) (the value of the HTTP 'Content-Type') in the relevant IANA registry [MEDIATYPEREG].

Precisely which representation(s) are retrieved depends on a number of factors, including:

1. Whether the URI owner makes available any representations at all;
2. Whether the agent making the request has access privileges for those representations (see the section on linking and access control (§3.5.2) (see page 30));
3. If the URI owner has provided more than one representation (in different formats such as HTML, PNG, or RDF; in different languages such as English and Spanish; or transformed dynamically according to the hardware or software capabilities of the recipient), the resulting representation may depend on negotiation between the user agent and server.
4. The time of the request; the world changes over time, so representations of resources are also likely to change over time.

Assuming that a representation has been successfully retrieved, the expressive power of the representation's format will affect how precisely the representation provider communicates resource state. If the representation communicates the state of the resource inaccurately, this inaccuracy or ambiguity may lead to confusion among users about what the resource is. If different users reach different conclusions about what the resource is, they may interpret this as a URI collision (§2.2.1) (see page 12). Some communities, such as the ones developing the Semantic Web, seek to provide a framework for accurately communicating the semantics of a resource in a machine readable way. Machine readable semantics may alleviate some of the ambiguity associated with natural language descriptions of resources.

3.2. Representation Types and Internet Media Types

A ***representation*** is data that encodes information about resource state. Representations do not necessarily describe the resource, or portray a likeness of the resource, or represent the resource in other senses of the word "represent".

Representations of a resource may be sent or received using interaction protocols. These protocols in turn determine the form in which representations are conveyed on the Web. HTTP, for example, provides for transmission of representations as octet streams typed using Internet media types [RFC2046 (see page 53)].

Just as it is important to reuse existing URI schemes whenever possible, there are significant benefits to using media typed octet streams for representations even in the unusual case where a new URI scheme and associated protocol is to be defined. For example, if the Oaxaca weather were conveyed to Nadia's browser using a protocol other than HTTP, then software to render formats such as text/xhmtl+xml and image/png would still be usable if the new protocol supported transmission of those types. This is an example of the principle of orthogonal specifications (§5.1) (see page 46).

Good practice: Reuse representation formats

New protocols created for the Web SHOULD transmit representations as octet streams typed by Internet media types.

The Internet media type mechanism does have some limitations. For instance, media type strings do not support versioning (§4.2.1) (see page 33) or other parameters. See TAG issues uriMediaType-9 and mediaTypeManagement-45 which concern aspects of the media type mechanism.

3.2.1. Representation types and fragment identifier semantics

The Internet Media Type defines the syntax and semantics of the fragment identifier (introduced in Fragment Identifiers (§2.6) (see page 18)), if any, that may be used in conjunction with a representation.

Story

In one of his XHTML pages, Dirk creates a hypertext link to an image that Nadia has published on the Web. He creates a hypertext link with `<a href="http://www.example.com/images/nadia#hat">Nadia's hat</a>`. Emma views Dirk's XHTML page in her Web browser and follows the link. The HTML implementation in her browser removes the fragment from the URI and requests the image "http://www.example.com/images/nadia". Nadia serves an SVG representation of the image (with Internet media type "image/svg+xml"). Emma's Web browser starts up an SVG implementation to view the image. It passes it the original URI including the fragment, "http://www.example.com/images/nadia#hat" to this implementation, causing a view of the hat to be displayed rather than the complete image.

Note that the HTML implementation in Emma's browser did not need to *understand the syntax or semantics* of the SVG fragment (nor does the SVG implementation have to understand HTML, WebCGM, RDF ... fragment syntax or semantics; it merely had to recognize the # delimiter from the URI syntax [URI] and remove the fragment when accessing the resource). This orthogonality (§5.1) (see page 46) is

an important feature of Web architecture; it is what enabled Emma's browser to provide a useful service without requiring an upgrade.

The semantics of a fragment identifier are defined by the set of representations that might result from a retrieval action on the primary resource. The fragment's format and resolution are therefore dependent on the type of a potentially retrieved representation, even though such a retrieval is only performed if the URI is dereferenced. If no such representation exists, then the semantics of the fragment are considered unknown and, effectively, unconstrained. Fragment identifier semantics are orthogonal to URI schemes and thus cannot be redefined by URI scheme specifications.

Interpretation of the fragment identifier is performed solely by the agent that dereferences a URI; the fragment identifier is not passed to other systems during the process of retrieval. This means that some intermediaries in Web architecture (such as proxies) have no interaction with fragment identifiers and that redirection (in HTTP [RFC2616], for example) does not account for fragments.

3.2.2. Fragment identifiers and content negotiation

Content negotiation refers to the practice of making available multiple representations via the same URI. Negotiation between the requesting agent and the server determines which representation is served (usually with the goal of serving the "best" representation a receiving agent can process). HTTP is an example of a protocol that enables representation providers to use content negotiation.

Individual data formats may define their own rules for use of the fragment identifier syntax for specifying different types of subsets, views, or external references that are identifiable as secondary resources by that media type. Therefore, representation providers must manage content negotiation carefully when used with a URI that contains a fragment identifier. Consider an example where the owner of the URI "http://weather.example.com/oaxaca/map#zicatela" uses content negotiation to serve two representations of the identified resource. Three situations can arise:

1. The interpretation of "zicatela" is defined consistently by both data format specifications. The representation provider decides when definitions of fragment identifier semantics are are sufficiently consistent.
2. The interpretation of "zicatela" is defined inconsistently by the data format specifications.
3. The interpretation of "zicatela" is defined in one data format specification but not the other.

The first situation—consistent semantics—poses no problem.

The second case is a server management error: representation providers must not use content negotiation to serve representation formats that have inconsistent fragment identifier semantics. This situation also leads to URI collision (§2.2.1) (see page 12).

The third case is not a server management error. It is a means by which the Web can grow. Because the Web is a distributed system in which formats and agents are deployed in a non-uniform manner, Web architecture does not constrain authors to only use "lowest common denominator" formats. Content authors may take advantage of new data formats while still ensuring reasonable backward-compatibility for agents that do not yet implement them.

In case three, behavior by the receiving agent should vary depending on whether the negotiated format defines fragment identifier semantics. When a received data format does not define fragment identifier semantics, the agent should not perform silent error recovery (see page 48) unless the user has given consent; see [CUAP (see page 52)] for additional suggested agent behavior in this case.

See related TAG issue RDFinXHTML-35.

3.3. Inconsistencies between Representation Data and Metadata

Successful communication between two parties depends on a reasonably shared understanding of the semantics of exchanged messages, both data and metadata. At times, there may be inconsistencies between a message sender's data and metadata. Examples, observed in practice, of inconsistencies between representation data and metadata include:

- The actual character encoding of a representation (e.g., "iso-8859-1", specified by the `encoding` attribute in an XML declaration) is inconsistent with the charset parameter in the representation metadata (e.g., "utf-8", specified by the 'Content-Type' field in an HTTP header).
- The namespace (§4.5.3) (see page 40) of the root element of XML representation data (e.g., as specified by the "xmlns" attribute) is inconsistent with the value of the 'Content-Type' field in an HTTP header.

On the other hand, there is no inconsistency in serving HTML content with the media type "text/plain", for example, as this combination is licensed by specifications.

Receiving agents should detect protocol inconsistencies and perform proper error recovery.

Constraint: Data-metadata inconsistency

Agents MUST NOT ignore message metadata without the consent of the user.

Thus, for example, if the parties responsible for "weather.example.com" mistakenly label the satellite photo of Oaxaca as "image/gif" instead of "image/jpeg", and if Nadia's browser detects a problem, Nadia's browser must not ignore the problem (e.g., by simply rendering the JPEG image) without Nadia's consent. Nadia's browser can notify Nadia of the problem or notify Nadia and take corrective action.

Furthermore, representation providers can help reduce the risk of inconsistencies through careful assignment of representation metadata (especially that which applies across representations). The section on media types for XML (§4.5.7)

presents an example of reducing the risk of error by providing no metadata about character encoding when serving XML.

The accuracy of metadata relies on the server administrators, the authors of representations, and the software that they use. Practically, the capabilities of the tools and the social relationships may be the limiting factors.

The accuracy of these and other metadata fields is just as important for dynamic Web resources, where a little bit of thought and programming can often ensure correct metadata for a huge number of resources.

Often there is a separation of control between the users who create representations of resources and the server managers who maintain the Web site software. Given that it is generally the Web site software that provides the metadata associated with a resource, it follows that coordination between the server managers and content creators is required.

Good practice: Metadata association

Server managers SHOULD allow representation creators to control the metadata associated with their representations.

In particular, content creators need to be able to control the content type (for extensibility) and the character encoding (for proper internationalization).

The TAG finding *"Authoritative Metadata"* discusses in more detail how to handle data/metadata inconsistency and how server configuration can be used to avoid it.

3.4. Safe Interactions

Nadia's retrieval of weather information (an example of a read-only query or lookup) qualifies as a "safe" interaction; a ***safe interaction*** is one where the agent does not incur any obligation beyond the interaction. An agent may incur an obligation through other means (such as by signing a contract). If an agent does not have an obligation before a safe interaction, it does not have that obligation afterwards.

Other Web interactions resemble orders more than queries. These ***unsafe interactions*** may cause a change to the state of a resource and the user may be held responsible for the consequences of these interactions. Unsafe interactions include subscribing to a newsletter, posting to a list, or modifying a database. **Note:** In this context, the word "unsafe" does not necessarily mean "dangerous"; the term "safe" is used in section 9.1.1 of [RFC2616 (see page 54)] and "unsafe" is the natural opposite.

Story

Nadia decides to book a vacation to Oaxaca at "booking.example.com." She enters data into a series of online forms and is ultimately asked for credit card information to purchase the airline tickets. She provides this information in another form. When she presses the "Purchase" button, her browser opens another network connection to the server at "booking.example.com" and sends a message composed of form data using the POST method. This is an unsafe interaction (see page 26); Nadia wishes to change the state of the system by exchanging money for airline tickets.

The server reads the POST request, and after performing the booking transaction returns a message to Nadia's browser that contains a representation of the results of Nadia's request. The representation data is in XHTML so that it can be saved or printed out for Nadia's records.

Note that neither the data transmitted with the POST nor the data received in the response necessarily correspond to any resource identified by a URI.

Safe interactions are important because these are interactions where users can browse with confidence and where agents (including search engines and browsers that pre-cache data for the user) can follow hypertext links safely. Users (or agents acting on their behalf) do not commit themselves to anything by querying a resource or following a hypertext link.

Principle: Safe retrieval

Agents do not incur obligations by retrieving a representation.

For instance, it is incorrect to publish a URI that, when followed as part of a hypertext link, subscribes a user to a mailing list. Remember that search engines may follow such hypertext links.

The fact that HTTP GET, the access method most often used when following a hypertext link, is safe does not imply that all safe interactions must be done through HTTP GET. At times, there may be good reasons (such as confidentiality requirements or practical limits on URI length) to conduct an otherwise safe operation using a mechanism generally reserved for unsafe operations (e.g., HTTP POST).

For more information about safe and unsafe operations using HTTP GET and POST, and handling security concerns around the use of HTTP GET, see the TAG finding *"URIs, Addressability, and the use of HTTP GET and POST"*.

3.4.1. Unsafe interactions and accountability

Story

Nadia pays for her airline tickets online (through a POST interaction as described above). She receives a Web page with confirmation information and wishes to bookmark it so that she can refer to it when she calculates her expenses. Although Nadia can print out the results, or save them to a file, she would also like to bookmark them.

Transaction requests and results are valuable resources, and like all valuable resources, it is useful to be able to refer to them with a persistent URI (§3.5.1). However, in practice, Nadia cannot bookmark her commitment to pay (expressed via the POST request) or the airline company's acknowledgment and commitment to provide her with a flight (expressed via the response to the POST).

There are ways to provide persistent URIs for transaction requests and their results. For transaction requests, user agents can provide an interface for managing transactions where the user agent has incurred an obligation on behalf of the user. For transaction results, HTTP allows representation providers to associate a URI with the results of an HTTP POST request using the "Content-Location" header (described in section 14.14 of [RFC2616 (see page 54)]).

3.5. Representation Management

Story

Since Nadia finds the Oaxaca weather site useful, she emails a review to her friend Dirk recommending that he check out 'http://weather.example.com/oaxaca'. Dirk clicks on the resulting hypertext link in the email he receives and is frustrated by a 404 (not found). Dirk tries again the next day and receives a representation with "news" that is two-weeks old. He tries one more time the next day only to receive a representation that claims that the weather in Oaxaca is sunny, even though his friends in Oaxaca tell him by phone that in fact it is raining. Dirk and Nadia conclude that the URI owners are unreliable or unpredictable. Although the URI owner has chosen the Web as a communication medium, the owner has lost two customers due to ineffective representation management.

A URI owner may supply zero or more authoritative representations of the resource identified by that URI. There is a benefit to the community in providing representations.

Good practice: Available representation

A URI owner SHOULD provide representations of the resource it identifies

For example, owners of XML namespace URIs should use them to identify a namespace document (§4.5.4).

Just because representations are available does not mean that it is always desirable to retrieve them. In fact, in some cases the opposite is true.

Principle: Reference does not imply dereference

An application developer or specification author SHOULD NOT require networked retrieval of representations each time they are referenced.

Dereferencing a URI has a (potentially significant) cost in computing and bandwidth resources, may have security implications, and may impose significant latency on the dereferencing application. Dereferencing URIs should be avoided except when necessary.

The following sections discuss some aspects of representation management, including promoting URI persistence (§3.5.1), managing access to resources (§3.5.2) (see page 30), and supporting navigation (§3.5.3) (see page 30).

3.5.1. URI persistence

As is the case with many human interactions, confidence in interactions via the Web depends on stability and predictability. For an information resource, persistence depends on the consistency of representations. The representation provider decides when representations are sufficiently consistent (although that determination generally takes user expectations into account).

Although persistence in this case is observable as a result of representation retrieval, the term ***URI persistence*** is used to describe the desirable property that, once associated with a resource, a URI should continue indefinitely to refer to that resource.

Good practice: Consistent representation

A URI owner SHOULD provide representations of the identified resource consistently and predictably.

URI persistence is a matter of policy and commitment on the part of the URI owner (see page 12). The choice of a particular URI scheme provides no guarantee that those URIs will be persistent or that they will not be persistent.

HTTP [RFC2616 (see page 54)] has been designed to help manage URI persistence. For example, HTTP redirection (using the 3xx response codes) permits servers to tell an agent that further action needs to be taken by the agent in order to fulfill the request (for example, a new URI is associated with the resource).

In addition, content negotiation (see page 24) also promotes consistency, as a site manager is not required to define new URIs when adding support for a new format specification. Protocols that do not support content negotiation (such as FTP)

require a new identifier when a new data format is introduced. Improper use of content negotiation can lead to inconsistent representations.

For more discussion about URI persistence, see [Cool (see page 52)].

3.5.2. Linking and access control

It is reasonable to limit access to a resource (for commercial or security reasons, for example), but merely identifying the resource is like referring to a book by title. In exceptional circumstances, people may have agreed to keep titles or URIs confidential (for example, a book author and a publisher may agree to keep the URI of page containing additional material secret until after the book is published), otherwise they are free to exchange them.

As an analogy: The owners of a building might have a policy that the public may only enter the building via the main front door, and only during business hours. People who work in the building and who make deliveries to it might use other doors as appropriate. Such a policy would be enforced by a combination of security personnel and mechanical devices such as locks and pass-cards. One would not enforce this policy by hiding some of the building entrances, nor by requesting legislation requiring the use of the front door and forbidding anyone to reveal the fact that there are other doors to the building.

Story

Nadia sends to Dirk the URI of the current article she is reading. With his browser, Dirk follows the hypertext link and is asked to enter his subscriber username and password. Since Dirk is also a subscriber to services provided by "weather.example.com," he can access the same information as Nadia. Thus, the authority for "weather.example.com" can limit access to authorized parties and still provide the benefits of URIs.

The Web provides several mechanisms to control access to resources; these mechanisms do not rely on hiding or suppressing URIs for those resources. For more information, see the TAG finding *"'Deep Linking' in the World Wide Web"*.

3.5.3. Supporting Navigation

It is a strength of Web Architecture that links can be made and shared; a user who has found an interesting part of the Web can share this experience just by republishing a URI.

Story

Nadia and Dirk want to visit the Museum of Weather Forecasting in Oaxaca. Nadia goes to "http://maps.example.com", locates the museum, and mails the URI "http://maps.example.com/oaxaca?lat=17.065;lon=-96.716;scale=6" to Dirk. Dirk goes to "http://mymaps.example.com", locates the museum, and mails the URI "http://mymaps.example.com/geo?sessionID=765345;userID=Dirk" to Nadia. Dirk reads Nadia's email and is able to follow the link to the map. Nadia reads Dirk's email, follows the link, and receives an error message 'No such session/user'. Nadia has to start again from "http://mymaps.example.com" and find the museum location once more.

For resources that are generated on demand, machine generation of URIs is common. For resources that might usefully be bookmarked for later perusal, or shared with others, server managers should avoid needlessly restricting the reusability of such URIs. If the intention is to restrict information to a particular user, as might be the case in a home banking application for example, designers should use appropriate access control (§3.5.2) (see page 30) mechanisms.

Interactions conducted with HTTP POST (where HTTP GET could have been used) also limit navigation possibilities. The user cannot create a bookmark or share the URI because HTTP POST transactions do not typically result in a different URI as the user interacts with the site.

3.6. Future Directions for Interaction

There remain open questions regarding Web interactions. The TAG expects future versions of this document to address in more detail the relationship between the architecture described herein, Web Services, peer-to-peer systems, instant messaging systems (such as [RFC3920 (see page 54)]), streaming audio (such as RTSP [RFC2326]), and voice-over-IP (such as SIP [RFC3261 (see page 54)]).

4. Data Formats

A data format specification (for example, for XHTML, RDF/XML, SMIL, XLink, CSS, and PNG) embodies an agreement on the correct interpretation of representation data. The first data format used on the Web was HTML. Since then, data formats have grown in number. Web architecture does not constrain which data formats content providers can use. This flexibility is important because there is constant evolution in applications, resulting in new data formats and refinements of existing formats. Although Web architecture allows for the deployment of new data formats, the creation and deployment of new formats (and agents able to handle them) is expensive. Thus, before inventing a new data format (or "meta" format such as XML), designers should carefully consider re-using one that is already available.

For a data format to be usefully interoperable between two parties, the parties must agree (to a reasonable extent) about its syntax and semantics. Shared understanding of a data format promotes interoperability but does not imply constraints on usage; for instance, a sender of data cannot count on being able to constrain the behavior of a data receiver.

Below we describe some characteristics of a data format that facilitate integration into Web architecture. This document does not address generally beneficial characteristics of a specification such as readability, simplicity, attention to programmer goals, attention to user needs, accessibility, nor internationalization. The section on architectural specifications (§7.1) includes references to additional format specification guidelines.

4.1. Binary and Textual Data Formats

Binary data formats are those in which portions of the data are encoded for direct use by computer processors, for example 32 bit little-endian two's-complement and 64 bit IEEE double-precision floating-point. The portions of data so represented include numeric values, pointers, and compressed data of all sorts.

A textual data format is one in which the data is specified in a defined encoding as a sequence of characters. HTML, Internet e-mail, and all XML-based formats (§4.5) (see page 39) are textual. Increasingly, internationalized textual data formats refer to the Unicode repertoire [UNICODE (see page 54)] for character definitions.

If a data format is textual, as defined in this section, that does not imply that it should be served with a media type beginning with "text/". Although XML-based formats are textual, many XML-based formats do not consist primarily of phrases in natural language. See the section on media types for XML (§4.5.7) (see page 44) for issues that arise when "text/" is used in conjunction with an XML-based format.

In principle, all data can be represented using textual formats. In practice, some types of content (e.g., audio and video) are generally represented using binary formats.

The trade-offs between binary and textual data formats are complex and application-dependent. Binary formats can be substantially more compact, particularly for complex pointer-rich data structures. Also, they can be consumed more rapidly by agents in those cases where they can be loaded into memory and used with little or no conversion. Note, however, that such cases are relatively

uncommon as such direct use may open the door to security issues that can only practically be addressed by examining every aspect of the data structure in detail.

Textual formats are usually more portable and interoperable. Textual formats also have the considerable advantage that they can be directly read by human beings (and understood, given sufficient documentation). This can simplify the tasks of creating and maintaining software, and allow the direct intervention of humans in the processing chain without recourse to tools more complex than the ubiquitous text editor. Finally, it simplifies the necessary human task of learning about new data formats; this is called the "view source" effect.

It is important to emphasize that intuition as to such matters as data size and processing speed is not a reliable guide in data format design; quantitative studies are essential to a correct understanding of the trade-offs. Therefore, designers of a data format specification should make a considered choice between binary and textual format design.

See TAG issue binaryXML-30.

4.2. Versioning and Extensibility

In a perfect world, language designers would invent languages that perfectly met the requirements presented to them, the requirements would be a perfect model of the world, they would never change over time, and all implementations would be perfectly interoperable because the specifications would have no variability.

In the real world, language designers imperfectly address the requirements as they interpret them, the requirements inaccurately model the world, conflicting requirements are presented, and they change over time. As a result, designers negotiate with users, make compromises, and often introduce extensibility mechanisms so that it is possible to work around problems in the short term. In the long term, they produce multiple versions of their languages, as the problem, and their understanding of it, evolve. The resulting variability in specifications, languages, and implementations introduces interoperability costs.

Extensibility and versioning are strategies to help manage the natural evolution of information on the Web and technologies used to represent that information. For more information about how these strategies introduce variability and how that variability impacts interoperability, see Variability in Specifications.

See TAG issue XMLVersioning-41, which concerns good practices for designing extensible XML languages and for handling versioning. See also "Web Architecture: Extensible Languages" [EXTLANG (see page 56)].

4.2.1. Versioning

There is typically a (long) transition period during which multiple versions of a format, protocol, or agent are simultaneously in use.

Good practice: Version information

A data format specification SHOULD provide for version information.

4.2.2. Versioning and XML namespace policy

Story

Nadia and Dirk are designing an XML data format to encode data about the film industry. They provide for extensibility by using XML namespaces and creating a schema that allows the inclusion, in certain places, of elements from any namespace. When they revise their format, Nadia proposes a new optional `lang` attribute on the `film` element. Dirk feels that such a change requires them to assign a new namespace name, which might require changes to deployed software. Nadia explains to Dirk that their choice of extensibility strategy in conjunction with their namespace policy allows certain changes that do not affect conformance of existing content and software, and thus no change to the namespace identifier is required. They chose this policy to help them meet their goals of reducing the cost of change.

Dirk and Nadia have chosen a particular namespace change policy that allows them to avoid changing the namespace name whenever they make changes that do not affect conformance of deployed content and software. They might have chosen a different policy, for example that any new element or attribute has to belong to a namespace other than the original one. Whatever the chosen policy, it should set clear expectations for users of the format.

In general, changing the namespace name of an element completely changes the element name. If "a" and "b" are bound to two different URIs, `a:element` and `b:element` are as distinct as `a:eieio` and `a:xyzzy`. Practically speaking, this means that deployed applications will have to be upgraded in order to recognize the new language; the cost of this upgrade may be very high.

It follows that there are significant tradeoffs to be considered when deciding on a namespace change policy. If a vocabulary has no extensibility points (that is, if it does not allow elements or attributes from foreign namespaces or have a mechanism for dealing with unrecognized names from the same namespace), it may be absolutely necessary to change the namespace name. Languages that allow some form of extensibility without requiring a change to the namespace name are more likely to evolve gracefully.

Good practice: Namespace policy

An XML format specification SHOULD include information about change policies for XML namespaces.

As an example of a change policy designed to reflect the variable stability of a namespace, consider the W3C namespace policy for documents on the W3C Recommendation track. The policy sets expectations that the Working Group responsible for the namespace may modify it in any way until a certain point in the process ("Candidate Recommendation") at which point W3C constrains the set of possible changes to the namespace in order to promote stable implementations.

Note that since namespace names are URIs, the owner of a namespace URI has the authority to decide the namespace change policy.

4.2.3. Extensibility

Requirements change over time. Successful technologies are adopted and adapted by new users. Designers can facilitate the transition process by making careful choices about extensibility during the design of a language or protocol specification.

In making these choices, the designers must weigh the trade-offs between extensibility, simplicity, and variability. A language without extensibility mechanisms may be simpler and less variable, improving initial interoperability. However, it's likely that changes to that language will be more difficult, possibly more complex and more variable, than if the initial design had provided such mechanisms. This may decrease interoperability over the long term.

Good practice: Extensibility mechanisms

A specification SHOULD provide mechanisms that allow any party to create extensions.

Extensibility introduces variability which has an impact on interoperability. However, languages that have no extensibility mechanisms may be extended in ad hoc ways that impact interoperability as well. One key criterion of the mechanisms provided by language designers is that they allow the extended languages to remain in conformance with the original specification, increasing the likelihood of interoperability.

Good practice: Extensibility conformance

Extensibility MUST NOT interfere with conformance to the original specification.

Application needs determine the most appropriate extension strategy for a specification. For example, applications designed to operate in closed environments may allow specification designers to define a versioning strategy that would be impractical at the scale of the Web.

Good practice: Unknown extensions

A specification SHOULD specify agent behavior in the face of unrecognized extensions.

Two strategies have emerged as being particularly useful:

1. "Must ignore": The agent ignores any content it does not recognize.
2. "Must understand": The agent treats unrecognized markup as an error condition.

A powerful design approach is for the language to allow either form of extension, but to distinguish explicitly between them in the syntax.

Additional strategies include prompting the user for more input and automatically retrieving data from available hypertext links. More complex strategies are also possible, including mixing strategies. For instance, a language can include mechanisms for overriding standard behavior. Thus, a data format can specify "must ignore" semantics but also allow for extensions that override that semantics in light of application needs (for instance, with "must understand" semantics for a particular extension).

Extensibility is not free. Providing hooks for extensibility is one of many requirements to be factored into the costs of language design. Experience suggests that the long term benefits of a well-designed extensibility mechanism generally outweigh the costs.

See “D.3 Extensibility and Extensions” in [QA (see page 56)].

4.2.4. Composition of data formats

Many modern data format include mechanisms for composition. For example:

- It is possible to embed text comments in some image formats, such as JPEG/JFIF. Although these comments are embedded in the containing data, they are not intended to affect the display of the image.
- There are container formats such as SOAP which fully expect content from multiple namespaces but which provide an overall semantic relationship of message envelope and payload.
- The semantics of combining RDF documents containing multiple vocabularies are well-defined.

In principle, these relationships can be mixed and nested arbitrarily. A SOAP message, for example, can contain an SVG image that contains an RDF comment which refers to a vocabulary of terms for describing the image.

Note however, that for general XML there is no semantic model that defines the interactions within XML documents with elements and/or attributes from a variety of namespaces. Each application must define how namespaces interact and what effect the namespace of an element has on the element's ancestors, siblings, and descendants.

See TAG issues mixedUIXMLNamespace-33 (concerning the meaning of a document composed of content in multiple namespaces), xmlFunctions-34 (concerning one approach for managing XML transformation and composability), and RDFinXHTML-35 (concerning the interpretation of RDF when embedded in an XHTML document).

4.3. Separation of Content, Presentation, and Interaction

The Web is a heterogeneous environment where a wide variety of agents provide access to content to users with a wide variety of capabilities. It is good practice for authors to create content that can reach the widest possible audience, including users with graphical desktop computers, hand-held devices and mobile phones,

users with disabilities who may require speech synthesizers, and devices not yet imagined. Furthermore, authors cannot predict in some cases how an agent will display or process their content. Experience shows that the separation of content, presentation, and interaction promotes the reuse and device-independence of content; this follows from the principle of orthogonal specifications (§5.1) (see page 46).

This separation also facilitates reuse of authored source content across multiple delivery contexts. Sometimes, functional user experiences suited to any delivery context can be generated by using an adaptation process applied to a representation that does not depend on the access mechanism. For more information about principles of device-independence, see [DIPRINCIPLES (see page 56)].

Good practice: Separation of content, presentation, interaction

A specification SHOULD allow authors to separate content from both presentation and interaction concerns.

Note that when content, presentation, and interaction are separated by design, agents need to recombine them. There is a recombination spectrum, with "client does all" at one end and "server does all" at the other.

There are advantages to each approach. For instance when a client (such as a mobile phone) communicates device capabilities to the server (for example, using CC/PP), the server can tailor the delivered content to fit that client. The server can, for example, enable faster downloads by adjusting links to refer to lower resolution images, smaller video or no video at all. Similarly, if the content has been authored with multiple branches, the server can remove unused branches before delivery. In addition, by tailoring the content to match the characteristics of a target client, the server can help reduce client side computation. However, specializing content in this manner reduces caching efficiency.

On the other hand, designing content that that can be recombined on the client also tends to make that content applicable to a wider range of devices. This design also improves caching efficiency and offers users more presentation options. Media-dependent style sheets can be used to tailor the content on the client side to particular groups of target devices. For textual content with a regular and repeating structure, the combined size of the text content plus the style sheet is typically less than that of fully recombined content; the savings improve further if the style sheet is reused by other pages.

In practice a combination of both approaches is often used. The design decision about where on this spectrum an application should be placed depends on the power on the client, the power and the load on the server, and the bandwidth of the medium that connects them. If the number of possible clients is unbounded, the application will scale better if more computation is pushed to the client.

Of course, it may not be desirable to reach the widest possible audience. Designers should consider appropriate technologies, such as encryption and access control (§3.5.2), for limiting the audience.

Some data formats are designed to describe presentation (including SVG and XSL Formatting Objects). Data formats such as these demonstrate that one can only separate content from presentation (or interaction) so far; at some point it becomes necessary to talk about presentation. Per the principle of orthogonal specifications (§5.1) (see page 46) these data formats should *only* address presentation issues.

See the TAG issues formattingProperties-19 (concerning interoperability in the case of formatting properties and names) and contentPresentation-26 (concerning the separation of semantic and presentational markup).

4.4. Hypertext

A defining characteristic of the Web is that it allows embedded references to other resources via URIs. The simplicity of creating hypertext links using absolute URIs (`<a href="http://www.example.com/foo">`) and relative URI references (`<a href="foo">` and `<a href="foo#anchor">`) is partly (perhaps largely) responsible for the success of the hypertext Web as we know it today.

When a representation of one resource contains a reference to another resource, expressed with a URI identifying that other resource, this constitutes a ***link*** between the two resources. Additional metadata may also form part of the link (see [XLink10 (see page 55)], for example). **Note:** In this document, the term "link" generally means "relationship", not "physical connection".

Good practice: Link identification

A specification SHOULD provide ways to identify links to other resources, including to secondary resources (via fragment identifiers).

Formats that allow content authors to use URIs instead of local identifiers promote the network effect: the value of these formats grows with the size of the deployed Web.

Good practice: Web linking

A specification SHOULD allow Web-wide linking, not just internal document linking.

Good practice: Generic URIs

A specification SHOULD allow content authors to use URIs without constraining them to a limited set of URI schemes.

What agents do with a hypertext link is not constrained by Web architecture and may depend on application context. Users of hypertext links expect to be able to navigate among representations by following links.

Good practice: Hypertext links

A data format SHOULD incorporate hypertext links if hypertext is the expected user interface paradigm.

Data formats that do not allow content authors to create hypertext links lead to the creation of "terminal nodes" on the Web.

4.4.1. URI references

Links are commonly expressed using ***URI references*** (defined in section 4.2 of [URI (see page 55)]), which may be combined with a base URI to yield a usable URI. Section 5.1 of [URI (see page 55)] explains different ways to establish a base URI for a resource and establishes a precedence among them. For instance, the base URI may be a URI for the resource, or specified in a representation (see the `base` elements provided by HTML and XML, and the HTTP 'Content-Location' header). See also the section on links in XML (§4.5.2).

Agents resolve a URI reference before using the resulting URI to interact with another agent. URI references help in content management by allowing content authors to design a representation locally, i.e., without concern for which global identifier may later be used to refer to the associated resource.

4.5. XML-Based Data Formats

Many data formats are ***XML-based***, that is to say they conform to the syntax rules defined in the XML specification [XML10 (see page 55)] or [XML11 (see page 55)]. This section discusses issues that are specific to such formats. Anyone seeking guidance in this area is urged to consult the "Guidelines For the Use of XML in IETF Protocols" [IETFXML] (see page 52), which contains a thorough discussion of the considerations that govern whether or not XML ought to be used, as well as specific guidelines on how it ought to be used. While it is directed at Internet applications with specific reference to protocols, the discussion is generally applicable to Web scenarios as well.

The discussion here should be seen as ancillary to the content of [IETFXML] (see page 52). Refer also to "XML Accessibility Guidelines" [XAG] (see page 57) for help designing XML formats that lower barriers to Web accessibility for people with disabilities.

4.5.1. When to use an XML-based format

XML defines textual data formats that are naturally suited to describing data objects which are hierarchical and processed in a chosen sequence. It is widely, but not universally, applicable for data formats; an audio or video format, for example, is unlikely to be well suited to expression in XML. Design constraints that would suggest the use of XML include:

1. Requirement for a hierarchical structure.
2. Need for a wide range of tools on a variety of platforms.
3. Need for data that can outlive the applications that currently process it.

4. Ability to support internationalization in a self-describing way that makes confusion over coding options unlikely.
5. Early detection of encoding errors with no requirement to "work around" such errors.
6. A high proportion of human-readable textual content.
7. Potential composition of the data format with other XML-encoded formats.
8. Desire for data easily parsed by both humans and machines.
9. Desire for vocabularies that can be invented in a distributed manner and combined flexibly.

4.5.2. Links in XML

Sophisticated linking mechanisms have been invented for XML formats. XPointer allows links to address content that does not have an explicit, named anchor. [XLink (see page 55)] is an appropriate specification for representing links in hypertext (§4.4) (see page 38) XML applications. XLink allows links to have multiple ends and to be expressed either inline or in "link bases" stored external to any or all of the resources identified by the links it contains.

Designers of XML-based formats may consider using XLink and, for defining fragment identifier syntax, using the XPointer framework and XPointer element() Schemes.

XLink is not the only linking design that has been proposed for XML, nor is it universally accepted as a good design. See also TAG issue xlinkScope-23.

4.5.3. XML namespaces

The purpose of an XML namespace (defined in [XMLNS (see page 55)]) is to allow the deployment of XML vocabularies (in which element and attribute names are defined) in a global environment and to reduce the risk of name collisions in a given document when vocabularies are combined. For example, the MathML and SVG specifications both define the `set` element. Although XML data from different formats such as MathML and SVG can be combined in a single document, in this case there could be ambiguity about which `set` element was intended. XML namespaces reduce the risk of name collisions by taking advantage of existing systems for allocating globally scoped names: the URI system (see also the section on URI allocation (§2.2.2)). When using XML namespaces, each local name in an XML vocabulary is paired with a URI (called the namespace URI) to distinguish the local name from local names in other vocabularies.

The use of URIs confers additional benefits. First, each URI/local name pair can be mapped to another URI, grounding the terms of the vocabulary in the Web. These terms may be important resources and thus it is appropriate to be able to associate URIs with them.

[RDFXML (see page 53)] uses simple concatenation of the namespace URI and the local name to create a URI for the identified term. Other mappings are likely to be more suitable for hierarchical namespaces; see the related TAG issue abstractComponentRefs-37.

Designers of XML-based data formats who declare namespaces thus make it possible to reuse those data formats and combine them in novel ways not yet imagined. Failure to declare namespaces makes such reuse more difficult, even impractical in some cases.

Good practice: Namespace adoption

A specification that establishes an XML vocabulary SHOULD place all element names and global attribute names in a namespace.

Attributes are always scoped by the element on which they appear. An attribute that is "global," that is, one that might meaningfully appear on elements of many types, including elements in other namespaces, should be explicitly placed in a namespace. Local attributes, ones associated with only a particular element type, need not be included in a namespace since their meaning will always be clear from the context provided by that element.

The `type` attribute from the W3C XML Schema Instance namespace "http://www.w3.org/2001/XMLSchema-instance" ([XMLSCHEMA (see page 56)], section 4.3.2) is an example of a global attribute. It can be used by authors of any vocabulary to make an assertion in instance data about the type of the element on which it appears. As a global attribute, it must always be qualified. The `frame` attribute on an HTML table is an example of a local attribute. There is no value in placing that attribute in a namespace since the attribute is unlikely to be useful on an element other than an HTML table.

Applications that rely on DTD processing must impose additional constraints on the use of namespaces. DTDs perform validation based on the lexical form of the element and attribute names in the document. This makes prefixes syntactically significant in ways that are not anticipated by [XMLNS (see page 55)].

4.5.4. Namespace documents

Story

Nadia receives representation data from "weather.example.com" in an unfamiliar data format. She knows enough about XML to recognize which XML namespace the elements belong to. Since the namespace is identified by the URI "http://weather.example.com/2003/format", she asks her browser to retrieve a representation of the identified resource. She gets back some useful data that allows her to learn more about the data format. Nadia's browser may also be able to perform some operations automatically (i.e., unattended by a human overseer) given data that has been optimized for software agents. For example, her browser might, on Nadia's behalf, download additional agents to process and render the format.

Another benefit of using URIs to build XML namespaces is that the namespace URI can be used to identify an information resource that contains useful information, machine-usable and/or human-usable, about terms in the namespace. This type of

information resource is called a ***namespace document***. When a namespace URI owner provides a namespace document, it is authoritative for the namespace.

There are many reasons to provide a namespace document. A person might want to:

- understand the purpose of the namespace,
- learn how to use the markup vocabulary in the namespace,
- find out who controls it and associated policies,
- request authority to access schemas or collateral material about it, or
- report a bug or situation that could be considered an error in some collateral material.

A processor might want to:

- retrieve a schema, for validation,
- retrieve a style sheet, for presentation, or
- retrieve ontologies, for making inferences.

In general, there is no established best practice for creating representations of a namespace document; application expectations will influence what data format or formats are used. Application expectations will also influence whether relevant information appears directly in a representation or is referenced from it.

Good practice: Namespace documents

The owner of an XML namespace name SHOULD make available material intended for people to read and material optimized for software agents in order to meet the needs of those who will use the namespace vocabulary.

For example, the following are examples of data formats for namespace documents: [OWL10 (see page 53)], [RDDL (see page 53)], [XMLSCHEMA (see page 56)], and [XHTML11]. Each of these formats meets different requirements described above for satisfying the needs of an agent that wants more information about the namespace. Note, however, issues related to fragment identifiers and content negotiation (§3.2.2) (see page 24) if content negotiation is used.

See TAG issues namespaceDocument-8 (concerning desired characteristics of namespace documents) and abstractComponentRefs-37 (concerning the use of fragment identifiers with namespace names to identify abstract components).

4.5.5. QNames in XML

Section 3 of "Namespaces in XML" [XMLNS (see page 55)] provides a syntactic construct known as a QName for the compact expression of qualified names in XML documents. A qualified name is a pair consisting of a URI, which names a namespace, and a local name placed within that namespace. "Namespaces in XML" provides for the use of QNames as names for XML elements and attributes.

Other specifications, starting with [XSLT10 (see page 56)], have employed the idea of using QNames in contexts other than element and attribute names, for example

in attribute values and in element content. However, general XML processors cannot reliably recognize QNames as such when they are used in attribute values and in element content; for example, the syntax of QNames overlaps with that of URIs. Experience has also revealed other limitations to QNames, such as losing namespace bindings after XML canonicalization.

Constraint: QNames Indistinguishable from URIs

Do not allow both QNames and URIs in attribute values or element content where they are indistinguishable.

For more information, see the TAG finding *"Using QNames as Identifiers in Content"*.

Because QNames are compact, some specification designers have adopted the same syntax as a means of identifying resources. Though convenient as a shorthand notation, this usage has a cost. There is no single, accepted way to convert a QName into a URI or vice versa. Although QNames are convenient, they do not replace the URI as the identification system of the Web. The use of QNames to identify Web resources without providing a mapping to URIs is inconsistent with Web architecture.

Good practice: QName Mapping

A specification in which QNames serve as resource identifiers MUST provide a mapping to URIs.

See XML namespaces (§4.5.3) (see page 40) for examples of some mapping strategies.

See also TAG issues rdfmsQnameUriMapping-6 (concerning the mapping of QNames to URIs), qnameAsId-18 (concerning the use of QNames as identifiers in XML content), and abstractComponentRefs-37 (concerning the use of fragment identifiers with namespace names to identify abstract components).

4.5.6. XML ID semantics

Consider the following fragment of XML: `<section name="foo">`. Does the `section` element have what the XML Recommendation refers to as the ID `foo` (i.e., "foo" must not appear in the surrounding XML document more than once)? One cannot answer this question by examining the element and its attributes alone. In XML, the quality of "being an ID" is associated with the type of an attribute, not its name. Finding the IDs in a document requires additional processing.

1. Processing the document with a processor that recognizes DTD attribute list declarations (in the external or internal subset) might reveal a declaration that identifies the `name` attribute as an ID. **Note:** This processing is not necessarily part of validation. A non-validating, DTD-aware processor can recognize IDs.
2. Processing the document with a W3C XML schema might reveal an element declaration that identifies the `name` attribute as an W3C XML Schema `ID`.

3. In practice, processing the document with another schema language, such as RELAX NG [RELAXNG (see page 53)], might reveal the attributes declared to be of ID in the XML Schema sense. Many modern specifications begin processing XML at the Infoset [INFOSET (see page 52)] level and do not specify normatively how an Infoset is constructed. For those specifications, any process that establishes the ID type in the Infoset (and Post Schema Validation Infoset (PSVI) defined in [XMLSCHEMA (see page 56)]) may usefully identify the attributes of type ID.
4. In practice, applications may have independent means (such as those defined in the XPointer specification, [XPTRFR (see page 56)] section 3.2) of locating identifiers inside a document.

To further complicate matters, DTDs establish the ID type in the Infoset whereas W3C XML Schema produces a PSVI but does not modify the original Infoset. This leaves open the possibility that a processor might only look in the Infoset and consequently would fail to recognize schema-assigned IDs.

See the TAG issue xmlIDSemantics-32 for additional background information and [XML-ID (see page 55)] for a solution under development.

4.5.7. Media types for XML

RFC 3023 defines the Internet media types "application/xml" and "text/xml", and describes a convention whereby XML-based data formats use Internet media types with a "+xml" suffix, for example "image/svg+xml".

There are two problems associated with the “text” media types: First, for data identified as "text/*", Web intermediaries are allowed to "transcode", i.e., convert one character encoding to another. Transcoding may make the self-description false or may cause the document to be not well-formed.

Good practice: XML and "text/*"

In general, a representation provider SHOULD NOT assign Internet media types beginning with "text/" to XML representations.

Second, representations whose Internet media types begin with "text/" are required, unless the `charset` parameter is specified, to be considered to be encoded in US-ASCII. Since the syntax of XML is designed to make documents self-describing, it is good practice to omit the `charset` parameter, and since XML is very often not encoded in US-ASCII, the use of "text/" Internet media types effectively precludes this good practice.

Good practice: XML and character encodings

In general, a representation provider SHOULD NOT specify the character encoding for XML data in protocol headers since the data is self-describing.

4.5.8. Fragment identifiers in XML

The section on media types and fragment identifier semantics (§3.2.1) discusses the interpretation of fragment identifiers. Designers of an XML-based data format specification should define the semantics of fragment identifiers in that format. The XPointer Framework [XPTRFR (see page 56)] provides an interoperable starting point.

When the media type assigned to representation data is "application/xml", there are no semantics defined for fragment identifiers, and authors should not make use of fragment identifiers in such data. The same is true if the assigned media type has the suffix "+xml" (defined in "XML Media Types" [RFC3023 (see page 54)]), and the data format specification does not specify fragment identifier semantics. In short, just knowing that content is XML does not provide information about fragment identifier semantics.

Many people assume that the fragment identifier `#abc`, when referring to XML data, identifies the element in the document with the ID "abc". However, there is no normative support for this assumption. A revision of RFC 3023 is expected to address this.

See TAG issue fragmentInXML-28.

4.6. Future Directions for Data Formats

Data formats enable the creation of new applications to make use of the information space infrastructure. The Semantic Web is one such application, built on top of RDF [RDFXML (see page 53)]. This document does not discuss the Semantic Web in detail; the TAG expects that future volumes of this document will. See the related TAG issue httpRange-14.

5. General Architecture Principles

A number of general architecture principles apply to all three bases of Web architecture.

5.1. Orthogonal Specifications

Identification, interaction, and representation are orthogonal concepts, meaning that technologies used for identification, interaction, and representation may evolve independently. For instance:

- Resources are identified with URIs. URIs can be published without building any representations of the resource or determining whether any representations are available.
- A generic URI syntax allows agents to function in many cases without knowing specifics of URI schemes.
- In many cases one may change the representation of a resource without disrupting references to the resource (for example, by using content negotiation (see page 24)).

When two specifications are orthogonal, one may change one without requiring changes to the other, even if one has dependencies on the other. For example, although the HTTP specification depends on the URI specification, the two may evolve independently. This orthogonality increases the flexibility and robustness of the Web. For example, one may refer by URI to an image without knowing anything about the format chosen to represent the image. This has facilitated the introduction of image formats such as PNG and SVG without disrupting existing references to image resources.

Principle: Orthogonality

Orthogonal abstractions benefit from orthogonal specifications.

Experience demonstrates that problems arise where orthogonal concepts occur in a single specification. Consider, for example, the HTML specification which includes the orthogonal x-www-form-urlencoded specification. Software developers (for example, of [CGI (see page 52)] applications) might have an easier time finding the specification if it were published separately and then cited from the HTTP, URI, and HTML specifications.

Problems also arise when specifications attempt to modify orthogonal abstractions described elsewhere. An historical version of the HTML specification added a "`Refresh`" value to the `http-equiv` attribute of the `meta` element. It was defined to be equivalent to the HTTP header of the same name. The authors of the HTTP specification ultimately decided not to provide this header and that made the two specifications awkwardly at odds with each other. The W3C HTML Working Group eventually removed the "`Refresh`" value.

A specification should clearly indicate which features overlap with those governed by another specification.

5.2. Extensibility

The information in the Web and the technologies used to represent that information change over time. Extensibility is the property of a technology that promotes evolution without sacrificing interoperability. Some examples of successful technologies designed to allow change while minimizing disruption include:

- the fact that URI schemes are orthogonally specified;
- the use of an open set of Internet media types in mail and HTTP to specify document interpretation;
- the separation of the generic XML grammar and the open set of XML namespaces for element and attribute names;
- extensibility models in Cascading Style Sheets (CSS), XSLT 1.0, and SOAP;
- user agent plug-ins.

An example of an unsuccessful extension mechanism is HTTP mandatory extensions [HTTPEXT]. The community has sought mechanisms to extend HTTP, but apparently the costs of the mandatory extension proposal (notably in complexity) outweighed the benefits and thus hampered adoption.

Below we discuss the property of "extensibility," exhibited by URIs, some data formats, and some protocols (through the incorporation of new messages).

Subset language: one language is a subset (or "profile") of a second language if any document in the first language is also a valid document in the second language and has the same interpretation in the second language.

Extended language: If one language is a subset of another, the latter superset is called an extended language; the difference between the languages is called the extension. Clearly, extending a language is better for interoperability than creating an incompatible language.

Ideally, many instances of a superset language can be safely and usefully processed as though they were in the subset language. Languages that can evolve this way, allowing applications to provide new information when necessary while still interoperating with applications that only understand a subset of the current language, are said to be "extensible." Language designers can facilitate extensibility by defining the default behavior of unknown extensions—for example, that they be ignored (in some defined way) or should be considered errors.

For example, from early on in the Web, HTML agents followed the convention of ignoring unknown tags. This choice left room for innovation (i.e., non-standard elements) and encouraged the deployment of HTML. However, interoperability problems arose as well. In this type of environment, there is an inevitable tension between interoperability in the short term and the desire for extensibility. Experience shows that designs that strike the right balance between allowing change and preserving interoperability are more likely to thrive and are less likely to disrupt the Web community. Orthogonal specifications (§5.1) (see page 46) help reduce the risk of disruption.

For further discussion, see the section on versioning and extensibility (§4.2) (see page 33). See also TAG issue xmlProfiles-29 and HTML Dialects.

5.3. Error Handling

Errors occur in networked information systems. An error condition can be well-characterized (e.g., well-formedness errors in XML or 4xx client errors in HTTP) or arise unpredictably. ***Error correction*** means that an agent repairs a condition so that within the system, it is as though the error never occurred. One example of error correction involves data retransmission in response to a temporary network failure. ***Error recovery*** means that an agent does not repair an error condition but continues processing by addressing the fact that the error has occurred.

Agents frequently *correct* errors without user awareness, sparing users the details of complex network communications. On the other hand, it is important that agents *recover* from error in a way that is evident to users, since the agents are acting on their behalf.

> ***Principle: Error recovery***
>
> Agents that recover from error by making a choice without the user's consent are not acting on the user's behalf.

An agent is not required to interrupt the user (e.g., by popping up a confirmation box) to obtain consent. The user may indicate consent through pre-selected configuration options, modes, or selectable user interface toggles, with appropriate reporting to the user when the agent detects an error. Agent developers should not ignore usability issues when designing error recovery behavior.

To promote interoperability, specification designers should identify predictable error conditions. Experience has led to the following observations about error-handling approaches.

- Protocol designers should provide enough information about an error condition so that an agent can address the error condition. For instance, an HTTP 404 status code (not found) is useful because it allows user agents to present relevant information to users, enabling them to contact the representation provider in case of problems.
- Experience with the cost of building a user agent to handle the diverse forms of ill-formed HTML content convinced the designers of the XML specification to require that agents fail upon encountering ill-formed content. Because users are unlikely to tolerate such failures, this design choice has pressured all parties into respecting XML's constraints, to the benefit of all.
- An agent that encounters unrecognized content may handle it in a number of ways, including by considering it an error; see also the section on extensibility and versioning (§4.2).
- Error behavior that is appropriate for a person may not be appropriate for software. People are capable of exercising judgement in ways that software applications generally cannot. An informal error response may suffice for a person but not for a processor.

See the TAG issue contentTypeOverride-24, which concerns the source of authoritative metadata.

5.4. Protocol-based Interoperability

The Web follows Internet tradition in that its important interfaces are defined in terms of protocols, by specifying the syntax, semantics, and sequencing constraints of the messages interchanged. Protocols designed to be resilient in the face of widely varying environments have helped the Web scale and have facilitated communication across multiple trust boundaries. Traditional application programming interfaces (APIs) do not always take these constraints into account, nor should they be required to. One effect of protocol-based design is that the technology shared among agents often lasts longer than the agents themselves.

It is common for programmers working with the Web to write code that generates and parses these messages directly. It is less common, but not unusual, for end users to have direct exposure to these messages. It is often desirable to provide users with access to format and protocol details: allowing them to “view source,” whereby they may gain expertise in the workings of the underlying system.

6. Glossary

Content negotiation (see page 24)
The practice of providing multiple representations available via the same URI. Which representation is served depends on negotiation between the requesting agent and the agent serving the representations.

Dereference a URI (see page 20)
Access a representation of the resource identified by the URI.

Error correction (see page 48)
An agent repairs an error so that within the system, it is as though the error never occurred.

Error recovery (see page 48)
An agent invokes exceptional behavior because it does not correct the error.

Extended language (see page 47)
If one language is a subset of another, the latter is called an extended language.

Fragment identifier (see page 18)
The part of a URI that allows identification of a secondary resource.

Information resource (see page 11)
A resource which has the property that all of its essential characteristics can be conveyed in a message.

Link (see page 38)
A relationship between two resources when one resource (representation) refers to the other resource by means of a URI.

Message (see page 20)
A unit of communication between agents.

Namespace document (see page 42)
An information resource identified by an XML Namespace URI that contains useful information, machine-usable and/or human-usable, about terms in a particular XML namespace. It is useful, though not manditory, that the URI employed as a namespace name identifies a namespace document.

Representation (see page 22)
Data that encodes information about resource state.

Resource (see page 10)
Anything that might be identified by a URI.

Safe interaction (see page 26)
Interaction with a resource where an agent does not incur any obligation beyond the interaction.

Secondary resource (see page 18)
A resource related to another resource through the primary resource with additional identifying information (the fragment identifier).

Subset language (see page 47)
One language is a subset of a second language if any document in the first language is also a valid document in the second language and has the same interpretation in the second language.

URI (see page 5)
Acronym for Uniform Resource Identifier.

URI aliases (see page 11)
Two or more different URIs that that identify the same resource.

URI collision (see page 12)
The use of the same URI to refer to more than one resource in the context of Web protocols and formats.

URI ownership (see page 12)
A relationship between a URI and a social entity, such as a person, organization, or specification.

URI persistence (see page 29)
The social expectation that once a URI identifies a particular resource, it should continue indefinitely to refer to that resource.

URI reference (see page 39)
An operational shorthand for a URI.

Uniform Resource Identifier (URI) (see page 10)
A global identifier in the context of the World Wide Web.

Unsafe interaction (see page 26)
Interaction with a resource that is not safe interaction.

User agent (see page 5)
One type of Web agent; a piece of software acting on behalf of a person.

WWW (see page 5)
Acronym for World Wide Web.

Web (see page 5)
Shortened form of World Wide Web.

Web agent (see page 5)
A person or a piece of software acting on the information space on behalf of a person, entity, or process.

World Wide Web (see page 5)
An information space in which items of interest are identified by Uniform Resource Identifiers.

XML-based format (see page 39)
One that conforms to the syntax rules defined in the XML specification.

7. References

CGI

Common Gateway Interface/1.1 Specification. Available at http://hoohoo.ncsa.uiuc.edu/cgi/interface.html.

CHIPS

Common HTTP Implementation Problems, O. Théreaux, January 2003. This W3C Team Submission is available at http://www.w3.org/TR/chips/.

CUAP

Common User Agent Problems, K. Dubost, January 2003. This W3C Team Submission is available at http://www.w3.org/TR/cuap.

Cool

Cool URIs don't change T. Berners-Lee, W3C, 1998 Available at http://www.w3.org/Provider/Style/URI. Note that the title is somewhat misleading. It is not the URIs that change, it is what they identify.

Eng90

Knowledge-Domain Interoperability and an Open Hyperdocument System, D. C. Engelbart, June 1990.

HTTPEXT

Mandatory Extensions in HTTP, H. Frystyk Nielsen, P. Leach, S. Lawrence, 20 January 1998. This expired Internet Draft is available at http://www.w3.org/Protocols/HTTP/ietf-http-ext/draft-frystyk-http-mandatory.

IANASchemes

IANA's online registry of URI Schemes is available at http://www.iana.org/assignments/uri-schemes.

IETFXML

IETF *Guidelines For The Use of XML in IETF Protocols*, S. Hollenbeck, M. Rose, L. Masinter, eds., 2 November 2002. This Internet Draft is available at http://www.imc.org/ietf-xml-use/xml-guidelines-07.txt. If this document is no longer available, refer to the ietf-xml-use mailing list.

INFOSET

XML Information Set (Second Edition), R. Tobin, J. Cowan, Editors, W3C Recommendation, 04 February 2004, http://www.w3.org/TR/2004/REC-xml-infoset-20040204. Latest version available at http://www.w3.org/TR/xml-infoset.

IRI

IETF *Internationalized Resource Identifiers (IRIs)*, M. Dürst, M. Suignard, Eds., November 2004. In an 8 December 2004 announcement, the IESG approved draft-duerst-iri-11 as a Proposed Standard of the IETF. References to the IRI specification in Volume One of *Architecture of the World Wide Web* are to that Proposed Standard. Once the IETF issues a Request For Comments (RFC) number for the specification, this W3C Recommendation may be updated to refer to that RFC. See also the latest information about the IRI specification.

MEDIATYPEREG

IANA's online registry of Internet Media Types is available at http://www.iana.org/assignments/media-types/index.html.

OWL10
OWL Web Ontology Language Reference, M. Dean, G. Schreiber, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/. Latest version available at http://www.w3.org/TR/owl-ref/.

P3P10
The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, M. Marchiori, Editor, W3C Recommendation, 16 April 2002, http://www.w3.org/TR/2002/REC-P3P-20020416/. Latest version available at http://www.w3.org/TR/P3P/.

RDDL
Resource Directory Description Language (RDDL), J. Borden, T. Bray, eds., 1 June 2003. This document is available at http://www.tbray.org/tag/rddl/rddl3.html.

RDFXML
RDF/XML Syntax Specification (Revised), D. Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Latest version available at http://www.w3.org/TR/rdf-syntax-grammar.

RELAXNG
The RELAX NG schema language project.

REST
Representational State Transfer (REST), Chapter 5 of "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Thesis of R. T. Fielding, 2000. Designers of protocol specifications in particular should invest time in understanding the REST model and the relevance of its principles to a given design. These principles include statelessness, clear assignment of roles to parties, uniform address space, and a limited, uniform set of verbs. Available at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

RFC2045
IETF *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, N. Freed, N. Borenstein, November 1996. Available at http://www.ietf.org/rfc/rfc2045.txt.

RFC2046
IETF *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, N. Freed, N. Borenstein, November 1996. Available at http://www.ietf.org/rfc/rfc2046.txt.

RFC2119
IETF *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt.

RFC2141
IETF *RFC 2141: URN Syntax*, R. Moats, May 1997. Available at http://www.ietf.org/rfc/rfc2141.txt.

RFC2326
IETF *RFC 2326: Real Time Streaming Protocol (RTSP)*, H. Schulzrinne, A. Rao, R. Lanphier, April 1998. Available at: http://www.ietf.org/rfc/rfc2326.txt.

RFC2397
IETF *RFC 2397: The “data” URL scheme*, L. Masinter, August 1998. Available at: http://www.ietf.org/rfc/rfc2397.txt.

RFC2616
IETF *RFC 2616: Hypertext Transfer Protocol - HTTP/1.1*, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

RFC2717
IETF *Registration Procedures for URL Scheme Names*, R. Petke, I. King, November 1999. Available at http://www.ietf.org/rfc/rfc2717.txt.

RFC2718
IETF *RFC 2718: Guidelines for new URL Schemes*, L. Masinter, H. Alvestrand, D. Zigmond, R. Petke, November 1999. Available at: http://www.ietf.org/rfc/rfc2718.txt.

RFC2818
IETF *RFC 2818: HTTP Over TLS*, E. Rescorla, May 2000. Available at: http://www.ietf.org/rfc/rfc2818.txt.

RFC3023
IETF *RFC 3023: XML Media Types*, M. Murata, S. St. Laurent, D. Kohn, January 2001. Available at: http://www.ietf.org/rfc/rfc3023.txt

RFC3236
IETF *RFC 3236: The 'application/xhtml+xml' Media Type*, M. Baker, P. Stark, January 2002. Available at: http://www.ietf.org/rfc/rfc3236.txt

RFC3261
IETF *RFC 3261: SIP: Session Initiation Protocol*, J. Rosenberg, H. Schulzrinne, G. Camarillo, et. al., June 2002. Available at: http://www.ietf.org/rfc/rfc3261.txt

RFC3920
IETF *RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core*, P. Saint-Andre, Ed., October 2004. Available at: http://www.ietf.org/rfc/rfc3920.txt

RFC977
IETF *RFC 977: Network News Transfer Protocol*, B. Kantor, P. Lapsley, February 1986. Available at http://www.ietf.org/rfc/rfc977.txt.

SOAP12
SOAP Version 1.2 Part 1: Messaging Framework, J. Moreau, N. Mendelsohn, H. Frystyk Nielsen, *et. al.,* Editors, W3C Recommendation, 24 June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/. Latest version available at http://www.w3.org/TR/soap12-part1/.

SVG11
Scalable Vector Graphics (SVG) 1.1 Specification, 藤沢 淳, J. Ferraiolo, D. Jackson, Editors, W3C Recommendation, 14 January 2003, http://www.w3.org/TR/2003/REC-SVG11-20030114/. Latest version available at http://www.w3.org/TR/SVG11/.

UNICODE
The Unicode Consortium, The Unicode Standard, Version 4, ISBN 0-321-18578-1, as updated from time to time by the publication of new versions. See http://www.unicode.org/unicode/standard/versions for the latest Unicode version and additional information on versions of the standard and of the Unicode Character Database.

URI

IETF *Uniform Resource Identifiers (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, Eds., September 2004. In an 18 October 2004 announcement, the IESG approved draft-fielding-uri-rfc2396bis-07 as a Full Standard of the IETF. References to the URI specification in Volume One of *Architecture of the World Wide Web* are to that Full Standard. Once the IETF issues a Request For Comments (RFC) number for the specification, this W3C Recommendation may be updated to refer to that RFC. See also the latest information about the URI specification.

UniqueDNS

IAB Technical Comment on the Unique DNS Root, B. Carpenter, 27 September 1999. Available at http://www.icann.org/correspondence/iab-tech-comment-27sept99.htm.

VOICEXML2

Voice Extensible Markup Language (VoiceXML) Version 2.0, B. Porter, A. Hunt, K. Rehor, *et. al.,* Editors, W3C Recommendation, 16 March 2004, http://www.w3.org/TR/2004/REC-voicexml20-20040316/. Latest version available at http://www.w3.org/TR/voicexml20.

XHTML11

XHTML™ 1.1 - Module-based XHTML, S. McCarron, M. Altheim, Editors, W3C Recommendation, 31 May 2001, http://www.w3.org/TR/2001/REC-xhtml11-20010531. Latest version available at http://www.w3.org/TR/xhtml11/.

XLink10

XML Linking Language (XLink) Version 1.0, E. Maler, S. DeRose, D. Orchard, Editors, W3C Recommendation, 27 June 2001, http://www.w3.org/TR/2001/REC-xlink-20010627/. Latest version available at http://www.w3.org/TR/xlink/.

XML-ID

xml:id Version 1.0, D. Veillard, J. Marsh, Editors, W3C Working Draft (work in progress), 07 April 2004, http://www.w3.org/TR/2004/WD-xml-id-20040407. Latest version available at http://www.w3.org/TR/xml-id/.

XML10

Extensible Markup Language (XML) 1.0 (Third Edition), F. Yergeau, J. Paoli, C. M. Sperberg-McQueen, *et. al.,* Editors, W3C Recommendation, 04 February 2004, http://www.w3.org/TR/2004/REC-xml-20040204. Latest version available at http://www.w3.org/TR/REC-xml.

XML11

Extensible Markup Language (XML) 1.1, J. Paoli, C. M. Sperberg-McQueen, J. Cowan, *et. al.,* Editors, W3C Recommendation, 04 February 2004, http://www.w3.org/TR/2004/REC-xml11-20040204/. Latest version available at http://www.w3.org/TR/xml11/.

XMLNS

Namespaces in XML 1.1, R. Tobin, D. Hollander, A. Layman, *et. al.,* Editors, W3C Recommendation, 04 February 2004, http://www.w3.org/TR/2004/REC-xml-names11-20040204. Latest version available at http://www.w3.org/TR/xml-names11/.

XMLSCHEMA
XML Schema Part 1: Structures, D. Beech, M. Maloney, H. S. Thompson, *et. al.,* Editors, W3C Recommendation, 02 May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/. Latest version available at http://www.w3.org/TR/xmlschema-1/.

XPTRFR
XPointer Framework, E. Maler, N. Walsh, P. Grosso, *et. al.,* Editors, W3C Recommendation, 25 March 2003, http://www.w3.org/TR/2003/REC-xptr-framework-20030325/. Latest version available at http://www.w3.org/TR/xptr-framework/.

XSLT10
XSL Transformations (XSLT) Version 1.0, J. Clark, Editor, W3C Recommendation, 16 November 1999, http://www.w3.org/TR/1999/REC-xslt-19991116. Latest version available at http://www.w3.org/TR/xslt.

7.1. Architectural Specifications

ATAG10
Authoring Tool Accessibility Guidelines 1.0, C. McCathieNevile, I. Jacobs, J. Treviranus, *et. al.,* Editors, W3C Recommendation, 03 February 2000, http://www.w3.org/TR/2000/REC-ATAG10-20000203. Latest version available at http://www.w3.org/TR/ATAG10.

CHARMOD
Character Model for the World Wide Web 1.0: Fundamentals, R. Ishida, M. J. Dürst, M. Wolf, *et. al.,* Editors, W3C Working Draft (work in progress), 25 February 2004, http://www.w3.org/TR/2004/WD-charmod-20040225/. Latest version available at http://www.w3.org/TR/charmod/.

DIPRINCIPLES
Device Independence Principles, R. Gimson, Editor, W3C Note, 01 September 2003, http://www.w3.org/TR/2003/NOTE-di-princ-20030901/. Latest version available at http://www.w3.org/TR/di-princ/.

EXTLANG
Web Architecture: Extensible Languages, T. Berners-Lee, D. Connolly, 10 February 1998. This W3C Note is available at http://www.w3.org/TR/1998/NOTE-webarch-extlang-19980210.

Fielding
Principled Design of the Modern Web Architecture, R.T. Fielding and R.N. Taylor, UC Irvine. In Proceedings of the 2000 International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, June 2000, pp. 407-416. This document is available at http://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf.

QA
QA Framework: Specification Guidelines, D. Hazaël-Massieux, L. Rosenthal, L. Henderson, *et. al.,* Editors, W3C Working Draft (work in progress), 30 August 2004, http://www.w3.org/TR/2004/WD-qaframe-spec-20040830/. Latest version available at http://www.w3.org/TR/qaframe-spec/.

RFC1958
IETF *RFC 1958: Architectural Principles of the Internet*, B. Carpenter, June 1996. Available at http://www.ietf.org/rfc/rfc1958.txt.

SPECVAR

Variability in Specifications, L. Rosenthal, D. Hazaël-Massieux, Editors, W3C Working Draft (work in progress), 30 August 2004, http://www.w3.org/TR/2004/WD-spec-variability-20040830/. Latest version available at http://www.w3.org/TR/spec-variability/.

UAAG10

User Agent Accessibility Guidelines 1.0, J. Gunderson, I. Jacobs, E. Hansen, Editors, W3C Recommendation, 17 December 2002, http://www.w3.org/TR/2002/REC-UAAG10-20021217/. Latest version available at http://www.w3.org/TR/UAAG10/.

WCAG20

Web Content Accessibility Guidelines 2.0, W. Chisholm, J. White, B. Caldwell, *et. al.,* Editors, W3C Working Draft (work in progress), 30 July 2004, http://www.w3.org/TR/2004/WD-WCAG20-20040730/. Latest version available at http://www.w3.org/TR/WCAG20/.

WSA

Web Services Architecture, D. Booth, F. McCabe, E. Newcomer, *et. al.,* Editors, W3C Note, 11 February 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. Latest version available at http://www.w3.org/TR/ws-arch/.

XAG

XML Accessibility Guidelines, S. B. Palmer, C. McCathieNevile, D. Dardailler, Editors, W3C Working Draft (work in progress), 03 October 2002, http://www.w3.org/TR/2002/WD-xag-20021003. Latest version available at http://www.w3.org/TR/xag.

8. Acknowledgments

This document was authored by the W3C Technical Architecture Group which included the following participants: Tim Berners-Lee (co-Chair, W3C), Tim Bray (Antarctica Systems), Dan Connolly (W3C), Paul Cotton (Microsoft Corporation), Roy Fielding (Day Software), Mario Jeckle (Daimler Chrysler), Chris Lilley (W3C), Noah Mendelsohn (IBM), David Orchard (BEA Systems), Norman Walsh (Sun Microsystems), and Stuart Williams (co-Chair, Hewlett-Packard).

The TAG appreciates the many contributions on the TAG's public mailing list, www-tag@w3.org (archive), which have helped to improve this document.

In addition, contributions by David Booth, Erik Bruchez, Kendall Clark, Karl Dubost, Bob DuCharme, Martin Duerst, Olivier Fehr, Al Gilman, Tim Goodwin, Elliotte Rusty Harold, Tony Hammond, Sandro Hawke, Ryan Hayes, Dominique Hazaël-Massieux, Masayasu Ishikawa, David M. Karr, Graham Klyne, Jacek Kopecky, Ken Laskey, Susan Lesch, Håkon Wium Lie, Frank Manola, Mark Nottingham, Bijan Parsia, Peter F. Patel-Schneider, David Pawson, Michael Sperberg-McQueen, Patrick Stickler, and Yuxiao Zhao are gratefully acknowledged.