The OpenURL Framework for Context-Sensitive Services

Abstract: The OpenURL Framework Standard defines an architecture for creating OpenURL Framework Applications. An OpenURL Framework Application is a networked service environment, in which packages of information are transported over a network. These packages have a description of a referenced resource at their core, and they are transported with the intent of obtaining context-sensitive services pertaining to the referenced resource. To enable the recipients of these packages to deliver such context-sensitive services, each package describes the referenced resource itself, the network context in which the resource is referenced, and the context in which the service request takes place.

This Standard specifies how to construct these packages as Representations of abstract information constructs called ContextObjects. To this end, the OpenURL Framework Standard defines the following core components: Character Encoding, Serialization, Constraint Language, ContextObject Format, Metadata Format, and Namespace. In addition, this Standard defines Transport, a core component that enables communities to specify how to transport ContextObject Representations. Finally, this Standard specifies how a community can deploy a new OpenURL Framework Application by defining a new Community Profile, the last core component.

This Standard defines the OpenURL Framework Registry and the rules that govern the usage of this Registry. The OpenURL Framework Registry contains all instances of all core components created by communities that have deployed OpenURL Framework Applications. This Standard defines and registers the initial content of the OpenURL Framework Registry, thereby deploying two distinct OpenURL Framework Applications.

An American National Standard
Developed by the
National Information Standards Organization

Approved: April 15, 2005
by the
American National Standards Institute
Reaffirmed May 13, 2010

Published by the National Information Standards Organization
Baltimore, Maryland, U.S.A.
About NISO Standards

NISO Standards are developed by the Working Groups and Standing Committees of the National Information Standards Organization. The development process is a strenuous one that includes a rigorous peer review of proposed standards open to each NISO Voting Member and any other interested party. Final approval of the standard involves verification by the American National Standards Institute that its requirements for due process, consensus, and other approval criteria have been met by NISO. Once verified and approved, NISO Standards also become American National Standards.

This standard may be revised or withdrawn at any time. For current information on the status of this standard contact the NISO office or visit the NISO website at:
http://www.niso.org

Published by

NISO
3600 Clipper Mill Road
Suite 302
Baltimore, MD 21211
www.niso.org

Copyright © 2010 by the National Information Standards Organization

All rights reserved under International and Pan-American Copyright Conventions. For noncommercial purposes only, this publication may be reproduced or transmitted in any form or by any means without prior permission in writing from the publisher, provided it is reproduced accurately, the source of the material is identified, and the NISO copyright status is acknowledged. All inquiries regarding translations into other languages or commercial reproduction or distribution should be addressed to: NISO 3600 Clipper Mill Road, Suite 302, Baltimore, MD 21211.

ISSN: 1041-5653  National Information Standards series
ISBN: 978-1-937522-38-4
## Contents

### Foreword

1. Purpose and Scope ................................................................. 1

2. Referenced Standards ............................................................ 2

3. Notational Conventions .......................................................... 4

4. Definitions ............................................................................. 5

### Part 1: ContextObjects and Transports

5. ContextObject, Entity, and Descriptor ........................................... 11
   - 5.1 ContextObject and Entity .................................................. 11
   - 5.2 Descriptor ........................................................................... 12
     - 5.2.1 Identifier ....................................................................... 12
     - 5.2.2 By-Value Metadata ...................................................... 13
     - 5.2.3 By-Reference Metadata ................................................. 13
     - 5.2.4 Private Data ................................................................. 13
   - 5.3 Constraints ......................................................................... 14

6. Registry .................................................................................. 15
   - 6.1 Registry Entries ............................................................... 15
   - 6.2 Registry Identifiers ........................................................... 17
   - 6.3 Using the Registry ............................................................ 17

7. Formats .................................................................................. 18
   - 7.1 Serializations [Registry] .................................................... 19
   - 7.2 Constraint Languages [Registry] ........................................ 20
   - 7.3 Constraint Definitions ...................................................... 20

8. Representing ContextObjects .................................................... 21
   - 8.1 Character Encodings [Registry] ........................................ 21
   - 8.2 ContextObject Formats [Registry] ...................................... 22

9. Representing Entities ............................................................... 24
   - 9.1 Namespaces [Registry] ...................................................... 24
   - 9.2 Metadata Formats [Registry] .............................................. 25

10. Transporting ContextObject Representations: Transports [Registry] ... 27

11. Defining Applications: Community Profiles [Registry] .................... 28

### Part 2: The KEV ContextObject Format

12. The KEV ContextObject Format ............................................... 33
   - 12.1 The KEV Serialization .................................................... 33
   - 12.2 The Z39.88-2004 Matrix Constraint Language .................. 34
   - 12.3 Constraint Definitions in the KEV ContextObject Format ...... 35
     - 12.3.1 Z39.88-2004 Matrix Constraint Definition for the KEV ContextObject Format .... 35
     - 12.3.2 Z39.88-2004 Matrix Constraint Definitions for KEV Metadata Formats ............. 36
Examples

Example 1: Examples of Entities ................................................................. 12
Example 2: Identifiers for a Referent, Requester, and Resolver ........................... 12
Example 3: By-Value Metadata for a Referent .......................................................... 13
Example 4: By-Reference Metadata for a Requester ...................................................... 13
Example 5: Private Data for a Referent ................................................................. 14
Example 6: A Registry Entry .............................................................................. 16
Example 7: Identification of a Character Encoding .................................................... 22
Example 8: Identification of a ContextObject Format ................................................ 24
Example 9: Identification of Entities using Identifiers from Namespaces ................. 25
Example 10: Identification of Unregistered Metadata Formats .................................. 27
Example 11: KEV ContextObject Representation ................................................... 38
Example 12: Identifier Descriptors in a KEV ContextObject Representation .................. 41
Example 13: By-Value Metadata Descriptor in a KEV ContextObject Representation ...... 43
Example 14: By-Reference Metadata Descriptor as a Property List ............................ 43
Example 15: Private Data Descriptor in a KEV ContextObject Representation ................ 44
Example 16: KEV ContextObject Representation ................................................... 45
Example 17: XML ContextObject Representation ................................................... 63
Example 18: Identifier Descriptors in an XML ContextObject Representation ............. 67
Example 19: Referent with a By-Value Metadata Descriptor ....................................... 68
Example 20: Requester with a By-Reference Metadata Descriptor ............................. 69
Example 21: ReferringEntity with a Private Data Descriptor ........................................ 69
Example 22: XML ContextObject Representation ................................................... 70
Example 23: By-Reference OpenURL Transport using HTTP GET ............................. 76
Example 24: By-Reference OpenURL Transport using HTTP POST ............................ 77
Example 25: By-Value OpenURL Transport using HTTP GET .................................... 79
Example 26: By-Value OpenURL Transport using HTTP POST .................................... 80
Example 27: Inline OpenURL Transport using HTTP GET ....................................... 83
Example 28: An HTML Form (POST Method) to generate an Inline OpenURL Transport ... 85
Example 29: Inline OpenURL Transport using HTTP POST ....................................... 85
Foreword

(This foreword is not part of The OpenURL Framework for Context Sensitive Services, ANSI/NISO Z39.88-2004 (R2010). It is included for information only.)

History

As the World Wide Web began its explosive growth in the early 1990s, the scholarly-information community made available digital scholarly materials, consisting of metadata and full-text content. As this body of materials grew, it became increasingly difficult to provide adequate links between related information assets, distributed across many collections and controlled by different custodians. By 1999, the scholarly-information community had embarked on several linking efforts, surveyed in Van de Sompel and Hochstenbach [R1].

In 1999, NISO started a series of invitational workshops to explore issues in the area of reference linking. Representatives from the library, publishing, and information services communities identified the appropriate-copy problem as a major issue, because its solution was expected to solve other link-resolution problems. The appropriate-copy problem arises when multiple copies of a resource exist, and each copy is governed by a different access policy. A specific user should be directed to a copy of the resource that is governed by an access policy compatible with that user’s access privileges. None of the proposed linking architectures could accomplish this.

A series of collaborations by Herbert Van de Sompel (Ghent University, Los Alamos National Laboratory, and Cornell University), Patrick Hochstenbach (Ghent University), and Oren Beit-Arie (Ex Libris) culminated in the solution of the appropriate-copy problem. Their solution also addressed related issues in the delivery of context-sensitive services for the web-based scholarly information environment. This collaboration resulted in:

- Development of the SFX linking server and the OpenURL architecture [R1] [R2] [R3].
- Publication of the OpenURL 0.1 specification that defines an HTTP GET syntax for transporting metadata and identifiers from an information service to a linking server. The transported metadata and identifiers describe a referenced item and some contextual information [R4].
- Publication of the OpenURL Framework, which provides a design for context-sensitive reference linking in the Web-based scholarly information environment [R5].

The scholarly-information community quickly embraced the OpenURL 0.1 specification. Publishers, vendors of abstracting and indexing databases, preprint systems, and CrossRef (a Registration Agency for Digital Object Identifiers or DOIs) introduced OpenURLs in their systems. Many libraries implemented OpenURL-conformant linking servers that provide their users with context-sensitive links. This quick adoption by so many constituents established OpenURL 0.1 as a de-facto standard.

In preparation for the NISO Standardization effort, Herbert Van de Sompel and Oren Beit-Arie studied OpenURLs in real environments and analyzed the information sent to OpenURL resolvers. Based on this analysis, they proposed the Bison-Futé model [R6], a generalization of OpenURL 0.1 based on the notion of a ContextObject. A ContextObject is an information construct that formalizes and generalizes the information packaged in OpenURL 0.1 requests:
<table>
<thead>
<tr>
<th>OpenURL 0.1</th>
<th>ContextObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>The OpenURL 0.1 specification explicitly includes the referenced resource, the system that provides the OpenURL (sid), and the linking server that is the target of the OpenURL.</td>
<td>The ContextObject definition formalizes these resources into, respectively, the Referent, the Referrer, and the Resolver Entities.</td>
</tr>
<tr>
<td>OpenURL 0.1 usage showed that three more resources were regularly described in the Private Identifier (pid): the initiator of an OpenURL transport (the user clicking the OpenURL), the citing scholarly work, and the type of service requested (for example, “provide full-text”).</td>
<td>The ContextObject definition formalizes these resources into, respectively, the Requester, the ReferringEntity, and the ServiceType Entities.</td>
</tr>
<tr>
<td>OpenURL 0.1 specifies one set of metadata keys to describe a referenced scholarly work by inline metadata.</td>
<td>The ContextObject definition generalizes this description method into the By-Value Metadata Descriptor of Entities. This Descriptor depends on Metadata Formats that are made available through registration in the OpenURL Framework Registry.</td>
</tr>
<tr>
<td>OpenURL 0.1 specifies namespaces for identifiers of referenced scholarly works, for the system that provides the OpenURL, and for the target of the OpenURL.</td>
<td>The ContextObject definition generalizes this description method into the Identifier Descriptor of Entities. This Descriptor depends on Namespaces for Identifiers that are made available through registration in the OpenURL Framework Registry.</td>
</tr>
<tr>
<td>OpenURL 0.1 allows for private data to describe a referenced scholarly work by a method that is specific to the provider of the OpenURL. To process this syntax, a linking server must enter into some prior agreement with the provider of the OpenURL.</td>
<td>The ContextObject definition generalizes this description method into the Private Data Descriptor of Entities.</td>
</tr>
<tr>
<td>OpenURL 0.1 usage showed that the Private Identifier (pid) of OpenURL 0.1 often contains a pointer to metadata describing the referenced scholarly work.</td>
<td>The ContextObject definition generalizes this description method into the By-Reference Metadata Descriptor of Entities. This Descriptor depends on Metadata Formats that are made available through registration in the OpenURL Framework Registry.</td>
</tr>
</tbody>
</table>

In 2001, NISO formed Committee AX to prepare this Standard. The Committee’s charge [R7] was to develop an extensible mechanism for the representation and transportation of packages of metadata and identifiers that are useful in the delivery of context-sensitive services. The table above shows how the OpenURL 0.1 specification inspired the ContextObject concept of the Bison-Futé model. The Committee took this concept as the starting point for its work. To achieve extensibility, the Committee embedded the ContextObject concept in a general (and abstract) framework, called the OpenURL Framework.

This framework is defined in Part 1 and consists of the following core components: Namespaces for Identifiers, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles. To create an instantiation of the OpenURL...
Framework for a particular application domain, a community must specify and register specific selections of these core components in the OpenURL Framework Registry.

Although it retains the name OpenURL in its title for historical reasons, the OpenURL Framework is neutral with respect to application domain. The Committee hopes that the ContextObject specified in this Standard will become a generic component for systems providing contextual services pertaining to resources that are referenced on networks.

References


Preamble

Wide adoption of any technology or process is often a result of its simplicity coupled with the effective meeting of a market need. OpenURL has been embraced and adopted by the scholarly-information community for these very reasons.

The early implementation of OpenURL was simple in concept: it used HTTP GET or POST to transfer information about an item (a journal article, for example) from an online service to a linking server. The specifications were simple. They described the protocol, the syntax, and how a referenced item is to be represented by using particular sets of data element names on a URL. This is best demonstrated by an example of an OpenURL:
Because of this simplicity, developers working for online service providers and developers of linking servers could quickly understand how OpenURL worked and develop their own products. The scholarly-information community responded by quickly adopting link resolvers as a basic component of its digital library infrastructure.

The Committee recognized the status of this early version of OpenURL as a de facto standard. The OpenURL Framework Standard (Z39.88-2004) refers to the early version as OpenURL 0.1, and the OpenURL 0.1 specifications are retained in the OpenURL Framework Registry <http://www.openurl.info/registry/docs/pdf/openurl-01.pdf>.

The simplicity of OpenURL 0.1 is due, in part, to the static nature of its syntax, the limited number of genres supported, the fixed sets of data elements, and the fixed transport protocol (HTTP). However, the fixed nature of OpenURL 0.1 not only limits its expansion within the scholarly-information community, it also prevents other communities from adopting OpenURL for similar needs. For example, OpenURL 0.1 cannot be extended to cover other genres of materials. It cannot even extend the data element sets for existing genres. The OpenURL Framework Standard is about providing such extensibility.

In developing this Standard, the Committee wanted to provide the needed extensibility while retaining the simplicity of the original OpenURL. To accomplish this, the fundamentals of an extensible framework had to be put in place. This document describes these fundamentals: the framework upon which OpenURL can be extended with new genres, new data elements, different character encodings (to support non-English use of the OpenURL), various network protocols, and data representations.

Part 1 (Sections 5 through 11) describes the ContextObject. The ContextObject is the information construct that describes an item that is the subject of a service request and the context within which the request is being made. While the term ContextObject may be new, the concept it represents is entirely compatible with OpenURL 0.1. Indeed, the original OpenURLs were precisely about a request to provide a service (for example, asking a link server to provide a menu of relevant links) expressed in an HTTP link, whereby the HTTP link described an item and provided some context within which it was referenced. Part 1 formalizes the expression of the item description, its context, and the service being requested. The definitions of all concepts are separated from their representation and the protocol by which the representations are transported.

Very few bounds are placed on how ContextObjects can be extended or applied. The Committee did not want to prescribe the limits on what kinds of creative applications there might be for ContextObjects and the OpenURL Framework in other communities. For example, the Committee could imagine storing ContextObjects in databases or using ContextObjects as the containers to transfer item and context information between servers in a web services environment. Two linking servers could talk to one another by exchanging ContextObjects. In the latter scenario, ContextObjects might be transported using an XML-based protocol such as SOAP. OpenURL 0.1 did not provide such capabilities.

Such abstraction and open-endedness is sometimes disconcerting for the development community. System providers may be reluctant to invest development resources if they fear that the Standard is
too general to be able to create interoperable solutions or that the Standard may change without their knowledge or involvement. To address these issues, the Committee introduced the notions of the OpenURL Framework Registry, OpenURL Framework Application, and Community Profile.

The OpenURL Framework Registry <http://www.openurl.info/registry> provides a mechanism for the public disclosure of specific selections for the representation and transportation of ContextObjects. For the purpose of this discussion, these selections will be described as registered entries. Each registered entry is assigned a unique identifier so it may be referenced unambiguously.

An OpenURL Framework Application is one instantiation of the OpenURL Framework meant for a specific community of adopters in a particular application domain. In essence, Part 1 specifies how a community can define their own OpenURL Framework Application. In general terms, a community defines an OpenURL Framework Application by selecting entries from the Registry it needs to represent and transport ContextObjects. If necessary and/or desired, the community may define new entries and, subject approval of Registry administrators, register these new entries.

A Community Profile is an unambiguous summary of one OpenURL Framework Application. For an implementer, the Community Profile unambiguously specifies the scope and boundaries of compliance by listing a selection of registered entries that OpenURL Framework Applications within that community are expected to support. To prevent a given Community Profile from becoming a moving target for a developer, the Committee envisions the Registry being under strict version control. When a community chooses to evolve its OpenURL Framework Application, it develops a new Community Profile. It may have to create new entries, register them, and have new unique identifiers assigned to them.

Part 2 (Sections 12 through 15) defines a ContextObject Format inspired by the query string of the HTTP(S) GET request as specified in OpenURL 0.1. The Key/Encoded-Value or KEV ContextObject Format defines how to represent a ContextObject as a concatenation of ampersand-delimited Key/Encoded-Value pairs. The foremost purpose of the KEV ContextObject Format is backward compatibility. It provides an elegant transition from the OpenURL 0.1 specification to this Standard.

Part 3 (Sections 16 through 19) defines a ContextObject Format based on XML (eXtensible Markup Language). XML Documents are widely used in the exchange of structured text and data between computer applications. The XML ContextObject Format is about the future. Using the full expressive power of the XML syntax, ContextObjects can convey greater detail, which Resolvers can use to provide more appropriate services.

Part 4 (Sections 20 through 22) specifies mechanisms by which ContextObject Representations can be transported using the HTTP(S) protocol. Collectively, these are called OpenURL Transports.

Parts 2, 3, and 4 define the initial content of the OpenURL Framework Registry, which is sufficient to deploy two OpenURL Framework Applications. These two Applications are defined by two Community Profiles: the Level 1 San Antonio Community Profile (SAP1) and the Level 2 San Antonio Community Profile (SAP2). They are defined, respectively, in Appendix C and Appendix D.

SAP1 formalizes the OpenURL 0.1 specification under the new Standard and adds a few of the enhancements requested by the scholarly-information community. To further assist developers in transitioning their existing application to one that complies with SAP1, a set of Implementation Guidelines have been created to provide step-by-step instructions <http://www.openurl.info/registry/docs/implementation_guidelines>.

To illustrate how SAP1 retains the original simplicity of OpenURL 0.1, let us convert the OpenURL 0.1 sample given earlier to an OpenURL that conforms with the OpenURL Framework Application defined by SAP1:

&url_ctx_fmt=info:ofi/fmt:kev:mtx:ctx
&rft_val_fmt=info:ofi/fmt:kev:mtx:journal
Three new tags have been added for version control and format declaration (to describe to the linking server what follows), and prefixes have been added to the tags to avoid ambiguity.

With tools like the Implementation Guidelines, the San Antonio Profiles, and the OpenURL Framework Registry, the OpenURL Framework Standard extends the ideas underlying the original OpenURL to new and creative uses, while it retains the simplicity of its predecessor.

**Technical Considerations**

Recognizing the international environments in which ContextObjects will be used, the Committee selected Unicode as the abstract character repertoire for ContextObjects. Without excluding other encoding forms, the Committee selected UTF-8 as the default encoding form of the Unicode Coded Character Set.

This Standard originated in the scholarly-information community for the purpose of providing context-sensitive linking services. The significantly more general OpenURL Framework is a reflection of NISO’s charge to the Committee to develop an extensible Standard. Extensibility is implemented through the OpenURL Framework Registry. Initially, this Registry contains entries that support the creation of OpenURL Framework Applications in the scholarly-information community. However, other user communities may add new entries to support different applications. The Registry records the following:

- To support the representation of ContextObjects and the resources of which ContextObjects convey descriptions:
  - Character Encodings
  - Formats to express ContextObjects, including the Serializations, Constraint Languages, and Constraint Definitions used by those Formats. For example, the XML ContextObject Format uses XML as its Serialization and is constrained by an XML Schema.
  - Namespaces used to identify resources of which ContextObjects contain descriptions
  - Metadata Formats used to represent particular classes of resources of which ContextObjects contain descriptions
- Methods to transport ContextObject Representations
- Community Profiles that list selections of the above made by specific communities for their OpenURL Framework Application

The initial Registry contains two Formats to express ContextObjects: the Key/Encoded-Value Format and the XML Format. Communities may define and register new ContextObject Formats, thereby enabling the creation of new OpenURL Framework Applications. The initial Registry also contains a suite of HTTP(S)-based methods to transport representations of ContextObjects. Two Community Profiles are included in the initial content of the Registry. The Committee created these
to provide support for the existing OpenURL 0.1 application as used in the scholarly-information community under this Standard.

The Level 1 San Antonio Community Profile (SAP1): A Community Profile that is based on the Key/Encoded-Value Format to represent ContextObjects. It uses Namespaces and Metadata Formats that are important to the scholarly-information community. In the definition of this Community Profile, care has been taken to provide a certain level of backward compatibility with the OpenURL 0.1 specification, while at the same time providing enhanced capabilities to describe referenced resources and the network context in which the references occur.

The Level 2 San Antonio Community Profile (SAP2): A Community Profile that is based on the XML Format to represent ContextObjects. It uses Namespaces and Metadata Formats that are important to the scholarly-information community. It introduces a new level of expressiveness to describe referenced resources and the network context in which the references occur.

Reaffirmation

This standard underwent a periodic review in 2009-2010. It was approved for reaffirmation by NISO on February 12, 2012 and by ANSI on May 13, 2010.

Suggestions for improving this standard are welcome. They should be sent to the National Information Standards Organization, 3600 Clipper Mill Road, Suite 302, Baltimore, MD 21211, telephone (301) 654 2512.

Trademarks, Service Marks

Wherever used in this Standard, all terms that are trademarks or service marks are and remain the property of their respective owners.

This Standard was processed and approved for submittal to ANSI by the National Information Standards Organization. It was balloted by the NISO Voting Members January 26, 2004-March 10, 2004. This Standard will be up for review in 2009. Suggestions for improving this Standard are welcome. They should be sent to the National Information Standards Organization, 4733 Bethesda Avenue, Suite 300, Bethesda, MD  20814. NISO approval of this Standard does not imply that all Voting Members voted for its approval.

Disclaimer

Use of this Standard is voluntary. NISO accepts no responsibility for any loss or damage caused to any person or organization as result of any error, omission, or misleading statement in the information presented in this Standard or due to implementing this Standard.

NISO Voting Members

At the time it approved the Standard in 2005, NISO had the following Voting Members:

3M
Susan Boettcher, John Nelson (Alt)

American Association of Law Libraries
Robert L. Oakley, Mary Alice Baish (Alt)

American Chemical Society
tba

American Library Association
Betty Landesman

American Society for Information Science and Technology (ASIS&T)
Gail Thornburg

American Society of Indexers
Judith Gibbs

American Theological Library Association
Myron Chace

ARMA International
Diane Carlisle
Armed Forces Medical Library  
Diane Zehnpfennig, Emily Court (Alt)

Art Libraries Society of North America  
(ARLIS/NA)  
Sarah McCleskey

AllM International  
Betsy A. Fanning

Association of Information and Dissemination Centers (ASIDIC)  
Marjorie Hlava

Association of Jewish Libraries  
Caroline R. Miller, Elizabeth Vernon (Alt)

Association of Research Libraries (ARL)  
Duane E. Webster, Julia Blixrud (Alt)

Auto-Graphics, Inc.  
Paul Cope

Barnes & Noble, Inc.  
Douglas Cheney

Book Industry Communication  
Brian Green

California Digital Library  
Daniel Greenstein, John Kunze (Alt)

Cambridge Information Group  
Michael Cairns, Matthew Dunie (Alt)

College Center for Library Automation (CCLA)  
J. Richard Madaus, Ann Armbrister (Alt)

Colorado State Library  
Brenda Bailey-Hainer, Steve Wrede (Alt)

CrossRef  
Edward Pentz, Amy Brand (Alt)

Davandy, L.L.C.  
Michael J. Mellinger

Docutek Information Systems  
Philip Kesten, Slaven Zivkovic (Alt)

Dynix Corporation  
Lynn Thackeray, Gail Wanner (Alt)

EBSCO Information Services  
Gary Coker, Oliver Pesch (Alt)

Elsevier Science Inc.  
Anthony Ross, John Mancia (Alt)

Endeavor Information Systems, Inc.  
Verne Coppi, Cindy Miller (Alt)

Entopia, Inc.  
Igor Perisic

Ex Libris  
James Steenbergan

Fretwell-Downing Informatics  
Matthew Goldner, Robin Murray (Alt)

Gale Group  
Katherine Gruber, Justine Carson (Alt)

Geac Library Solutions  
Eric Conderaerts, Eloise Sullivan (Alt)

GIS Information Systems, Inc.  
Candy Zemon, Paul Huf (Alt)

H.W. Wilson Company  
Ann Case, Patricia Kuhr (Alt)

Helsinki University Library  
Juha Hakala

Index Data  
Sebastian Hammer, David Dorman (Alt)

Information Use Management and Policy Institute/FSU  
Charles McClure, John Carlo Bertot (Alt)

Infotrieve  
Jan Peterson

Innovative Interfaces, Inc.  
Gerald M. Kline, Betsy Graham (Alt)

Institute for Scientific Information  
Carolyn Finn

The International DOI Foundation  
Norman Paskin

John Wiley & Sons, Inc.  
Eric A. Swanson

Library Binding Institute  
Joanne Rock

Library of Congress  
Sally H. McCallum

The Library Corporation  
Mark Wilson, Ted Koppel (Alt)

Los Alamos National Laboratory  
Richard E. Luce

Lucent Technologies  
M. E. Brennan

Medical Library Association  
Nadine P. Ellero, Carla J. Funk (Alt)

MINITEX  
Cecelia Boone, William DeJohn (Alt)

Modern Language Association  
Daniel Bokser, B. Chen (Alt)

Motion Picture Association of America  
Axel aus der Muhlen

MuseGlobal, Inc.  
Kate Noerr, Clifford Hammond (Alt)

Music Library Association  
Mark McKnight, David Summerfield (Alt)

National Agricultural Library  
Eleanor G. Frierson, Gary K. McCone (Alt)
National Archives and Records Administration
Nancy Allard

National Federation of Abstracting and Information Services (NFAIS)
Marjorie Hlava

National Library of Medicine
Betsy L. Humphreys

National Security Agency
Kathleen Dolan

Nylink
Mary-Alice Lynch, Jane Neale (Alt)

OCLC, Inc.
Larry Olszewski

Openly Informatics, Inc.
Eric Hellman

ProQuest Information and Learning
Todd Fegan, James Brei (Alt)

Random House, Inc.
Laurie Stark

Recording Industry Association of America
Linda R. Bocchi, Michael Williams (Alt)

The Research Libraries Group
Lennie Stovel, Joan Aliprand (Alt)

Serials Solutions, Inc.
Mike McCracken

SIRSI Corporation
Greg Hathorn, Slavko Manojlovich (Alt)

Society for Technical Communication (STC)
Frederick M. O’Hara, Jr., Annette D. Reilly (Alt)

Society of American Archivists
Lisa Weber

Special Libraries Association (SLA)
Marcia Lei Zeng

Synapse Corporation
Trish Yancey, Dave Clarke (Alt)

TAGSYS, Inc.
John Jordon, Anne Salado (Alt)

Talis Information Ltd
Terry Willan, Katie Anstock (Alt)

Triangle Research Libraries Network
Mona C. Couts

U.S. Department of Commerce, NIST, Office of Information Services
tba

U.S. Department of Defense, DTIC (Defense Technical Information Center)
Gopalakrishnan Nair, Jane L. Cohen (Alt)

U.S. Department of Energy, Office of Scientific & Technical Information
Ralph L. Scott, Karen J. Spence (Alt)

U.S. Government Printing Office
Judith C. Russell, T.C. Evans (Alt)

U.S. National Commission on Libraries and Information Science (NCLIS)
Robert E. Molyneux

VTLS, Inc.
Carl Grant

WebFeat
Todd Miller, Paul Duncan (Alt)


At the time NISO approved this Standard for reaffirmation in 2010, the following were members of the Z39.88-2004 Review Voting Pool:

American Library Association
Cindy Hepfer

American Psychological Association
Janice Fleming

American Society for Information Science & Technology (ASIS&T)
Marcia Zeng

Association of Information & Dissemination Centers (ASIDIC)
Drucilla Ekwurzel

Association of Research Libraries (ARL)
Julia Blixrud

Auto-Graphics, Inc.
Mary Jackson

Bibliographical Center for Research

Brenda Bailey-Hainer

Council on Library and Information Resources (CLIR)
Amy Friedlander

CrossRef
Chuck Koscher

Dow Jones Factiva
Marti Heyman

EBSCO Information Services
Oliver Pesch

EnvisionWare, Inc.
Robert Walsh

Ex Libris, Inc.
Nettie Lagace
Inera Inc.
Bruce Rosenblum

INFLIBNET Centre
Rajesh Chandrakar

International DOI Foundation
Norman Paskin

Ithaka/JSTOR/ARTstor
Evan Owens

Library of Congress
John Zagas

MINITEX Library Information Network
Cecelia Boone

Music Library Association
Mark McKnight

National Agricultural Library
Gary McCone

National Federation of Advanced Information Services (NFAIS)
Marjorie Hlava

National Library of Medicine (NLM)
Barbara Rapp

Polaris Library Systems
Eric Graham

Serials Solutions, Inc.
Harry Kaplanian

Standards Australia
LoAnne Lam

The Library Corporation (TLC)
DJ Miller

NISO Board of Directors

At the time NISO approved this Standard in 2005, the following individuals served on its Board of Directors:

Jan Peterson, Chair
Infotrieve

Carl Grant, Vice Chair and Chair-Elect
VTLS, Inc.

Beverly P. Lynch, Immediate Past Chair
UCLA Graduate School of Education & Information Studies

Michael J. Mellinger, Treasurer
Davandy, LLC

Patricia Stevens, Chair of SDC
OCLC, Inc.

Patricia R Harris, Executive Director / Secretary
NISO

Directors:

Brian Green
BIC/EDITEUR

Daniel Greenstein
California Digital Library

Jose-Marie Griffiths
University of Pittsburgh

Deborah Loeding
The H. W. Wilson Company

Richard E. Luce
Los Alamos National Laboratory

Sally McCallum
Library of Congress

Oliver Pesch
EBSCO Publishing
Committee AX Members

The following individuals served on Committee AX, The OpenURL Framework for Context-Sensitive Services that developed the 2005 standard:

- **Ann Apps**  
The University of Manchester, UK

- **Oren Beit-Arie**  
Ex Libris (USA), Inc

- **Karim Boughida**  
Getty Research Institute

- **Karen Coyle**  
University of California

- **Todd Fegan**  
ProQuest Information and Learning

- **Tony Hammond**  
Elsevier, Ltd. and (as of April 2004) Nature Publishing Group

- **Eric Hellman**  
Openly Informatics, Inc.

- **Lou Knecht**  
National Library of Medicine

- **Larry Lannom**  
Corporation for National Research Initiatives

- **Cliff Morgan** (NISO SDC Liaison)  
John Wiley & Sons, Ltd.

- **Mark Needleman**  
SIRSI Corporation

- **Eamonn Neylon**  
Manifest Solutions

- **Phil Norman**  
OCLC, Inc.

- **Oliver Pesch**  
EBSCO Publishing

- **Harry Samuels**  
Endeavor Information Systems, Inc.

- **Herbert Van de Sompel**  
Los Alamos National Laboratory

- **Eric F. Van de Velde (Chair)**  
California Institute of Technology

Acknowledgements

Standards Committee AX gratefully acknowledges the contributions made by Herbert Van de Sompel (Ghent University, Cornell University, Los Alamos National Laboratory); Patrick Hochstenbach (Ghent University); and Oren Beit-Arie (Ex Libris USA, Inc). Their original research, visionary thinking, and crucial contributions led to the formation of Committee AX and this Standard.

The Committee thanks the following individuals for their substantial assistance to the process of creating this Standard: Mary Alice Ball, University of Chicago Press; Priscilla Caplan, Florida Center for Library Automation; Young Jun Choi, KINS, Inc. (South Korea); Thom Hickey, OCLC; Marjorie Hlava, Access Innovations, Inc.; Gail M. Hodge, CENDI; Mike Hoover, ProQuest Information and Learning; Pat Stevens, OCLC; and Jeff Young, OCLC.

The Committee also thanks the following individuals for their efforts in providing the logistical support for one or more Committee meetings: Jacqueline J. Eudell, CNI; and Patti Franklin, CNRI.

The Committee thanks the following institutions for providing meeting space and logistical support for one or more Committee meetings: the California Institute of Technology, the Corporation for National Research Initiatives, and the Getty Research Institute.
The OpenURL Framework
for Context-Sensitive Services

1 Purpose and Scope

The OpenURL Framework Standard defines an architecture for creating OpenURL Framework Applications, briefly called Applications in the remainder of this Standard. An Application is a networked service environment, in which packages of information are transported over a network. These packages have a description of a referenced resource at their core, and they are transported with the intent of obtaining context-sensitive services pertaining to the referenced resource. To enable the recipients of these packages to deliver such context-sensitive services, each package describes the referenced resource itself, the network context in which the resource is referenced, and the context in which the service request takes place. These packages are ContextObject Representations.

Part 1 (Sections 5 through 11) defines the ContextObject as an abstract information construct. This Standard is independent of the application domain. It does not constrain the type of resources that may be described in a ContextObject. However, it does specify how communities can create concrete ContextObject Representations for use in their Applications. To that end, this Standard introduces the following core components of the OpenURL Framework: Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, and Namespaces.

Although ContextObject Representations may reside as autonomous data files in information systems, this Standard expects that ContextObject Representations will be transported between networked systems. Section 10 defines Transports, a core component of the OpenURL Framework in which communities specify how to transport ContextObject Representations in their Applications. This Standard does not restrict the purpose of such transportation. It is expected, however, that most transportations of ContextObject Representations will be requests for context-sensitive services pertaining to the referenced resource.

The targets of the transportation of ContextObject Representations are networked systems that are able to process ContextObject Representations and provide context-sensitive services. These systems are called Resolvers. Resolver behavior and usage are outside of the scope of this Standard. However, a community may use a Community Profile to define conformance for Resolvers that operate in its application domain. A community specifies its selections for each of the aforementioned core components in a Community Profile. This is the final core component of the OpenURL Framework, and it is defined in Section 11.

Section 6 defines the OpenURL Framework Registry and the rules that govern its usage. The OpenURL Framework Registry contains the selections for all core components made by communities that define Applications. The Registry ensures that this Standard can be used in many different application domains.

Parts 2, 3, and 4 specify the initial content of the OpenURL Framework Registry and provide detailed definitions of the registered content. The initial Registry deploys two Applications for the scholarly-information community. These Applications are defined by two Community Profiles:

- The Level 1 San Antonio Community Profile (SAP1), defined in Appendix C, is based on a Key/Encoded-Value Representation of ContextObjects. Key/Encoded-Value ContextObject Representations may be transported by any one of the three HTTP-based Transports defined in Part 4. The Transport defined in Section 22 was developed to provide a certain level of backward compatibility with the OpenURL 0.1 specification.
• The Level 2 San Antonio Community Profile (SAP2), defined in Appendix D, is based on an XML Representation of ContextObjects. XML ContextObject Representations may be transported by any one of two HTTP-based Transports defined in Part 4, Sections 20 and 21. The SAP2 Community Profile is not backward compatible with the OpenURL 0.1 specification.

Part 2 (Sections 12 through 15) defines a ContextObject Format inspired by the query string of the HTTP(S) GET request as specified in OpenURL 0.1. The Key/Encoded-Value ContextObject Format defines how to represent a ContextObject as a concatenation of ampersand-delimited Key/Encoded-Value pairs. The foremost purpose of the Key/Encoded-Value ContextObject Format is backward compatibility. It provides an elegant transition from the OpenURL 0.1 specification to this Standard.

Part 3 (Sections 16 through 19) defines a ContextObject Format based on XML (eXtensible Markup Language). XML Documents are widely used in the exchange of structured text and data between computer applications. The XML ContextObject Format is about the future. Using the full expressive power of the XML syntax, ContextObjects can be described in greater detail, which Resolvers can use to provide more and better services.

Part 4 (Sections 20 through 22) specifies mechanisms by which ContextObject Representations can be transported using the HTTP(S) protocol. Collectively, these are called OpenURL Transports.

Communities interested in deploying new Applications should use Parts 2, 3 and 4 as a guideline. Deploying a new Application consists of the following steps:

• Register any new definitions of the following core components of the OpenURL Framework that are needed to support the Application: Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Namespaces, and Transports.

• Construct a new Community Profile that defines the Application by selecting appropriate Registry entries.

• Register the Community Profile.

Communities should create Implementation Guidelines to simplify implementation and deployment of their Applications.

2 Referenced Standards

This Standard references the following existing standards:


3 Notational Conventions

The Internet Engineering Task Force (IETF) RFC 2119 [5] specifies the meaning of the following key words and key phrases: must, must not, required, shall, shall not, should, should not, recommended, may, and optional. When these appear in this Standard in a bold italic font style, they have the meaning as specified by IETF RFC 2119:

- **must**: This word, or the terms required or shall, mean that the definition is an absolute requirement of the specification.
- **must not**: This phrase, or the phrase shall not, mean that the definition is an absolute prohibition of the specification.
- **should**: This word, or the adjective recommended, mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- **should not**: This phrase, or the phrase not recommended mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- **may**: This word, or the adjective optional, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option must be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option must be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

Terms defined in this Standard will be shown in italics whenever this Standard uses them in the defined meaning, as in ContextObject or By-Value Metadata.

References to Registry Identifiers or portions of Identifiers are shown in bold, as in info:doi.

Throughout this Standard, examples are provided to support a better understanding of the key terms as they are defined. The examples are excerpts from valid Representations of ContextObjects. Many examples use an informal property-list syntax in which each property is listed on a separate line and a property consists of a key term and associated value, as in:

```
<key> = <value>
```

This property-list syntax is for illustrative purposes only. It is not part of this Standard, and it must not be used to represent ContextObjects in Applications. Parts 2 and 3 formally define two ContextObject Formats (KEV and XML). Only ContextObject Formats that are formally defined in the OpenURL Framework Registry are available for use in an Application.

Tables that specify constraints use the following short-hand notation:

<table>
<thead>
<tr>
<th>Constraint in short-hand</th>
<th>Minimum Occurrence</th>
<th>Maximum Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>≥0</td>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>≤1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>≥1</td>
<td>1</td>
<td>unbounded</td>
</tr>
</tbody>
</table>
Some Section titles end with the suffix “[Registry]”, as in “Section 7.1 Serializations [Registry]”. These Sections define core components of the OpenURL Framework, and instantiations of these core components must be registered.

This Standard uses many Identifiers based on the “info” URI scheme. On June 19th, 2003, representatives from NISO, NISO Committee AX, the IETF, and the W3C met to discuss the identification of resources in the OpenURL Framework. There was a consensus to proceed with the registration of a new top-level URI scheme. The first Internet-Draft for the “info” URI scheme was published on September 25th, 2003. A revision was published on December 5th, 2003. These drafts and any subsequent versions will be maintained on the <http://info-uri.info/> website. At the time of writing this Standard, the “info” URI scheme is awaiting approval by the Internet Engineering Steering Group (IESG) for publication as an informational RFC.

## 4 Definitions

The following terms when italicized in this Standard have the meanings indicated here:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(OpenURL Framework) Application</td>
<td>A networked service environment for the transportation of ContextObject Representations. The core characteristics of an Application are specified in a Community Profile.</td>
</tr>
<tr>
<td>By-Reference Metadata</td>
<td>A Descriptor that details properties of an Entity by the combination of: (1) a URI reference to a Metadata Format and (2) the network location of a particular instance of metadata about the Entity, the metadata being expressed according to the indicated Metadata Format.</td>
</tr>
<tr>
<td>By-Reference OpenURL Transport</td>
<td>A Transport that uses either the HTTP or the HTTPS network protocol for conveying over a network the reference to a ContextObject Representation. This reference is contained in the value associated with a single key within a query string, which is transported using either a GET or POST method.</td>
</tr>
<tr>
<td>By-Value Metadata</td>
<td>A Descriptor that specifies properties of an Entity by the combination of: (1) a URI reference to a Metadata Format; and (2) a particular instance of metadata about the Entity, expressed according to the indicated Metadata Format.</td>
</tr>
<tr>
<td>By-Value OpenURL Transport</td>
<td>A Transport that uses either the HTTP or the HTTPS network protocol for conveying over a network ContextObject Representations. The Representation is contained in the value associated with a single key within a query string, which is transported using either a GET or POST method.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Character Encoding</td>
<td>The combination of a character repertoire and an encoding form; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>Community Profile</td>
<td>The definition of an Application as a list of selections for all core components of the OpenURL Framework; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>Context</td>
<td>The network environment in which a Referent is referenced and in which a service request pertaining to the Referent takes place. In the ContextObject, the Context is expressed by five Entities: the ReferringEntity, the Requester, the ServiceType, the Resolver, and the Referrer.</td>
</tr>
<tr>
<td>ContextObject</td>
<td>An information construct that binds a description of a primary Entity — the referenced resource — together with descriptions of Entities that indicate the Context.</td>
</tr>
<tr>
<td>ContextObject Format</td>
<td>A Format to represent ContextObjects; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>ContextObject Representation</td>
<td>The Representation of a ContextObject according to a ContextObject Format.</td>
</tr>
<tr>
<td>Constraint Definition</td>
<td>A Constraint Definition specifies syntactic and semantic constraints for the representation of a given class of resources. The constraints are specified using a Constraint Language.</td>
</tr>
<tr>
<td>Constraint Language</td>
<td>A formalism used to specify syntactic and semantic restrictions on information constructs of a given class; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>Descriptor</td>
<td>A Descriptor specifies information about an Entity using one of the following four methods: Identifier, By-Reference Metadata, By-Value Metadata, or Private Data.</td>
</tr>
<tr>
<td>Entity</td>
<td>One of the six possible constituents of a ContextObject: Referent, Requester, Referrer, Resolver, ReferringEntity, or ServiceType.</td>
</tr>
<tr>
<td>Format</td>
<td>A concrete method of expression for a class of information constructs. It is a triple comprising: (1) a Serialization, (2) a Constraint Language, and (3) a Constraint Definition expressed in that Constraint Language.</td>
</tr>
<tr>
<td>Identifier</td>
<td>A Descriptor that unambiguously specifies an Entity by means of a URI.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Inline OpenURL Transport</td>
<td>A <em>Transport</em> that uses either the HTTP or the HTTPS network protocol for conveying over a network the <em>Representation</em> of one, and only one, <em>ContextObject</em>. This <em>Representation</em> consists of multiple key/value pairs within a query string, which is transported using either a GET or POST method.</td>
</tr>
<tr>
<td>KEV ContextObject Format</td>
<td>A <em>ContextObject Format</em> to represent one, and only one, <em>ContextObject</em> as a string of ampersand-delimited pairs, each pair consisting of a key and an associated value that is URL encoded.</td>
</tr>
<tr>
<td>KEV ContextObject (Representation)</td>
<td>A <em>Representation</em> of a <em>ContextObject</em> that conforms to the KEV ContextObject Format.</td>
</tr>
<tr>
<td>KEV Metadata Format</td>
<td>A <em>Metadata Format</em> to represent an <em>Entity</em> as a string of ampersand-delimited pairs, each pair consisting of a key and an associated value that is URL encoded.</td>
</tr>
<tr>
<td>KEV Metadata (Representation)</td>
<td>A <em>Representation</em> of an <em>Entity</em> that conforms to a KEV Metadata Format.</td>
</tr>
<tr>
<td>KEV Serialization</td>
<td>A method to hold in storage, or transmit over a network, the values within an information construct as a string of ampersand-delimited pairs, each pair consisting of a key and an associated value that is URL encoded.</td>
</tr>
<tr>
<td>Metadata Format</td>
<td>A <em>Format</em> to create a <em>By-Reference Metadata Descriptor</em> or a <em>By-Value Metadata Descriptor</em> of an <em>Entity</em>; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>Namespace</td>
<td>The set of all Uniform Resource Identifiers that comply with a specific URI scheme or a specific URN namespace; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>Private Data</td>
<td>A <em>Descriptor</em> that specifies information about an <em>Entity</em> using a method not defined in this Standard.</td>
</tr>
<tr>
<td>Referent</td>
<td>A resource that is referenced on a network, and about which the <em>ContextObject</em> is created; an <em>Entity</em> of the <em>ContextObject</em>.</td>
</tr>
<tr>
<td>Referrer</td>
<td>The resource that generates the <em>ContextObject</em>; an <em>Entity</em> of the <em>ContextObject</em>.</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>The resource that references the <em>Referent</em>; an <em>Entity</em> of the <em>ContextObject</em>.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><em>(OpenURL Framework)</em> Registry</td>
<td>The Registry provides a mechanism to record and publicize details of the core components of the OpenURL Framework: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles.</td>
</tr>
<tr>
<td>Registry Identifier</td>
<td>A unique name assigned on registration to specific Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles.</td>
</tr>
<tr>
<td>Representation</td>
<td>A sequence of bytes that represents a resource according to a Format.</td>
</tr>
<tr>
<td>Requester</td>
<td>The resource that requests services pertaining to the Referent; an Entity of the ContextObject.</td>
</tr>
<tr>
<td>Resolver</td>
<td>The resource at which a service request pertaining to the Referent is targeted; an Entity of the ContextObject.</td>
</tr>
<tr>
<td>Serialization</td>
<td>A method to hold in storage or transmit over a network the values within an information construct; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>ServiceType</td>
<td>The resource that defines the type of service (pertaining to the Referent) that is requested; an Entity of the ContextObject.</td>
</tr>
<tr>
<td>Transport</td>
<td>A network protocol and the method in which it is used to convey ContextObject Representations; a core component of the OpenURL Framework.</td>
</tr>
<tr>
<td>XML ContextObject Format</td>
<td>A ContextObject Format to represent one or more ContextObjects as an XML Document.</td>
</tr>
<tr>
<td>XML ContextObject (Representation)</td>
<td>A ContextObject Representation that conforms to the XML ContextObject Format.</td>
</tr>
<tr>
<td>XML Document</td>
<td>A sequence of bytes that satisfies the validity requirements of the Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation [1].</td>
</tr>
<tr>
<td>XML Metadata (Representation)</td>
<td>A Representation of an Entity that conforms to an XML Metadata Format.</td>
</tr>
<tr>
<td>XML Serialization</td>
<td>The method of using an XML Document and XML Format to represent a ContextObject.</td>
</tr>
</tbody>
</table>
The OpenURL Framework for Context-Sensitive Services

Part 1: ContextObjects and Transports

Part 1 (Sections 5 through 11) defines the core components of the OpenURL Framework: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles. Instances of these core components are preserved and made publicly available in the OpenURL Framework Registry. A community that wishes to create a new OpenURL Framework Application must create a new Community Profile. In this Community Profile, the community must specify instances for all core components, except the new Community Profile itself. If the Registry does not contain an instance of a core component needed by an Application, it is necessary to define and register an appropriate instance of the core component. The registration of a component makes this instance available for this or any other future Application. For example, if an Application needs to use a particular Namespace, that Namespace must be registered. Once registered, any Application may select this Namespace in its Community Profile.

Section 5 defines the ContextObject as an abstract information construct that consists of six Entities: Referent, ReferringEntity, Requester, ServiceType, Resolver, and Referrer. Each of these Entities is described using one or more Descriptors. There are four Descriptor types: Identifier, By-Value Metadata, By-Reference Metadata, and Private Data.

Section 6 defines the OpenURL Framework Registry and the rules that govern its usage. The Registry contains all instances of core components created by communities that deployed Applications. The Registry ensures that this Standard can be used in many different application domains.

Sections 7, 8, and 9 provide the framework for developing, defining, and registering methods to represent ContextObjects as character strings. Section 7 defines a Format as a triple consisting of a Serialization, a Constraint Language, and a Constraint Definition. Section 8 introduces ContextObject Formats and Character Encodings to represent ContextObjects. Section 9 introduces Metadata Formats and Namespaces to represent Entities.

Although ContextObject Representations may reside as autonomous data files in information systems, this Standard expects that ContextObject Representations will be transported between networked systems. Section 10 defines Transports, a core component of the OpenURL Framework. In a Transport, a community specifies how to convey over a network ContextObject Representations. This Standard does not restrict the purpose of the Transport. It is expected, however, that most Transports will be requests for context-sensitive services pertaining to the referenced resource (the Referent) and will be targeted at Resolvers.

A community specifies its selections for each of the core components in a Community Profile. This final core component of the OpenURL Framework is defined in Section 11.
5 ContextObject, Entity, and Descriptor

This Section defines the fundamental data structure of the OpenURL Framework Standard: the ContextObject. While this Standard does not restrict the use of the ContextObject to any particular environment or application, it was constructed to enable the delivery of context-sensitive services in a networked environment such as the Web.

The following scenario is used throughout this Section:

Caltech has an institutional linker server with URI http://links.caltech.edu/menu.

Jane Doe, a Caltech student with e-mail address jane.doe@caltech.edu, reads the following electronic scholarly article in the Elsevier ScienceDirect® collection:


In the reference list of that article, she comes across a reference to the following article:


In this example, Jane Doe wants services for the Bergelson article, to which she found a reference in ScienceDirect®. Jane Doe might want the full text of the article. The full text may be available from ScienceDirect® itself, an aggregator, or Caltech's interlibrary-loan department. The full-text service therefore depends on the identity and affiliation of the Jane Doe, which are part of the context of the reference to the Bergelson article. In other cases, different contextual information may be important.

The ContextObject data structure captures relevant information for the delivery of context-sensitive services pertaining to a referenced resource. Based on a study of real-world OpenURL 0.1 usage, the Committee included the following in the ContextObject data structure:

- a description of the referenced resource itself (the Bergelson article),
- a description of the resource that makes the reference (the McArthur article), and
- a description of four other resources that are useful in fulfilling service requests pertaining to the referenced resource:
  1. the agent requesting the service (Jane Doe),
  2. the type of service that is requested (full text),
  3. the system at which the service request is targeted (Caltech linking server), and
  4. the system where the service request originates (ScienceDirect®).

The formal definition of ContextObject follows.

5.1 ContextObject and Entity

A ContextObject is a data structure that binds together descriptions of:

- A Referent: A resource that is referenced on a network and about which the ContextObject is created
- A ReferringEntity: The resource that references the Referent
- A Requester: The resource that requests services pertaining to the Referent
- A **ServiceType**: The resource that defines the type of service (pertaining to the *Referent*) that is requested
- A **Resolver**: The resource at which a service request pertaining to the *Referent* is targeted
- A **Referrer**: The resource that generates the ContextObject

The ContextObject is created to enable the delivery of services pertaining to the *Referent*, which is at the core of the ContextObject. The descriptions of the ReferringEntity, the Requester, the ServiceType, the Resolver, and the Referrer express the Context in which the *Referent* is referenced and in which the request for services pertaining to the *Referent* takes place.

The remainder of this Standard uses the term *Entity* to refer to any of the six types of resources that *may* be described in a ContextObject.

Example 1 uses the scenario introduced above to illustrate all *Entities* of the ContextObject.

### Example 1: Examples of Entities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referent</td>
<td>The scholarly article by Bergelson</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>The scholarly article by McArthur</td>
</tr>
<tr>
<td>Requester</td>
<td>Jane Doe</td>
</tr>
<tr>
<td>ServiceType</td>
<td>Full text of the Bergelson article</td>
</tr>
<tr>
<td>Resolver</td>
<td>The Caltech linking server</td>
</tr>
<tr>
<td>Referrer</td>
<td>Elsevier’s ScienceDirect®</td>
</tr>
</tbody>
</table>

### 5.2 Descriptor

A **Descriptor** specifies information about an *Entity*. This Section defines the four types of **Descriptors** that are available in this Standard: **Identifier**, **By-Value Metadata**, **By-Reference Metadata**, and **Private Data**.

#### 5.2.1 Identifier

An **Identifier Descriptor** unambiguously specifies the *Entity* by means of a Uniform Resource Identifier (URI). This URI either points to the *Entity* itself or to metadata that specify the *Entity*.

### Example 2: Identifiers for a Referent, Requester, and Resolver

```
rft_id = info:doi/10.1126/science.275.5304.1320
rft_id = info:pmid/9036860
req_id = mailto:jane.doe@caltech.edu
res_id = http://links.caltech.edu/menu
```

Example 2 shows Identifier Descriptors for a Referent (the Bergelson article), a Requester (Jane Doe), and a Resolver (the Caltech linking server) using the informal property-list syntax of Section 3. The key names (*rft_id*, for example) resemble those introduced in Part 2 of this Standard. However, Part 1 of this Standard uses these names for illustration only and does not formally define them.

The Digital Object Identifier (DOI) **10.1126/science.275.5304.1320** identifies the Bergelson article. As such, the URI **info:doi/10.1126/science.275.5304.1320** is an Identifier Descriptor for the Referent. The PubMed identifier **9036860** identifies metadata for the Bergelson article. Therefore, the URI **info:pmid/9036860** is also an Identifier Descriptor for the Referent.
The e-mail address jane.doe@caltech.edu describes Jane Doe. The corresponding URI mailto:jane.doe@caltech.edu is an Identifier Descriptor for this Requester.

The URI http://links.caltech.edu/menu describes the institutional linking server at Caltech and is an Identifier Descriptor for this Resolver.

5.2.2 By-Value Metadata

A By-Value Metadata Descriptor specifies properties of the Entity by the combination of: (1) a URI reference to a Metadata Format; and (2) a particular instance of metadata about the Entity expressed according to this Metadata Format.

Example 3: By-Value Metadata for a Referent

```plaintext
rft_val_fmt = info:ofi/fmt:kev:mtx:journal
rft.aulast = Bergelson
rft.auinit = J
rft.date = 1997
rft.atitle = Isolation of a common receptor for coxsackie B viruses and adenoviruses 2 and 5
rft.jtitle = Science
rft.volume = 275
rft.spage = 1320
rft.epage = 1323
```

Example 3 shows a By-Value Metadata Descriptor for a Referent, the Bergelson article. The URI specified as the value of the rft_val_fmt key (info:ofi/fmt:kev:mtx:journal) identifies the Metadata Format used to describe the Bergelson article. Sections 6, 7, and 8 explain how to construct and interpret Registry Identifiers for Formats, such as info:ofi/fmt:kev:mtx:journal. This particular Registry Identifier identifies a Metadata Format for a journal article.

The remaining lines in Example 3 are metadata properties for the Referent. The metadata keys (aulast, auinit, date, etc.) are from the identified Metadata Format for a journal article. The metadata keys are prefixed with rft. to indicate that the metadata describe the Referent.

5.2.3 By-Reference Metadata

A By-Reference Metadata Descriptor specifies properties of the Entity by the combination of: (1) a URI reference to a Metadata Format; and (2) the network location — specified by means of a URI — of a particular instance of metadata about the Entity expressed according to this Metadata Format.

Example 4: By-Reference Metadata for a Requester

```plaintext
req_ref_fmt = http://lib.caltech.edu/mxt/ldap.html
req_ref = ldap://ldap.caltech.edu:389/janed
```

Example 4 shows a By-Reference Metadata Descriptor for a Requester, Jane Doe. The value associated with the req_ref key is a pointer to (or network location of) Jane Doe’s entry in the Caltech LDAP directory server. The value of the req_ref_fmt key specifies the Metadata Format of the document to which the value of the req_ref key points.

5.2.4 Private Data

A Private Data Descriptor specifies information about the Entity using a method not defined in this Standard. This Standard does not provide any global mechanisms to interpret Private Data. Instead, it is assumed that the Resolver and the Referrer have a common understanding, based on a tacit or
explicit bilateral agreement. To make it possible for the Resolver to interpret Private Data, a ContextObject that contains Private Data should identify the Referrer that created it.

Example 5: Private Data for a Referent

\[
\begin{align*}
\text{rft_dat} & = \text{cites/8///citedby/12} \\
\text{rfr_id} & = \text{info:sid/elsevier.com:ScienceDirect}
\end{align*}
\]

Example 5 shows a Private Data Descriptor for a Referent. The value associated with the rft_dat key, cites/8///citedby/12, is Private Data provided about the Referent. The value associated with the rfr_id key, info:sid/elsevier.com:ScienceDirect, is an Identifier Descriptor of the Referrer. Knowing the identity of the Referrer might help the Resolver to interpret the Private Data.

### 5.3 Constraints

The number of occurrences of each Entity that may be present in a ContextObject is constrained:

- A ContextObject must contain exactly one Referent.
- A ContextObject may contain at most one ReferringEntity, Requester, and Referrer.
- A ContextObject may contain zero or more ServiceTypes and Resolvers.

These fundamental constraints are summarized in the first two columns of Table 1.

The remaining columns of Table 1 indicate that:

- All four Descriptors may be used to describe each of the Entities.
- Each type of Descriptor may be used zero or more times for the description of a specific Entity. No ordering or priority is defined for multiple Descriptors.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Number</th>
<th>Descriptors</th>
<th>Identifier</th>
<th>By-Value Metadata</th>
<th>By-Reference Metadata</th>
<th>Private Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referent</td>
<td>1</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>≤ 1</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Requester</td>
<td>≤ 1</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ServiceType</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Resolver</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Referrer</td>
<td>≤ 1</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
</tbody>
</table>

ContextObject Formats that define specific methods to represent ContextObjects (see Section 8.2) must not relax the constraints expressed in Table 1, but they may restrict them:

- A ContextObject Format may limit the number of occurrences of ServiceType and Resolver Entities.
- A ContextObject Format may limit the number of Descriptors that may be used to describe each Entity.
- A ContextObject Format must not allow multiple Referent, ReferringEntity, Requester, or Referrer Entities.
Any additional constraints must be specified in ContextObject Format definitions.

When one Entity of a ContextObject is described by multiple Descriptors, those Descriptors must describe the same resource. For example, when a ContextObject contains two Identifier Descriptors and one By-Value Metadata Descriptor for one Referent Entity, all three Descriptors must describe the same Referent.

When there are multiple occurrences of the same Entity in one ContextObject, each occurrence must represent a different resource. These multiple occurrences must not be interpreted as variant descriptions of the same resource. For example, two Resolver Entities contained in one ContextObject must represent two distinct Resolvers. In this case, each Resolver Entity may have multiple Descriptors, each of which must be a variant description of the same Resolver.

ContextObject Formats that allow multiple occurrences of ServiceType and/or Resolver Entities must define how multiple Descriptors are grouped to bind to particular Entities.

The basic data model for ContextObjects does not constrain the number of ContextObjects that may be represented in an instance document that conforms to the ContextObject Format. ContextObject Formats may constrain this number, and each Community Profile provides this information (see Sections 11, 15, and 19).

6 Registry

This Section defines the OpenURL Framework Registry, referred to as the Registry in the remainder of this Standard. It is based at <http://www.openurl.info/registry>.

Upon approval of this Standard, NISO will establish one or more Maintenance Agencies for the Registry. The responsibilities and duties for Maintenance Agencies of the OpenURL Framework Standard are specified in Appendix A.

6.1 Registry Entries

When a community defines an Application, it must specify selections for each of the core components of the OpenURL Framework:

- For representing ContextObjects:
  - Character Encodings (one or more)
  - Serializations (one)
  - Constraint Languages (one)
  - ContextObject Formats (one)

- For representing Entities of ContextObjects:
  - Namespaces (zero or more)
  - Metadata Formats (zero or more)

- For transporting ContextObject Representations:
  - Transports (one or more)

- For defining Applications:
  - Community Profiles (one)
Figure 1 illustrates the structure of the Registry and shows how Community Profiles define the characteristics of an Application by listing community-specific selections for the core components of the OpenURL Framework.

Example 6: A Registry Entry

<table>
<thead>
<tr>
<th>info:ofi/nam:info:doi:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dc:title</td>
<td>Namespace for Digital Object Identifiers</td>
</tr>
<tr>
<td>dc:creator</td>
<td>International DOI Foundation</td>
</tr>
<tr>
<td>dc:date</td>
<td>2004-01-01</td>
</tr>
<tr>
<td>dc:identifier</td>
<td>info:doi/</td>
</tr>
</tbody>
</table>

Example 6 shows a Registry entry that describes an instance of a Namespace, a core component of the OpenURL Framework introduced in Section 9.1. This entry describes the Namespace of Digital Object Identifiers (DOI), which is introduced in Appendix C. The top row displays the Registry Identifier of the Registry entry, and remaining rows use Dublin Core metadata [18] to describe the Registry entry (see Sections 6.2 and 6.3).

Part 1 is concerned with the OpenURL Framework; it does not register specific instances of core components. In Parts 2, 3, and 4 and Appendices Appendix B, Appendix C, and Appendix D, this Standard defines, registers, and uses specific instances of core components. However, Registry entries in this Standard are provided for illustrative purposes only and are often only partially displayed. For example, the dc:date field is usually omitted as it is 2004-01-01 for all entries in the
6.2 Registry Identifiers

Upon registration, each instance of a core component receives a unique Registry Identifier, which is a URI of the form info:ofi/char-string, where:

- info is the name of the URI scheme;
- ofi represents the namespace under the info scheme reserved for Registry Identifiers; and
- char-string must be replaced by a unique character string assigned by the Registry upon registration of the instance of the core component.

Table 2 summarizes where to find information related to Registry Identifiers for core components. The first column lists core components. The second column displays the structure of their Registry Identifiers. The third column lists which Section defines each core component. The fourth and fifth columns (with KEV and XML heading, respectively) give the Sections and Appendices where instances of core components are introduced. This Standard initializes the Registry with entries that bootstrap two Applications for the scholarly-information community: one Application based on the KEV ContextObject Format and one based on the XML ContextObject Format. These entries may also be used by other communities.

Table 2: Core Components and their Registry Identifiers

<table>
<thead>
<tr>
<th>Core Component</th>
<th>Registry Identifier Structure</th>
<th>Framework</th>
<th>KEV</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serializations</td>
<td>info:ofi/fmt:_</td>
<td>7.1</td>
<td>12.1</td>
<td>16.1</td>
</tr>
<tr>
<td>Constraint Languages</td>
<td>info:ofi/fmt:<em>:</em></td>
<td>7.2</td>
<td>12.2</td>
<td>16.2</td>
</tr>
<tr>
<td>Character Encodings</td>
<td>info:ofi/enc:_</td>
<td>8.1</td>
<td>13.3</td>
<td>17.4</td>
</tr>
<tr>
<td>ContextObject Formats</td>
<td>info:ofi/fmt:<em>:</em>:ctx(^{(1)})</td>
<td>8.2</td>
<td>12.3.1</td>
<td>17.3</td>
</tr>
<tr>
<td>Namespaces</td>
<td>info:ofi/nam:_</td>
<td>9.1</td>
<td>C.5</td>
<td>D.5</td>
</tr>
<tr>
<td>Metadata Formats</td>
<td>info:ofi/fmt:<em>:</em>:(^{(2)})</td>
<td>9.2</td>
<td>14.2</td>
<td>18.2</td>
</tr>
<tr>
<td>Transports</td>
<td>info:ofi/tsp:_</td>
<td>10</td>
<td>20, 21</td>
<td>22</td>
</tr>
<tr>
<td>Community Profiles</td>
<td>info:ofi/pro:_</td>
<td>11</td>
<td>15, Appendix C</td>
<td>19, Appendix D</td>
</tr>
</tbody>
</table>

\(^{(1)}\) The last component of Registry Identifiers for ContextObject Formats must start with the reserved prefix ctx.

\(^{(2)}\) The last component of Registry Identifiers for Metadata Formats must not be named with the reserved prefix ctx.

6.3 Using the Registry

Given the Registry Identifier of a Registry entry, it is possible to obtain the Dublin Core metadata description and the actual definition of the entry. (In the URIs shown below, replace the bold and underlined keyword registry-identifier with the Registry Identifier of the Registry entry.)

The Dublin Core metadata description [18] of the entry is available in two forms:

- for display in a web browser: &lt;http://www.openurl.info/registry/dc/registry-identifier&gt;
- for direct access: &lt;http://www.openurl.info/registry/docs/dc/registry-identifier&gt;

The Dublin Core metadata may include dc:identifier fields, each containing a URI that points to a definition of the Registry entry. This mechanism provides access to multiple forms of the definition.

The following standard form URI always resolves to a definition of the registered resource:
If there are one or more `dc:identifier` fields in the `Registry` entry, the standard form URI resolves to the URI contained in the first `dc:identifier` field. There is no standard form URI available to access definitions pointed to by URIs in subsequent `dc:identifier` fields.

In Example 6, the `Registry Identifier` of the DOI Namespace is `info:ofi/nam:info:doi:`, and its Dublin Core metadata description is available in two forms:


The standard form URI `<http://www.openurl.info/registry/docs/info:ofi/nam:info:doi:>refers to the definition of the registered resource itself (the DOI Namespace), which is described by the Dublin Core metadata shown above.

Because there are one or more `dc:identifier` fields in this `Registry` entry, a resolution mechanism redirects the standard form URI to the URI in the first `dc:identifier` field. This URI, `<http://errol.oclc.org/info-uri.info/info:doi/?metadataPrefix=reg>`, resolves to a browser display of the definition of the DOI Namespace under the “info” URI scheme.

The URI in the second `dc:identifier` field, `<http://errol.oclc.org/info-uri.info/info:doi/.reg>`, points to the raw XML record defining the DOI Namespace under the “info” URI scheme (as opposed to the HTML rendition of this record).

The URI in the third `dc:identifier` field points to the info:doi/ namespace in the “info” URI scheme.

In the initial `Registry`, URIs following the pattern `<http://www.openurl.info/registry/docs/registry-identifier>` are reserved for "native forms", while the rest of the `Registry` is suitable for web browsing. The initial `Registry` contains the following entry types stored inside the `Registry`:

- Items described by Dublin Core metadata formatted for web browsing use the URI pattern `<http://www.openurl.info/registry/dc/registry-identifier>`. These same items in their native form use the URI pattern `<http://www.openurl.info/registry/docs/dc/registry-identifier>`. In this case, the native form is an XML Document that conforms to the XML Schema located at `<http://www.openarchives.org/OAI/2.0/oai_dc.xsd>`.
- In Section 11, an XML Schema to define Community Profiles will be introduced. Web-browsable definitions of Community Profiles will use the URI pattern `<http://www.openurl.info/registry/pro/registry-identifier>`, and XML-based definitions will use the URI pattern `<http://www.openurl.info/registry/docs/pro/registry-identifier>`.

The initial `Registry` supports the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [17] as a machine interface for downloading `Registry` materials.

7 Formats

To enable the use of a wide variety of Representations of ContextObjects and their Entities, this Standard defines the notion of a Format.
A Format is a method to represent information constructs as character strings.

Each Format consists of a Serialization, a Constraint Language, and a Constraint Definition expressed using the Constraint Language. In this Standard, the set of three items defining a Format is called a triple and is represented by a short-hand notation as in:

\[
\{ \text{Serialization, Constraint Language, Constraint Definition} \}
\]

In Section 8.2, the Format notion is used to define ContextObject Format, which gives communities the ability to define, register, and use ContextObject Representations that are the most appropriate for their application domain. In Section 9.2, the Format notion is used to define Metadata Format, which gives communities the ability to define, register, and use appropriate methods to describe Entities of ContextObjects by means of By-Value or By-Reference Metadata Descriptors.

### 7.1 Serializations [Registry]

For representing ContextObjects and their Entities, this Standard supports the use of a variety of Serializations.

A Serialization is a method by which structured information can be held in storage and/or can be transmitted over a network.

The description of a resource, such as a ContextObject or an Entity, is often a hierarchical and complex structure at the conceptual level. The form in which it is stored and/or transmitted over a network, however, is a simple character string. An example of such a storage and/or transmission form is XML.

Serializations must be registered before use in an Application.

Communities may use Serializations that are already in the Registry, or they may register additional Serializations as needed.

Upon registration, a Serialization is assigned a Registry Identifier, formed by concatenating three character strings:

- **info:ofi/**, which represents the namespace under the info scheme reserved for Registry Identifiers
- **fmt:**, the character string used to introduce Format-related Identifiers in the info:ofi/ Namespace
- a character string that is assigned on registration and identifies the Serialization.

Registry Identifiers of Serializations are used to support Registry management and to identify Serializations in Community Profiles. In typical use, Registry Identifiers of Serializations do not show up in Representations of ContextObjects or their Entities.

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Format-related</th>
<th>Serialization</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>info:ofi/</td>
<td>fmt:</td>
<td>xml</td>
<td>info:ofi/fmt:xml</td>
</tr>
</tbody>
</table>

Table 3 illustrates the construction of Registry Identifiers for the two Serializations in the initial Registry:

- KEV: A resource is represented as a string of ampersand-delimited pairs, each pair consisting of a key and an associated URL-encoded value. In the remainder of this Standard, Key/Encoded-Value is abbreviated as KEV. (See Section 12.1.)
7.2 Constraint Languages [Registry]

For expressing syntactic and semantic constraints on the representation of ContextObjects and their Entities, this Standard supports the use of a variety of Constraint Languages.

A Constraint Language is a method to specify syntactic and semantic restrictions on information constructs of a given class that are to be serialized. Each Constraint Language is tied to one Serialization.

Constraint Languages must be registered before use in an Application.

Communities may use Constraint Languages that are already in the Registry, or they may register additional Constraint Languages as needed.

Upon registration, a Constraint Language is assigned a Registry Identifier, formed by concatenating four character strings:

- `info:ofi/`, which represents the namespace under the `info` scheme reserved for Registry Identifiers
- `fmt:`, the character string to introduce Format-related Identifiers in the `info:ofi/` Namespace
- a character string that identifies the registered Serialization to which the Constraint Language is tied followed by a colon character (`:`)
- a character string that is assigned on registration and identifies the Constraint Language.

Registry Identifiers of Constraint Languages are used to support Registry management and to identify Constraint Languages in Community Profiles. In typical use, Registry Identifiers of Constraint Languages do not show up in Representations of ContextObjects or their Entities.

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Format-related</th>
<th>Serialization</th>
<th>Constraint Language</th>
<th>Registry Identifier</th>
</tr>
</thead>
</table>

Table 4 illustrates the construction of Registry Identifiers for the two Constraint Languages in the initial Registry (see Sections 12.2 and 16.2):

- Z39.88-2004 Matrix: This Constraint Language, which is defined in Appendix B, defines how to construct a matrix that specifies how to describe a specific class of resources using a string of ampersand-delimited KEV pairs. For an example of a matrix that defines a KEV Metadata Format, see Section 12.3.2.
- XML Schema: The W3C XML Schema definition language is endorsed by the World Wide Web Consortium (W3C) to describe the structure and constrain the contents of XML 1.0 documents [3] [4]. For an example of the use of an XML Schema that defines an XML Metadata Format, see Section 16.3.2.

7.3 Constraint Definitions

A Constraint Definition specifies syntactic and semantic constraints for the Representation of a given class of resources. The constraints are specified using a Constraint Language.
This Standard uses two types of Constraint Definitions. Section 8.2 uses one type to constrain Representations of ContextObjects, leading to ContextObject Formats. Section 9.2 uses another type to constrain Representations of Entities of ContextObjects, leading to Metadata Formats, which enable By-Value and By-Reference Metadata Descriptors.

Section 12.3 shows how to use the Z39.88-2004 Matrix Constraint Language to define constraints on KEV Serializations. Tables Table 13 and Table 14 show excerpts of Constraint Definitions of, respectively, the KEV ContextObject Format and the KEV Metadata Format for items of the type “book”.

Section 16.3 shows how to use the XML Schema Constraint Language to define constraints on XML Serializations. Tables Table 18 and Table 19 show XML Schema Constraint Definitions of, respectively, the XML ContextObject Format and the XML Metadata Format for items of the type “journal”.

8 Representing ContextObjects

This Section defines two core components of the OpenURL Framework that are essential for the Representation of ContextObjects: Character Encodings and ContextObject Formats.

8.1 Character Encodings [Registry]

For the Representation of ContextObjects, this Standard supports the use of a variety of Character Repertoires and Encoding Forms as defined in Character Encoding Model [13].

A Character Encoding is a combination of a Character Repertoire and an Encoding Form.

All Character Encodings used in Applications must be taken from the Internet Assigned Naming Authority (IANA) List of Registered Character Sets [7]. When a ContextObject Representation declares that it is using a specific Character Encoding, it must follow the specification of the corresponding IANA character set, as shown in the IANA list.

All Character Encodings must be registered before use in an Application.

Communities may use Character Encodings that are already in the Registry, or they may register additional Character Encodings from the IANA list as needed.

Upon registration, a Character Encoding is assigned a Registry Identifier, formed by concatenating three character strings:

- **info:ofi/**, which represents the namespace under the info scheme reserved for Registry Identifiers
- **enc:**, a character string that uniquely identifies a core component of the OpenURL Framework, which for Character Encodings must be enc:
- a character string that identifies the IANA character set [7]. Use the character string in the official IANA definition that is tagged as
  - "preferred MIME name", if available, or
  - “Name”, if a preferred MIME name is not available.

Registry Identifiers of Character Encodings are typically used to declare the Character Encoding of a ContextObject Representation. They are also used to support Registry management and to identify Character Encodings in Community Profiles.
Table 5: Registry Identifiers for Character Encodings

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Core Component</th>
<th>IANA preferred MIME name</th>
<th>IANA Name</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>info:ofi/enc:</td>
<td>UTF-8</td>
<td>info:ofi/enc:UTF-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>info:ofi/enc:Big5</td>
<td>Big5</td>
<td>Big5</td>
<td></td>
<td>info:ofi/enc:Big5</td>
</tr>
</tbody>
</table>

Table 5 shows how to construct Registry Identifiers for the Character Encodings in the initial Registry: ISO 8859-1 (ISO Latin 1), UTF-8 encoded Unicode, and Big5.

Example 7: Identification of a Character Encoding

```
ctx_enc = info:ofi/enc:UTF-8
```

Example 7 shows how a KEV ContextObject Representation specifies its Character Encoding by assigning the Registry Identifier of the Character Encoding to the `ctx_enc` administrative key.

8.2 ContextObject Formats [Registry]

A ContextObject Format is a specification of concrete selections for all three items of the Format triple `{ Serialization, Constraint Language, Constraint Definition } for the purpose of representing ContextObjects.

ContextObject Formats must not relax the constraints expressed in Table 1, but they may restrict them:

- A ContextObject Format may limit the number of occurrences of ServiceType and Resolver Entities.
- A ContextObject Format may limit the number of Descriptors that may be used to describe each Entity.
- A ContextObject Format must not allow multiple Referent, ReferringEntity, Requester, or Referrer Entities.

Any additional constraints must be specified in ContextObject Format definitions.

The basic data model for ContextObjects does not constrain the number of ContextObjects that may be represented in an instance document that conforms to the ContextObject Format. ContextObject Formats may constrain this number, and each Community Profile provides this information. (See Sections 11, 15, and 19.)

It is recommended that a ContextObject Format provide the capability to convey administrative information. The Registry may require providing this capability.

ContextObject Formats must be registered before use in an Application.

Communities may use ContextObject Formats that are already in the Registry, or they may register additional ContextObject Formats as needed.

Upon registration, a ContextObject Format is assigned a Registry Identifier, formed by concatenating three character strings:

- `info:ofi/`, which represents the namespace under the `info` scheme reserved for Registry Identifiers
• fmt: a character string used to introduce Format-related Identifiers in the info:ofi/ Namespace

• a character string that identifies the Format triple for the ContextObject Format, consisting of:
  – a character string that identifies the registered Serialization followed by a colon character ('\n')
  – a character string that identifies the registered Constraint Language followed by a colon character ('\n')
  – a character string that is assigned on registration and identifies the Constraint Definition. It must start with the reserved prefix ctx to indicate that this is a ContextObject Format.

Registry Identifiers of ContextObject Formats are used in ContextObject Representations to specify the Format by which the ContextObjects are represented. Registry Identifiers of ContextObject Formats are also used to support Registry management and to identify ContextObject Formats in Community Profiles.

Table 6: Registry Identifiers for ContextObject Formats

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Format-related</th>
<th>Serialization</th>
<th>Constraint Language</th>
<th>Constraint Definition</th>
<th>Registry Identifier</th>
</tr>
</thead>
</table>

Table 6 illustrates the construction of Registry Identifiers for the two ContextObject Formats in the initial Registry (described in Sections 12 and 16, respectively):

• The KEV ContextObject Format represented by the triple: { KEV, Z39.88-2004 Matrix, matrix of Table 13 }.
  – The KEV ContextObject Format restricts the number of ServiceType and Resolver Entities to “≤1” and the number of Referrer Entities to exactly one. (See Table 15.)
  – The KEV ContextObject Format includes the capability to convey administrative information; see Table 17.

• The XML ContextObject Format represented by the triple: { XML, XML Schema, XML Schema of Section 16.2 }.
  – The XML ContextObject Format restricts the number of Referrer Entities to exactly one; see Table 20.
  – The XML ContextObject Format includes the capability to convey administrative information; see Table 22.
  – To support new applications, communities could introduce new XML-based ContextObject Formats constrained by other syntactic constraint languages (DTD or RELAX NG, for example) or semantic constraint languages (RDFS or OWL, for example).
Example 8: Identification of a ContextObject Format

url_ctx_fmt = info:ofi/fmt:kev:mtx:ctx

Example 8 shows how a Registry Identifier of a ContextObject Format is used as the value of an url_ctx_fmt key of an OpenURL Transport (see Part 4) to specify the Format of the transported ContextObject Representation.

9 Representing Entities

This Section defines core components of the OpenURL Framework that are essential for the representing Entities of ContextObjects:

- Namespaces for describing Entities with Identifier Descriptors
- Metadata Formats for describing Entities with By-Value and/or By-Reference Metadata Descriptors

Entities may also be described by Private Data Descriptors. Because the nature of Private Data is not specified by this Standard, there is no infrastructure in the OpenURL Framework to support Private Data: none of the core components explicitly deal with Private Data, and Community Profiles do not contain any information to facilitate the use of Private Data in Applications.

The Metadata Format used to represent an Entity must be compatible with the ContextObject Format used to represent the ContextObject that contains the Entity. In most cases, the Metadata Format and the ContextObject Format must be based on the same Serialization and Constraint Language. This requirement is waived only if the Entity is described with a By-Reference Metadata Descriptor and the Metadata Format is registered.

In most cases, a Character Encoding used for the Representation of an Entity and the Character Encoding used for the Representation of the ContextObject that contains the Entity must be identical. This requirement is waived only for By-Reference Metadata, provided that it contains a standards-based declaration of its Character Encoding. In this case, the Character Encoding of the By-Reference Metadata may differ from that of the ContextObject. However, this is strongly discouraged, because it is not guaranteed that Resolvers will be able to process in a meaningful way the Character Encoding specified in the By-Reference Metadata.

9.1 Namespaces [Registry]

Identifier Descriptors describe Entities with Identifiers. This Standard provides for the use of Identifiers from a wide variety of namespaces. This Section defines which Identifiers are valid according to this Standard.

All Identifiers used in OpenURL Framework Applications must be Uniform Resource Identifiers (URIs) or Uniform Resource Names (URNs). URI schemes and URN namespaces are maintained by the Internet Assigned Numbers Authority (IANA) and are available at:

- IANA URI Schemes Registry [8]: <http://www.iana.org/assignments/uri-schemes>
- IANA URN Namespace Identifiers Registry [9]: <http://www.iana.org/assignments/urn-namespaces>

URI schemes or URN namespaces must be registered before use in an Application.

Registered URI schemes and URN namespaces are called Namespaces. This Standard does not support the use of unregistered Namespaces. Only Identifiers that belong to a registered Namespace may be used in Identifier Descriptors of Entities.
Communities may use Namespaces that are already in the Registry, or they may register additional Namespaces as needed.

Upon registration, a Namespace is assigned a Registry Identifier, formed by concatenating three character strings:

- `info:ofi/`, which represents the namespace under the info scheme reserved for Registry Identifiers
- `nam:`, a character string that uniquely identifies a core component of the OpenURL Framework, which for Namespaces must be `nam:`
- a character string indicating the actual URI scheme or URN namespace:
  - For URI schemes, use the string listed under the Column “Scheme Name” of the IANA URI Schemes registry [8].
  - For URN namespaces, use the character string `urn:` followed by the string listed under the Column “Registered Formal URN Namespaces” of the IANA URN Namespaces registry [9].

Registry Identifiers of Namespaces are used primarily to support Registry management and to identify Namespaces in Community Profiles. In typical use, Registry Identifiers of Namespaces do not show up in Representations of ContextObjects or their Entities.

Table 7: Registry Identifiers for Namespaces

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Core Component</th>
<th>URI Scheme or URN Namespace</th>
<th>Registry Identifier</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>info:ofi/</td>
<td>nam:</td>
<td>mailto</td>
<td>info:ofi/nam:mailto</td>
<td>mailto URI Scheme (RFC 2368 [15])</td>
</tr>
<tr>
<td>info:ofi/</td>
<td>nam:</td>
<td>urn:ISSN</td>
<td>info:ofi/nam:urn:ISSN</td>
<td>ISSN URN Namespace (RFC 3044 [16])</td>
</tr>
</tbody>
</table>

Table 7 illustrates the construction of Registry Identifiers for three Namespaces. For a list of all Namespaces in the initial Registry, see Sections C.5 and D.5.

Example 9: Identification of Entities using Identifiers from Namespaces

```
   rft_id = urn:ISBN:0262011808
   rft_id = info:pmid/9036860
   req_id = mailto:jane.doe@caltech.edu
   rfr_id = http://www.sciencedirect.com
```

Example 9 shows Identifiers used as Descriptors of two Referents, a Requester, and a Referrer. The Identifier `mailto:jane.doe@caltech.edu` belongs to the Namespace for the “mailto” URI Scheme, which was assigned the Registry Identifier `info:ofi/nam:mailto` upon registration.

9.2 Metadata Formats [Registry]

A Metadata Format is a specification of concrete selections for all three items of the Format triple `{ Serialization, Constraint Language, Constraint Definition }` for the purpose of representing an Entity.
Metadata Formats define Formats that can be used for the Representation of Entities of ContextObjects by means of By-Value and/or By-Reference Metadata Descriptors.

Metadata Formats used in Applications should be registered. This Standard recommends the use of registered Metadata Formats when feasible, but it supports the use of unregistered Metadata Formats that meet the requirements described below.

A Metadata Format used to represent an Entity must be compatible with the ContextObject Format used to represent the ContextObject that contains the Entity. In most cases, the Metadata Format and the ContextObject Format must be based on the same Serialization and Constraint Language. This requirement is waived only if the Entity is described with a By-Reference Metadata Descriptor and the underlying Metadata Format is registered.

Communities may use Metadata Formats that are already in the Registry, they may register additional Metadata Formats, or they may use unregistered Metadata Formats.

Upon registration, a registered Metadata Format is assigned a Registry Identifier, formed by concatenating three character strings:

- \texttt{info:ofi/}, which represents the namespace under the \texttt{info} scheme reserved for Registry Identifiers
- \texttt{fmt:}, a character string used to introduce Format-related Identifiers in the \texttt{info:ofi/} Namespace
- a character string that identifies the Format triple. It consists of:
  - a character string that identifies the registered Serialization followed by a colon character (\textquoteleft:\textquoteright)
  - a character string that identifies the registered Constraint Language followed by a colon character (\textquoteleft:\textquoteright)
  - a character string that is assigned on registration and associates a name with the Metadata Format. This name must not start with the prefix \texttt{ctx}, which is reserved for ContextObject Formats. The name of the Metadata Format may be non-unique in the Registry but the Registry Identifier must be unique to the Registry. No relationship is assumed between Metadata Formats with the same or similar names.

Registry Identifiers of registered Metadata Formats are used in ContextObject Representations to specify the Format by which the Entities in the ContextObjects are represented. Registry Identifiers of Metadata Formats are also used to support Registry management and to identify registered Metadata Formats in Community Profiles.

An unregistered Metadata Format must be identified by means of a URI. This URI

- must not reside under the \texttt{info:ofi/} namespace of the \texttt{info} URI scheme; and
- must be network-dereferenceable and point to a document that contains the constraint definition of the Metadata Format.

Identification of unregistered Metadata Formats is used in ContextObject Representations to specify the Format by which the Entities in the ContextObjects are represented.

\textbf{Table 8: Registry Identifiers for Registered Metadata Formats}

<table>
<thead>
<tr>
<th>&quot;info&quot; URI Namespace</th>
<th>Core Component</th>
<th>Serialization</th>
<th>Constraint Language</th>
<th>Constraint Definition</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{info:ofi/}</td>
<td>\texttt{fmt:}</td>
<td>\texttt{kev:}</td>
<td>\texttt{mtx:}</td>
<td>\texttt{book}</td>
<td>\texttt{info:ofi/fmt:kev:mtx:book}</td>
</tr>
<tr>
<td>\texttt{info:ofi/}</td>
<td>\texttt{fmt:}</td>
<td>\texttt{xm:}</td>
<td>\texttt{xsd:}</td>
<td>\texttt{patent}</td>
<td>\texttt{info:ofi/fmt:xm:xsd:patent}</td>
</tr>
</tbody>
</table>
Table 8 illustrates the construction of Registry Identifiers for two registered Metadata Formats:

- KEV Metadata Format for items of the type "book", specified by the Format triple { KEV, Z39.88-2004 Matrix, book }
- XML Metadata Format for items of the type "patent", specified by the Format triple {XML, XSD, patent }

(For a list of all Metadata Formats in the initial Registry, see Table 26 of Appendix C for KEV Metadata Formats and Table 28 of Appendix D for XML Metadata Formats.)

Example 10: Identification of Unregistered Metadata Formats

```
rft_val_fmt = http://www.example.net/mtx/cars.html
```

Example 10 shows the identification of an unregistered Metadata Format for By-Value Metadata of a Referent in a KEV ContextObject Representation. The file cars.html is a Constraint Definition.

10 Transporting ContextObject Representations: Transports [Registry]

A Transport is a method by which a ContextObject Representation may be transported over a network. A Transport is the combination of a network protocol and a method by which this network protocol transports a ContextObject Representation.

Transports must be registered before use in an Application.

Communities may use Transports that are already in the Registry, or they may register additional Transports as needed.

Upon registration, a Transport is assigned a Registry Identifier, formed by concatenating four character strings:

- info:ofi/, which represents the namespace under the info scheme reserved for Registry Identifiers
- tsp:, a character string that uniquely identifies a core component of the OpenURL Framework, which for Transports must be tsp:
- a character string that is assigned on registration and identifies the network protocol used by the Transport followed by a colon character (':')
- a character string that is assigned on registration and identifies the actual Transport.

Registry Identifiers of Transports are used primarily to support Registry management and to identify Transports in Community Profiles. In typical use, Registry Identifiers of Transports do not show up in Representations of ContextObjects or their Entities.

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Core Component</th>
<th>Network Protocol</th>
<th>Transport</th>
<th>Registry Identifier</th>
</tr>
</thead>
</table>
Table 9 illustrates the construction of Registry Identifiers for three Transports. (See Sections 20 through 22 for all six Transports that are in the initial Registry.)

11 Defining Applications: Community Profiles [Registry]

When communities create a new OpenURL Framework Application, they must make selections for each of the core components introduced so far. They must list these selections in a Community Profile that specifies the core characteristics of the Application.

A Community Profile defines the core characteristics of an Application as a list of Registry entries. This list contains Registry Identifiers for:

- One, and only one, ContextObject Format upon which the Application is built. Because of the nature of ContextObject Formats, this implies a selection of:
  - One Serialization
  - One Constraint Language
  - One or more Character Encodings
  - A set of constraints on the type and number of Entities that may be described in a ContextObject
  - A set of constraints on the type and number of Descriptors that may be used for the description of Entities of a ContextObject
  - A constraint on the number of ContextObjects that may be represented in an instance document that conforms to the ContextObject Format
- Zero or more registered Metadata Formats that may be used to describe Entities with By-Value and/or By-Reference Metadata Descriptors. Because of the nature of registered Metadata Formats, this choice implies a selection of:
  - For registered Metadata Formats used in By-Value Metadata Descriptors:
    - One Serialization, which must be the Serialization used by the ContextObject Format
    - One Constraint Language, which must the Constraint Language used by the ContextObject Format
    - One or more Character Encodings, which must be the same as those used by the ContextObject Format
  - For registered Metadata Formats used in By-Reference Metadata Descriptors:
    - One or more Serializations, which may be the same as the Serialization used by the ContextObject Format
    - One or more Constraint Languages, which may be the same as the Constraint Language used by the ContextObject Format
    - One or more Character Encodings, which should be the same as those used by the ContextObject Format
- Zero or more Namespaces that may be used to describe Entities with an Identifier Descriptor
- One or more Transports that specify how ContextObject Representations in the chosen ContextObject Format may be transported
A Community Profile must be expressed with an XML Document that conforms to the XML Schema provided in the Registry at <http://www.openurl.info/registry/docs/info:ofi/fmt:xml:xsd:pro>, where it is registered as a Format. The Dublin Core metadata [18] of this Registry entry are:

<table>
<thead>
<tr>
<th>info:ofi/fmt:xml:xsd:pro</th>
<th>dc:title</th>
<th>XML Format to represent Community Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc:creator</td>
<td>NISO Committee AX, OpenURL Standard Committee</td>
<td></td>
</tr>
</tbody>
</table>

In addition to the mandatory expression of a Community Profile as an XML Document, it is strongly recommended that communities create a human readable description of their Application and its corresponding Community Profile for the benefit of implementers.

This Standard does not prescribe or limit Resolver responses to service requests. However, a Resolver that conforms with a Community Profile should be able to process requests that are valid according to that Community Profile. A Resolver that conforms with a Community Profile may ignore requests that contain items not specified in the Community Profile.

Community Profiles must be registered before use in an Application.

Communities may use Community Profiles already in the Registry, or they may register additional Community Profiles as needed.

Upon registration, a Community Profile is assigned a Registry Identifier, formed by concatenating three character strings:

- info:ofi/, which represents the namespace under the info scheme reserved for Registry Identifiers
- pro:, a character string that uniquely identifies a core component of the OpenURL Framework, which for Community Profiles must be pro:
- a character string that is assigned on registration and identifies the Community Profile.

Registry Identifiers of Community Profiles are used primarily to support Registry management and to identify Community Profiles. In typical use, Registry Identifiers of Community Profiles do not show up in ContextObject Representations.

Table 10: Registry Identifiers for Community Profiles

<table>
<thead>
<tr>
<th>“info” URI Namespace</th>
<th>Core Component</th>
<th>Name</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>info:ofi/</td>
<td>pro:</td>
<td>sap1</td>
<td>info:ofi/pro:sap1</td>
</tr>
<tr>
<td>info:ofi/</td>
<td>pro:</td>
<td>sap2</td>
<td>info:ofi/pro:sap2</td>
</tr>
</tbody>
</table>

Table 10 illustrates the construction of Registry Identifiers for the two Community Profiles in the initial Registry. (See Sections 15 and 19.)

Table 11 excerpts the Level 1 San Antonio Community Profile (SAP1), available in the Registry at <http://www.openurl.info/registry/docs/pro/info:ofi/pro:sap1>. This excerpt defines some cardinality constraints on the KEV ContextObject Format, upon which SAP1 is built. (See Section 13.1.)
### Table 11: SAP1 Community Profile, Excerpt

```
<context-object-format>
  <context-object minOccurs="1" maxOccurs="1">
    <referent minOccurs="1" maxOccurs="1">
      <identifier minOccurs="0" maxOccurs="unbounded"/>
      <by-value-metadata minOccurs="0" maxOccurs="1"/>
      <by-reference-metadata minOccurs="0" maxOccurs="1"/>
      <private-data minOccurs="0" maxOccurs="1"/>
    </referent>
    <referring-entity minOccurs="0" maxOccurs="1">
      <identifier minOccurs="0" maxOccurs="unbounded"/>
      <by-value-metadata minOccurs="0" maxOccurs="1"/>
      <by-reference-metadata minOccurs="0" maxOccurs="1"/>
      <private-data minOccurs="0" maxOccurs="1"/>
    </referring-entity>
    <requester minOccurs="0" maxOccurs="1">
      <identifier minOccurs="0" maxOccurs="unbounded"/>
      <by-value-metadata minOccurs="0" maxOccurs="1"/>
      <by-reference-metadata minOccurs="0" maxOccurs="1"/>
      <private-data minOccurs="0" maxOccurs="1"/>
    </requester>
    <service-type minOccurs="0" maxOccurs="1">
      <identifier minOccurs="0" maxOccurs="unbounded"/>
      <by-value-metadata minOccurs="0" maxOccurs="1"/>
      <by-reference-metadata minOccurs="0" maxOccurs="1"/>
      <private-data minOccurs="0" maxOccurs="1"/>
    </service-type>
    <resolver minOccurs="0" maxOccurs="1">
      <identifier minOccurs="0" maxOccurs="unbounded"/>
      <by-value-metadata minOccurs="0" maxOccurs="1"/>
      <by-reference-metadata minOccurs="0" maxOccurs="1"/>
      <private-data minOccurs="0" maxOccurs="1"/>
    </resolver>
    <referrer minOccurs="0" maxOccurs="1">
      <identifier minOccurs="0" maxOccurs="unbounded"/>
      <by-value-metadata minOccurs="0" maxOccurs="1"/>
      <by-reference-metadata minOccurs="0" maxOccurs="1"/>
      <private-data minOccurs="0" maxOccurs="1"/>
    </referrer>
  </context-object>
</context-object-format>
```
The OpenURL Framework for Context-Sensitive Services

Part 2: The KEV ContextObject Format

Part 1 (Sections 5 through 11) defines the core components of the OpenURL Framework: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles.

Parts 2, 3, and 4 (Sections 12 through 22) define instances of these core components that illustrate the abstract concepts of Part 1. These instances form the initial content of the Registry. Each instance is described, given a Registry Identifier, and entered into the Registry at <http://www.openurl.info/registry/>. The initial Registry launches two Applications of the OpenURL Framework Standard intended for the scholarly-information community. The first Application provides a migration path from OpenURL 0.1 to the OpenURL Framework Standard. The second Application provides a path for future growth by harnessing the full expressive power of XML.

Part 2 defines a ContextObject Format inspired by the query string of the HTTP(S) GET request as specified in OpenURL 0.1. Part 3 defines a ContextObject Format based on XML. Part 4 defines six Transports. Four of these Transports are generic and may be used with any ContextObject Format. Two of the Transports are developed specifically for the ContextObject Format defined in Part 2 to provide a migration path from OpenURL 0.1 to this Standard.

Part 2 (Sections 12 through 15) defines a particular instance of a ContextObject Format inspired by the query string of the HTTP(S) GET request as specified in OpenURL 0.1. The Key/Encoded-Value ContextObject Format defines how to represent a ContextObject as a string of ampersand-delimited Key/Encoded-Value pairs. In the remainder of this Standard, the term Key/Encoded-Value will be abbreviated to KEV.

Section 12 describes and registers the following instances of core components necessary to define the KEV ContextObject Format: the KEV Serialization, the Z39.88-2004 Matrix Constraint Language, and Constraint Definitions that define the KEV ContextObject Format and illustrate KEV Metadata Formats. Sections 13 and 14 apply the KEV ContextObject Format to obtain KEV ContextObject Representations.

Using the KEV ContextObject Format, a ContextObject is represented as a URL-encoded form ready for transport by HTTP(S) GET and HTTP(S) POST. The Inline OpenURL Transports defined in Section 22 transport a KEV ContextObject Representation as the query string of an HTTP(S) GET request or as the message body of an HTTP(S) POST. These Inline Transports, the generic Transports of Sections 20 and 21, and the KEV ContextObject Format form the basis for an easy migration path from OpenURL 0.1 to this Standard. This migration path is formalized in an Application defined by the Level 1 San Antonio Community Profile; see Section 15 and Appendix C.
12 The KEV ContextObject Format

This Section introduces Format triples of the KEV ContextObject Format and of the KEV Metadata Formats necessary to describe Entities. These Format triples consist of instances of core components, which are identified, described, and entered into the Registry (see Section 6) at <http://www.openurl.info/registry>. The Format triples consist of:

- The KEV Serialization (Section 12.1)
- The Z39.88-2004 Matrix Constraint Language (Section 12.2)
- Constraint Definitions expressed in the Z39-88-2004 Matrix Constraint Language for the KEV ContextObject Format (Sections 12.3.1 and 13) and for the KEV Metadata Formats (Sections 13 and 14).

While Sections 12, 13, and 14 introduce, describe, and illustrate these elements, the Registry is the authoritative source for their complete specification.

12.1 The KEV Serialization

The KEV Serialization resembles the query component of an HTTP GET request. Often, HTTP GET requests are constructed to transmit information from a user agent to a processing agent. The user agent builds an HTTP URI query component from an HTML form data set on a GET method request and appends this component to an HTTP URI with a question-mark character (‘?’) as a separator. The processing agent residing at this HTTP URI interprets and processes the query component. The syntax of the query component is a list of key/value pairs delimited by ampersand characters (‘&’), such as:

    key1=value1&key2=value2

The key/value pairs are delimited by an equals character (‘=’) and concatenated with an ampersand character (‘&’). Keys may occur multiple times in order to associate multiple values with each key. IETF RFC 2396 [6] reserves the following characters for special use within the query component:

- ‘;’, ‘/’, ‘?’; ‘:’, ‘@’, ‘&’, ‘=’, ‘+’, ‘$’, and ‘.’

These characters must be escaped by URL-encoding.

The key/value syntax is also used on hyperlinks embedded in HTML documents to send parameters to a processing agent. Similarly, on a request that uses the POST method, user agents use the same syntax to include the form data set within an HTTP entity body.

Keys must be constructed from characters that remain invariant under URL-encoding (also known as safe characters). Values may be constructed from both safe and unsafe characters and must be URL-encoded. This explains the name Key/Encoded-Value or KEV for this Serialization.

To simplify the descriptions that follow, we include a leading ampersand character (‘&’) with each KEV pair, as in &key=value. By doing this, the complete KEV Serialization is a simple concatenation of KEV pairs.
12.2 The Z39.88-2004 Matrix Constraint Language

The Z39.88-2004 Matrix Constraint Language is used to specify constraints for descriptions of resources expressed using the KEV Serialization. The Z39.88-2004 Matrix Constraint Language is used to define the syntax and semantics of the KEV ContextObject Format and KEV Metadata Formats.


Table 12: Structure of the Z39.88-2004 Matrix

<table>
<thead>
<tr>
<th>Delim</th>
<th>Key</th>
<th>Equals</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>[* Key *]</td>
<td>=</td>
<td>&lt;[* Value *]&gt;</td>
<td>0</td>
<td>1</td>
<td>[* Item definition *]</td>
</tr>
<tr>
<td>#</td>
<td>[* ... *]</td>
<td>[* ... *]</td>
<td>[* ... *]</td>
<td>[* ... *]</td>
<td>[* ... *]</td>
<td>[* ... *]</td>
</tr>
</tbody>
</table>

Table 12 shows the structure of a Z39.88-2004 Matrix. It consists of the following columns:

- Delim: the ampersand character (‘&’) delimiter for rows containing syntax rules or the hash character (‘#’) for comment rows
- Key: the key being defined
- Equals character (‘=’)
- Value: the data type for the value associated with the key
- Min: the minimum occurrence allowed for the key; an integer
- Max: the maximum occurrence allowed for the key; an integer or an asterisk character (‘*’) to denote ‘unbounded’
- Description: a full name of the key, a semantic definition of the key, and any further information

Each row of the Z39.88-2004 Matrix with an ampersand character (‘&’) in the first column describes the construction of a valid KEV pair. Rows of the Z39.88-2004 Matrix that have a hash character (‘#’) in the first column are comment rows and must be ignored.

One valid KEV pair is obtained by concatenating table entries from the first four columns of a Z39.88-2004 Matrix row that begins with an ampersand character (‘&’). Several valid KEV pairs may be concatenated to obtain a description of a resource compliant with a Z39.88-2004 Constraint Definition. The order in which KEV pairs are concatenated is not important.

In comment rows, replace the character string "[* ... *]" with descriptive text. Descriptive text must not occur in the Delim column. Usually, only the Description column contains descriptive text.

In the Key column of non-comment rows, the character string "[* Key *]" must be replaced with the name of a valid key.

The Value column of a non-comment row of the Z39.88-2004 Matrix assigns a data type to the key, and "[* Value *]" should be replaced with one of the following available data types:

- <data>: character string
- <id>: character string for an Identifier (Section 5.2.1)
12.3 Constraint Definitions in the KEV ContextObject Format

The main Constraint Definition associated with the KEV Serialization and the Z39.88-2004 Matrix Constraint Language is the KEV ContextObject Format. This Format defines the Representation of a ContextObject as a concatenation of KEV pairs of the form &key=value.

In addition, there are Constraint Definitions known as KEV Metadata Formats that define the Representation of Entities of ContextObjects as a concatenation of KEV pairs. These Representations may be used for both By-Value and/or By-Reference Metadata Descriptors.

In the Registry, a Constraint Definition for a Format expressed in the Z39.88-2004 Matrix Constraint Language is described by the following metadata:

- dc:title: the title of the Format
- dc:creator: the name of the community that defined the Format
- dc:description: a brief description of the Format
- dc:identifier: a locator of the Z39.88-2004 Matrix that defines the Format
- dcterms:created: the date when the Format was created
- dcterms:modified: the date when the Format was modified

Z39.88-2004 Matrix definitions are primarily intended for human reading. To this end, the XHTML Matrix has an associated style sheet that displays the first four rows of each column in bold type to highlight the syntax embedded in the Matrix. However, machine reading is supported, and each cell of the Matrix has an associated class attribute. The W3C XHTML validator button at the foot of the page should be used to validate the XHTML Matrix.

A template for the Z39.88-2004 Matrix that may be used in the creation of KEV Metadata Formats is available in Appendix B and in the Registry at <http://www.openurl.info/registry/docs/html/mtx.html>.

12.3.1 Z39.88-2004 Matrix Constraint Definition for the KEV ContextObject Format


The Z39.88-2004 Matrix that defines the KEV ContextObject Format is available at <http://www.openurl.info/registry/docs/mtx/info:ofi/fmt:kev:mtx:ctx>. Table 13 is an excerpt that shows the administrative keys (names starting with ctx) and the Referent keys (names starting with rft). Section 13.2 specifies all keys that may occur in a KEV ContextObject Representation.
## Table 13: Z39.88-2004 Matrix Constraint Definition of KEV ContextObject Format, Excerpt

<table>
<thead>
<tr>
<th>Delim</th>
<th>Key</th>
<th>Equals</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>ctx_</td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>Administration. As Admin is an optional field in a ContextObject, any of the keys with prefix ctx_ may be present.</td>
</tr>
<tr>
<td>&amp;</td>
<td>ctx_ver</td>
<td>=</td>
<td>Z39.88-2004</td>
<td>0</td>
<td>1</td>
<td>ContextObject version. This has a fixed value.</td>
</tr>
<tr>
<td>&amp;</td>
<td>ctx_enc</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0</td>
<td>1</td>
<td>ContextObject encoding. The value for ctx_enc specifies the character encoding used in the ContextObject. Legitimate values are taken from the IANA list at <a href="http://www.iana.org/assignments/character-sets">http://www.iana.org/assignments/character-sets</a>. The values to be used in the ContextObject are those listed next to Name or—if available—the values with an indication of 'preferred MIME name' in the IANA list. UTF-8 is the default value, representing UTF-8 encoded Unicode.</td>
</tr>
<tr>
<td>&amp;</td>
<td>ctx_id</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0</td>
<td>1</td>
<td>ContextObject Identifier.</td>
</tr>
<tr>
<td>&amp;</td>
<td>ctx_tim</td>
<td>=</td>
<td>&lt;time&gt;</td>
<td>0</td>
<td>1</td>
<td>ContextObject timestamp. YYYY-MM-DD or YYYY-MM-DDThh:mm:ssTZD</td>
</tr>
<tr>
<td>#</td>
<td>rft_</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>Referent. As Referent is a mandatory Entity in a ContextObject, at least one of the keys with prefix rft_ must be present</td>
</tr>
<tr>
<td>&amp;</td>
<td>rft_id</td>
<td>=</td>
<td>&lt;id&gt;</td>
<td>0</td>
<td>*</td>
<td>Referent Identifier. Multiple instances of rft_id do not indicate multiple Referents, but rather multiple ways to identify a single Referent</td>
</tr>
<tr>
<td>&amp;</td>
<td>rft_val_fmt</td>
<td>=</td>
<td>&lt;fmt-id&gt;</td>
<td>0</td>
<td>1</td>
<td>Identifier of By-Value Metadata Format for a Referent. Identifier of the Metadata Format used for the description of the Referent through By-Value Metadata</td>
</tr>
<tr>
<td>#</td>
<td>rft_val</td>
<td>=</td>
<td></td>
<td>0</td>
<td>0</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>&amp;</td>
<td>rft.&lt;m-key&gt;</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0</td>
<td>*</td>
<td>By-Value Metadata key for a Referent. The &lt;m-key&gt; is a key defined in the KEV Metadata Format specified by the value of the rft_val_fmt key, which must be present. Use of the rft. prefix is mandatory.</td>
</tr>
<tr>
<td>&amp;</td>
<td>rft_ref_fmt</td>
<td>=</td>
<td>&lt;fmt-id&gt;</td>
<td>0</td>
<td>1</td>
<td>By-Reference Metadata Format for a Referent. The rft_ref key must also be present.</td>
</tr>
<tr>
<td>&amp;</td>
<td>rft_ref</td>
<td>=</td>
<td>&lt;url&gt;</td>
<td>0</td>
<td>1</td>
<td>Location of By-Reference Metadata for a Referent. The rft_ref_fmt key must also be present. The Resolver should retrieve the metadata from the specified location.</td>
</tr>
<tr>
<td>&amp;</td>
<td>rft_dat</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0</td>
<td>1</td>
<td>Referent Private Data</td>
</tr>
</tbody>
</table>

### 12.3.2 Z39.88-2004 Matrix Constraint Definitions for KEV Metadata Formats

The Z39.88-2004 Matrix Constraint Language is also used to define KEV Metadata Formats. Table 26 in Appendix C contains the list of KEV Metadata Formats that are in the initial Registry. For each of these KEV Metadata Formats, the Registry at <http://www.openurl.info/registry> contains a complete and authoritative Constraint Definition.

Table 14 is an excerpt of a Constraint Definition to describe a class of Entities of the type “book”. The complete Constraint Definition is available at <http://www.openurl.info/registry/docs/mtx/info:ofi/fmt:kev:mtx:book>.
Keys specified in Z39.88-2004 Matrix Constraint Definitions that define KEV Metadata Formats must consist of alphanumeric characters only. They must not contain underscore characters ('_').

Table 14: Z39.88-2004 Matrix Constraint Definition of KEV Metadata Format for “book”, Excerpt

<table>
<thead>
<tr>
<th>Delim</th>
<th>Key</th>
<th>Equals</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>aulast</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>aufirst</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>auinit</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>auinit1</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>auinitm</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>ausuffix</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>au</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 *</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>aucorp</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>btitle</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>atitle</td>
<td>=</td>
<td>&lt;data&gt;</td>
<td>0 1</td>
<td></td>
</tr>
</tbody>
</table>

Description

First author's family name. This may be more than one word. In many citations, the author's family name is recorded first and is followed by a comma, i.e. Smith, Fred James is recorded as aulast=smith.

First author's given name or names or initials. This data element may contain multiple words and punctuation, i.e. "Fred F", "Fred James".

First author's first and middle initials.

First author's first initial.

First author's middle initial.

First author's name suffix. Qualifiers on an author's name such as "Jr." or "III" are entered here. For example, Smith, Fred Jr. is recorded as ausuffix=jr.

This data element contains the full name of a single author; "Smith, Fred M" or "Harry S. Truman", for example.

Organization or corporation that is the author or creator of the book; "Mellon Foundation", for example.

The title of the book. This can also be expressed as title, for compatibility with version 0.1; "moby dick or the white whale", for example.

Chapter title. Chapter title is included if it is a distinct title; "The Push Westward.", for example.

Book title. Provided for compatibility with version 0.1. Prefer btitle.

13 KEV ContextObject Representations

Registry Identifier info:ofi/fmt:kev:mtx:ctx

The KEV Format represents one, and only one, ContextObject as a string of ampersand-delimited pairs, each pair consisting of a key and an associated value that must be URL-encoded.

The KEV ContextObject Format triple consists of:

- The KEV Serialization (Section 12.1), recorded in the Registry under Registry Identifier info:ofi/fmt:kev
- The Z39.88-2004 Matrix Constraint Language (Section 12.2), recorded in the Registry under Registry Identifier info:ofi/fmt:kev:mtx
- The Z39.88-2004 Matrix Constraint Definition (Section 12.3), recorded in the Registry under Registry Identifier info:ofi/fmt:kev:mtx:ctx

Example 11 displays a KEV ContextObject Representation. The first part of the example is formatted for readability, and the second part is the actual KEV ContextObject Representation with URL-encoded values (see Section 13.4). This example includes administrative keys (beginning with ctx),
two Identifier Descriptors to describe the Referent (beginning with rft), Identifier Descriptors for the ReferringEntity, Requester, and Referrer (rfe_id, req_id, and rfr_id, respectively). (Example 13 will show the use of By-Value Metadata in a KEV ContextObject Representation.)

Example 11: KEV ContextObject Representation

Formatted for readability:
```plaintext
ctx_ver = Z39.88-2004
& ctx_enc = info:ofi/enc:UTF-8
& ctx_id = 456
& ctx_tim = 2002-03-20T08:55:12Z
& rft_id = info:doi/10.1126/science.275.5304.1320
& rft_id = info:pmid/9036860
& rfe_id = info:doi/10.1006/mthe.2000.0239
& req_id = mailto:jane.doe@caltech.edu
& rfr_id = info:sid/elsevier.com:ScienceDirect
```

URL-encoded:
```plaintext
```

13.1 Cardinality Constraints on the KEV ContextObject Format

The KEV ContextObject Format restricts the number of Entities that may be present in each ContextObject, the number of Descriptors that may be used to describe Entities, and the number of ContextObjects that may be bundled in a single KEV Representation. These constraints are specified and summarized in Table 15 (compare this with the fundamental restrictions of Table 1).

<table>
<thead>
<tr>
<th>Entity</th>
<th>Number</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identifier</td>
</tr>
<tr>
<td>Referent</td>
<td>1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Requester</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ServiceType</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Resolver</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Referrer</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ContextObjects</td>
<td>1</td>
<td>N/A</td>
</tr>
</tbody>
</table>

13.2 Keys in the KEV ContextObject Format

The rules for the creation of KEV pairs are:

- The first character of a key must be alphanumeric. The other characters of keys must be alphanumeric, the underscore character ("_"), or the dot character (".").
A key **must** be separated from its associated value by an equals character (‘=’).

Values **must** be URL-encoded (see Section 14.4).

The default Character Encoding for values is UTF-8 encoded Unicode, but it is possible to declare the use of other Character Encodings (see Section 14.3).

KEV pairs **must** be concatenated using the ampersand character (‘&’) to form a single string.

There are five types of keys in the **KEV ContextObject Format**:

- Keys to identify **Entity Descriptors**
- Keys to identify **Metadata Formats** used for By-Value Metadata Descriptors
- Keys to identify **Metadata Formats** used for By-Reference Metadata Descriptors
- Keys to specify administrative information about the **ContextObject**
- Metadata keys of a **KEV Metadata Format**

Sections 13.2.1 through 13.2.4 examine the first four types of keys. Section 14.2 examines metadata keys of **KEV Metadata Formats**.

### 13.2.1 Keys for Entity Descriptors

Keys of **Entity Descriptors** **must** contain at least one underscore character (‘_’). As shown in Table 16, they are a concatenation of:

- an abbreviated form of the **Entity** name (The column under the heading **Entities** lists the full name of each **Entity** together with its abbreviated form.)
- an underscore character (‘_’)
- an abbreviated form of the **Descriptor** name (The row under the heading **Descriptors** lists the full name of each **Descriptor** and its abbreviated form.)

For example, the key `rft_id` indicates a Referent (rft) described by an Identifier (id).

#### Table 16: KEV ContextObject Format – Keys for Entity Descriptors

<table>
<thead>
<tr>
<th>Entities</th>
<th>Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Referent</strong></td>
<td><strong>rd</strong></td>
</tr>
<tr>
<td><strong>rft</strong></td>
<td><strong>rft_id</strong></td>
</tr>
<tr>
<td></td>
<td><strong>rft_val_fmt</strong> (13.2.2)</td>
</tr>
<tr>
<td></td>
<td><strong>rft_ref_fmt</strong></td>
</tr>
<tr>
<td></td>
<td><strong>rft_dat</strong></td>
</tr>
<tr>
<td><strong>ReferringEntity</strong></td>
<td><strong>rd</strong></td>
</tr>
<tr>
<td><strong>rfe</strong></td>
<td><strong>rfe_id</strong></td>
</tr>
<tr>
<td></td>
<td><strong>rfe_val_fmt</strong> (13.2.2)</td>
</tr>
<tr>
<td></td>
<td><strong>rfe_ref_fmt</strong></td>
</tr>
<tr>
<td></td>
<td><strong>rfe_dat</strong></td>
</tr>
<tr>
<td><strong>Requester</strong></td>
<td><strong>rd</strong></td>
</tr>
<tr>
<td><strong>req</strong></td>
<td><strong>req_id</strong></td>
</tr>
<tr>
<td></td>
<td><strong>req_val_fmt</strong> (13.2.2)</td>
</tr>
<tr>
<td></td>
<td><strong>req_ref_fmt</strong></td>
</tr>
<tr>
<td></td>
<td><strong>req_dat</strong></td>
</tr>
<tr>
<td><strong>ServiceType</strong></td>
<td><strong>rd</strong></td>
</tr>
<tr>
<td><strong>svc</strong></td>
<td><strong>svc_id</strong></td>
</tr>
<tr>
<td></td>
<td><strong>svc_val_fmt</strong> (13.2.2)</td>
</tr>
<tr>
<td></td>
<td><strong>svc_ref_fmt</strong></td>
</tr>
<tr>
<td></td>
<td><strong>svc_dat</strong></td>
</tr>
<tr>
<td><strong>Resolver</strong></td>
<td><strong>rd</strong></td>
</tr>
<tr>
<td><strong>res</strong></td>
<td><strong>res_id</strong></td>
</tr>
<tr>
<td></td>
<td><strong>res_val_fmt</strong> (13.2.2)</td>
</tr>
<tr>
<td></td>
<td><strong>res_ref_fmt</strong></td>
</tr>
<tr>
<td></td>
<td><strong>res_dat</strong></td>
</tr>
<tr>
<td><strong>Referrer</strong></td>
<td><strong>rd</strong></td>
</tr>
<tr>
<td><strong>rfr</strong></td>
<td><strong>rfr_id</strong></td>
</tr>
<tr>
<td></td>
<td><strong>rfr_val_fmt</strong> (13.2.2)</td>
</tr>
<tr>
<td></td>
<td><strong>rfr_ref_fmt</strong></td>
</tr>
<tr>
<td></td>
<td><strong>rfr_dat</strong></td>
</tr>
</tbody>
</table>
13.2.2 Keys for By-Value Metadata Descriptors

Metadata keys **must** consist of alphanumeric characters only. Metadata keys **must not** contain underscore characters ('_').

When used in *By-Value Metadata*, metadata keys **must** be preceded by an *Entity* prefix (the abbreviated *Entity* name listed in the first column of Table 16) followed the period character ('.').

For example, the prefix `rfe.` and metadata key `au` combined as `rfe.au` denotes the author of a *ReferringEntity*, while `rft.au` would refer to the author of the *Referent*.

13.2.3 Keys for By-Reference Metadata Descriptors

When used in *By-Reference Metadata*, metadata keys **must not** be preceded by any prefix.

It is anticipated that *By-Reference Metadata* are constructed well before the *ContextObject* is formed (presumably as part of a database). At that time, the item described by this metadata is not yet a particular *Entity*: it could be a *Referent* in one *ContextObject* and a *ReferringEntity* in another *ContextObject*.

13.2.4 Keys for Administrative Data

Keys for administrative data about the *ContextObject* are composed of the prefix `ctx` for *ContextObject*, the underscore character ('_'), and an abbreviated suffix. Table 17 gives the keys and their cardinality constraints.

<table>
<thead>
<tr>
<th>Key</th>
<th>Number</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx_enc</td>
<td>≤ 1</td>
<td>Character Encoding. Registry Identifier of the form <code>info:ofi/enc:</code>_ (Section 13.3)</td>
</tr>
<tr>
<td>ctx_id</td>
<td>≤ 1</td>
<td>Identifier of <em>ContextObject</em></td>
</tr>
<tr>
<td>ctx_tim</td>
<td>≤ 1</td>
<td>ISO 8601datetime specifying the time of creation of the <em>ContextObject</em></td>
</tr>
</tbody>
</table>

Note: The `ctx_id` key **may** have limited use for *KEV ContextObject Representations*. It is included for consistency with the *XML ContextObject Format* (see Part 3) and possibly other *ContextObject Formats*. *XML ContextObject Representations* are more likely to be stored in databases, in which case a *ContextObject* identifier might be helpful for fast retrieval.

13.3 Character Encoding in the KEV ContextObject Format

UTF-8 encoded Unicode is the default *Character Encoding* of the *KEV ContextObject Format*. The use of a different *Character Encoding** must be specified in the `ctx_enc` KEV pair:

- The default value associated with the `ctx_enc` key is `info:ofi/enc:UTF-8`, the Registry Identifier of UTF-8 encoded Unicode. This value specifies the use of Unicode as the character set and UTF-8 as the character encoding of that character set throughout the KEV *ContextObject Representation*. When the *Character Encoding* is UTF-8 encoded Unicode, the `ctx_enc` KEV pair is **optional**.
- Values associated with the `ctx_enc` key **must** be Registry Identifiers of Character Encodings taken from the Registry. Registered character sets **must** be from the Internet Assigned Naming Authority (IANA) List of Registered Character Sets [7]. Upon registration, a unique Registry Identifier of the form `info:ofi/enc:name` is assigned to the character set. In `info:ofi/enc:name`, the character string `name` is taken from the IANA list, as defined in Section 8.1.
The character set and character encoding for all characters used in a KEV ContextObject Representation must follow the corresponding specification shown in the IANA list.

The Character Encodings that are initially registered are listed in Table 26 of Appendix C.

By-Reference Metadata must use Unicode as the character set and UTF-8 as the character encoding, unless the By-Reference Metadata description explicitly declares otherwise.

### 13.4 URL-Encoding in the KEV ContextObject Format

Values of KEV pairs must be URL-encoded to ensure that the KEV ContextObject Representation is ready to be transported over the HTTP(S) protocol. URL-encoding eliminates confusion that could occur when special characters, such as equals character (‘=’) and ampersand character (‘&’), are used within values of KEV pairs.

Rules for URL-encoding values are:

- The alphanumeric characters (letters and digits), the period character (‘.’), the hyphen character (‘-‘), the asterisk character (‘*‘), and the underscore character (‘_‘) remain the same.
- The space character (‘ ‘) is converted into a plus sign (‘+’) or into the character string “%20”.
- For all other characters, each byte of the character is converted into a three-character string “%XY” where “XY” is the two-digit hexadecimal representation of the byte.

### 14 Entity Descriptors in the KEV ContextObject Format

A Descriptor specifies information about an Entity. There are four types of Descriptors that may be used in the KEV ContextObject Format: Identifier, By-Value Metadata, By-Reference Metadata, and Private Data.

#### 14.1 Identifier Descriptors

An Identifier Descriptor specifies an Entity by means of a Uniform Resource Identifier (URI). This URI may be associated with the Entity itself or with metadata for the Entity. As described in Section 13.2.1, keys to identify Identifier Descriptors in the KEV ContextObject Format consist of two parts separated by an underscore character (‘_‘). The first part identifies the Entity, and the second part is the character string id, which specifies that the type of Descriptor is an Identifier Descriptor. For example, the key rft_id denotes an Identifier Descriptor for a Referent.

**Example 12: Identifier Descriptors in a KEV ContextObject Representation**

Formatted for readability:

```
& rft_id = info:doi/10.1126/science.275.5304.1320
& rft_id = info:pmid/9036860
& rfe_id = info:doi/10.1006/mthe.2000.0239
& req_id = mailto:jane.doe@caltech.edu
& rfr_id = info:sid/elsevier.com:ScienceDirect
& res_id = http://links.caltech.edu/menu
```

URL-encoded:

```
```
Example 12 shows several Identifier Descriptors as they would occur in a KEV ContextObject Representation.

14.2 By-Value and By-Reference Metadata Descriptors

A Metadata Format provides a concrete set of descriptive elements for the purpose of representing an Entity. For compatibility, Metadata Formats and the ContextObject Format must be based on the same Serialization and Constraint Language. This compatibility rule is waived for By-Reference Metadata, provided the Metadata Format is registered (see Section 14.2.1).

Metadata Formats used in the OpenURL Framework may be registered. Unregistered Metadata Formats must meet the requirements described in Section 9.2.

14.2.1 Rules Guiding By-Value and By-Reference Metadata Descriptors

The general rules for Metadata Formats are given in Section 9.2. This Section gives the rules for creating By-Value Metadata and By-Reference Metadata in the KEV ContextObject Format. The KEV ContextObject Format accommodates both registered and unregistered Metadata Formats.

- **Registered Metadata Formats**
  - Registered Metadata Formats must be identified by means of the Registry Identifier of the Metadata Format. The Registry maintains a one-to-one correspondence between the definition of a Metadata Format and its Registry Identifier. The identification of the Metadata Format must be provided as the value of a key with the suffix _fmt.
  - The corresponding By-Value Metadata Descriptor must use the KEV Serialization and must be valid according to a Z39.88-2004 Matrix. This Matrix must be in the Registry and correspond uniquely with the Registry Identifier used to identify the Metadata Format. The Registry Identifier of the Metadata Format must be of the form: info:ofi/fmt:kev:mtx:format_name. Note that validity refers to the string of KEV pairs after removal of the rft., rfe., rfr., req., res., and svc. prefixes. (See Section 13.2.2.)
  - The corresponding By-Reference Metadata Descriptor must be an instance document that conforms to the Metadata Format identified by the Registry Identifier. Because the Metadata Format is registered, the By-Reference Metadata Descriptor may use any registered Serialization, and the Metadata Format to which it conforms may use any registered Constraint Language. The By-Reference Metadata Descriptor is not limited to the KEV Serialization or the Z39.88-2004 Matrix Constraint Language.

- **Unregistered Metadata Formats**
  - Unregistered Metadata Formats must be identified by means of a URL that specifies the network location of the Z39.88-2004 Matrix that defines the KEV Metadata Format. The identification of the Metadata Format must be provided as the value of a key with the suffix _fmt. For example, a Metadata Format could be identified as: http://www.example.net/x-service.html.
  - The corresponding By-Value Metadata or By-Reference Metadata Descriptor must use the KEV Serialization; it must be a string of ampersand-delimited KEV pairs that is valid according to the Z39.88-2004 Matrix at the network location specified by the aforementioned URL. Note that validity refers to the string of KEV pairs after removal of the rft., rfe., rfr., req., res., and svc. prefixes. (See Section 13.2.2.)
14.2.2 By-Value Metadata Descriptors

A KEV By-Value Metadata Descriptor consists of:

- A KEV pair that specifies the URI of a Metadata Format as the value associated with a key of the form *_val_fmt. (The * stands for the abbreviated form of an Entity name.) This Metadata Format must be defined by means of a Z39.88-2004 Matrix Constraint Definition.

- A set of KEV pairs in which values are assigned to metadata keys. These keys must be valid according to the specified Metadata Format.

Example 13: By-Value Metadata Descriptor in a KEV ContextObject Representation

Formatted for readability:
```
& rft_val_fmt = info:ofi/fmt:kev:mtx:journal
& rft.atitle = Isolation of a common receptor for coxsackie B
& rft.jtitle = Science
& rft.aulast = Bergelson
& rft.auinit = J
& rft.date = 1997
& rft.volume = 275
& rft.spage = 1320
```

URL-encoded:
```
```

Example 13 shows a By-Value Metadata Descriptor for a Referent. In this example, the Referent is an article in a journal.


- KEV pairs that represent the Referent in this Metadata Format have keys with an rft. prefix. The character strings that follow the rft. prefix (atitle and jtitle, for example) are the key names defined in the Constraint Definition.

14.2.3 By-Reference Metadata Descriptors

A KEV By-Reference Metadata Descriptor consists of:

- A KEV pair that specifies the URI of a Metadata Format as the value associated with a key of the form *_ref_fmt. (The * stands for the abbreviated form of an Entity name.)

- A KEV pair that specifies the URL of a By-Reference Metadata description as the value associated with a key of the form *_ref.

Example 14: By-Reference Metadata Descriptor as a Property List

Formatted for readability:
```
& req_ref_fmt = http://lib.caltech.edu/fmt/ldap-mtx.html
& req_ref = http://ldap.caltech.edu/janed/record.txt
```
Example 14 shows a By-Reference Metadata Descriptor for a Requester. In this example, the Requester is a student at Caltech, identified by her LDAP record.

- The KEV pair `req_ref_fmt = http://lib.caltech.edu/fmt/ldap-mtx.html` specifies the Metadata Format. The key name indicates that this KEV pair specifies a By-Reference Metadata Format used to describe a Requester. The value identifies the Z39.88-2004 Matrix Constraint Definition of the Metadata Format. Since this Metadata Format is not registered, its Constraint Definition must be a Z39.88-2004 Matrix (see Sections 9.2 and 14.2.1).

- The KEV pair `req_ref = http://ldap.caltech.edu/janed/record.txt` specifies the location of the Descriptor of the Requester. The keys used in this Descriptor are defined in the Constraint Definition, and they must not be prefixed.

### 14.3 Private DataDescriptors

A Private Data Descriptor specifies information about the Entity using a method not defined in this Standard. This Standard does not provide any global mechanisms to interpret Private Data. Instead, it is assumed that the Resolver and the Referrer have a common understanding, based on a tacit or explicit bilateral agreement. To make it possible for the Resolver to interpret Private Data, a ContextObject that contains a Private Data Descriptor must identify the Referrer that created it.

As described in Section 13.2.1, keys to identify Private Data Descriptors in the KEV ContextObject Format consist of two parts separated by an underscore character ("_"). The first part identifies the Entity; the second part is the character string `dat` to specify that the Descriptor is a Private Data Descriptor. For example, the key `rfe_dat` is associated with a Private Data Descriptor for a ReferringEntity.

**Example 15: Private Data Descriptor in a KEV ContextObject Representation**

<table>
<thead>
<tr>
<th>Formatted for readability:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp; rfe_dat = cites/8///citedby/12</td>
</tr>
<tr>
<td>&amp; rfr_id = info:sid/elsevier.com:ScienceDirect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL-encoded:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;rfe_dat=cites%2F8%2F%2Fcitedby%2F12&amp;rfr_id=info%3Asid%2Felsevier.com%3AScienceDirect</td>
</tr>
</tbody>
</table>

Example 15 shows a Private Data Descriptor for a ReferringEntity.

- The KEV pair `rfe_dat = cites/8///citedby/12` is Private Data provided about the ReferringEntity. In this example, the ReferringEntity is a journal article identified by a proprietary identifier.

- The KEV pair `rfr_id = info:sid/elsevier.com:ScienceDirect` is an Identifier Descriptor of the Referrer, which might help the Resolver to interpret the Private Data.
14.4 Example of a KEV ContextObject Representation

Example 16 represents a complete ContextObject and combines several of the previous examples.

The initial four KEV pairs convey administrative information: the version of the ContextObject Format, the Character Encoding, the identifier of the ContextObject, and the time at which the ContextObject Representation was created.

The next nine KEV pairs form a By-Value Metadata Descriptor of the Referent. The rft_val_fmt KEV pair defines the Metadata Format by assigning the Registry Identifier of a Z39.88-2004 Matrix Constraint Definition (info:ofi/fmt:kev:mtx:journal) to the rft_val_fmt key. This is followed by KEV pairs whose keys consist of an rft. prefix and key names, such as atitle and jtitle, that are defined in the Constraint Definition. These KEV pairs are the By-Value Metadata.

The ten KEV pairs that follow form a By-Value Metadata Descriptor of the ReferringEntity. The structure of this part is identical to that used for the Referent.

The final two KEV pairs form a By-Reference Metadata Descriptor of the Requester. The req_ref_fmt KEV pair defines the Metadata Format by assigning the URL of a Z39.88-2004 Matrix Constraint Definition that defines the Metadata Format to the req_ref_fmt key. The req_ref KEV pair specifies the URL of the actual metadata.

Example 16: KEV ContextObject Representation

Formatted for readability:

```
ctx_ver = Z39.88-2004
& ctx_enc = info:ofi/enc:UTF-8
& ctx_id = 345871
& ctx_tim = 2002-03-20T08:55:12Z
& rft_val_fmt = info:ofi/fmt:kev:mtx:journal
& rft.atitle = Isolation of a common receptor for coxsackie B
& rft.jtitle = Science
& rft.au = Bergelson
& rft.auinit = J
& rft.date = 1997
& rft.volume = 275
& rft.spage = 1320
& rft.epage = 1323
& rfe_val_fmt = info:ofi/fmt:kev:mtx:journal
& rfe.atitle = p27-p16 Chimera: A Superior Antiproliferative
& rfe.jtitle = Molecular Therapy
& rfe.au = McArthur
& rfe.auinit = James
& rfe.date = 2001
& rfe.volume = 3
& rfe.issue = 1
& rfe.spage = 8
& rfe.epage = 13
& req_ref_fmt = http://lib.caltech.edu/fmt/ldap-mtx.html
& req_ref = http://ldap.caltech.edu/janed/record.txt
```
15 KEV-Based Community Profiles

A Community Profile lists a selection of Registry entries. This selection specifies the ContextObject Format, the Metadata Format(s), and the Transport(s) that form the core properties of an OpenURL Application. Further information on the creation of Community Profiles is found in Section 11.

A Resolver that conforms to the KEV ContextObject Format must process all items that conform to Registry entries specified in a Community Profile using the KEV ContextObject Format. Communities may define additional conformance rules in their Community Profiles.

Appendix C describes the Level 1 San Antonio Community Profile, which is an example of a Community Profile based on the KEV ContextObject Format. This Community Profile was developed by NISO Committee AX for the scholarly-information community. It deploys an Application that is similar to, but is more expressive than, the OpenURL 0.1 specification. In the remainder of this Standard, this Community Profile will be referred to as the SAP1 Community Profile. Its Registry Identifier is info:ofi/pro:sap1-2004.

Other communities are encouraged to use the KEV ContextObject Format to deploy their own OpenURL Applications. As specified in Section 11, each Application must be defined in a Community Profile. A straightforward way to deploy KEV-based Applications is to modify the SAP1 Community Profile to the needs of new communities.
The OpenURL Framework for Context-Sensitive Services

Part 3: The XML ContextObject Format

Part 1 (Sections 5 through 11) defines the core components of the OpenURL Framework: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles.

Parts 2, 3, and 4 (Sections 12 through 22) define instances of these core components that illustrate the abstract concepts of Part 1. These instances form the initial content of the Registry. Each instance is described, given a Registry Identifier, and entered into the Registry at <http://www.openurl.info/registry/>. The initial Registry launches two Applications of the OpenURL Framework Standard intended for the scholarly-information community. The first Application provides a migration path from OpenURL 0.1 to the OpenURL Framework Standard. The second Application provides a path for future growth by harnessing the full expressive power of XML.

Part 2 defines a ContextObject Format inspired by the query string of the HTTP(S) GET request as specified in OpenURL 0.1. Part 3 defines a ContextObject Format based on XML. Part 4 defines six Transports. Four of these Transports are generic and may be used with any ContextObject Format. Two of the Transports are developed specifically for the ContextObject Format defined in Part 2 to provide a migration path from OpenURL 0.1 to this Standard.

Part 3 (Sections 16 through 19) defines a ContextObject Format based on XML (eXtensible Markup Language). XML is a markup language from the World Wide Web Consortium [1]. Like HTML, it uses tags to describe text and data in documents, but XML provides the capability of creating customized tags. XML Documents are widely used in the exchange of structured text and data between computer applications. With the XML ContextObject Format, ContextObjects can convey greater detail, which Resolvers can use to provide more appropriate services.

Section 16 describes and registers the following instances of core components necessary to define the XML ContextObject Format: the XML Serialization, the XML Schema Constraint Language, and Constraint Definitions that define the XML ContextObject Format and illustrate XML Metadata Formats. Sections 17 and 18 apply the XML ContextObject Format to obtain XML ContextObject Representations.

Using the XML ContextObject Format, one or more ContextObjects are represented as an XML Document. This XML Document may be transported by any of the Transports defined in Sections 20 and 21. These Transports and the XML ContextObject Format form the basis for a new Application that makes available the full expressive power of the XML syntax and structure to providers of context-sensitive services for the scholarly-information community. This Application is defined by the Level 2 San Antonio Community Profile; see Section 19 and Appendix D.
16 The XML ContextObject Format

This Section introduces the Format triples of the XML ContextObject Format and the XML Metadata Formats necessary to describe Entities. The Format triples consist of:

- The XML Serialization (Section 16.1)
- The XML Schema Constraint Language (Section 16.2)
- Constraint Definitions expressed in XML Schema document instances that define the structure of the XML ContextObject Format (Sections 16.3.1 and 17) and of XML Metadata Formats (Sections 16.3.2 and 18).

While Sections 16, 17, and 18 introduce, describe, and illustrate these elements, the Registry is the authoritative source for their complete specification.

16.1 The XML Serialization

Registry Identifier info:ofi/fmt:xml

The XML Serialization is XML as defined in the Extensible Markup Language (XML) 1.0 (Third Edition) [1].

16.2 XML Schema as a Constraint Language

Registry Identifier info:ofi/fmt:xml:xsd

This Standard makes use of XML Schema [3] [4] to specify constraints and structures for resource descriptions expressed in the XML Serialization. In the XML Serialization, descriptions are expressed as XML Documents that conform to a registered XML Schema.

16.3 Constraint Definitions in the XML ContextObject Format

The main Constraint Definition associated with the XML Serialization and the XML Schema Constraint Language is the XML ContextObject Format. This Format defines the Representation of a ContextObject as an XML Document.

In addition, there are Constraint Definitions known as XML Metadata Formats that define the Representation of Entities of ContextObjects as XML Documents or XML Document fragments. These Representations may be used for both By-Value and/or By-Reference Metadata Descriptors.

In the Registry, a Constraint Definition for a Format expressed in the XML Schema Constraint Language is described by the following metadata:

- **dc:title**: the title of the Format
- **dc:creator**: the name of the community that defined the Format
- **dc:description**: a brief description of the Format
- **dc:identifier**: a locator of the XML Schema that defines the Format
- **dcterms:created**: the date when the Format was created
- **dcterms:modified**: the date when the Format was modified

XML Schemas are primarily intended for use by XML parsing and validation software.
### 16.3.1 XML Schema Constraint Definition for the XML ContextObject Format

*Registry identifier: info:ofi/fmt:xml:xsd:ctx*

Table 18 is the XML Schema for ContextObjects. This XML Schema Constraint Definition, also available in the Registry at <http://www.openurl.info/registry/docs/info:ofi/fmt:xml:xsd:ctx>, allows for the definition of multiple ContextObjects in one XML Document.

<table>
<thead>
<tr>
<th>Table 18: XML Schema Constraint Definition of XML ContextObject Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">dc:title</a>XML ContextObject Format&lt;/dc:title&gt;</td>
</tr>
<tr>
<td><a href="">dc:creator</a>NISO Committee AX, OpenURL Standard Committee&lt;/dc:creator&gt;</td>
</tr>
<tr>
<td><a href="">dc:creator</a>Herbert Van de Sompel&lt;/dc:creator&gt;</td>
</tr>
<tr>
<td><a href="">dc:description</a>This XML Schema defines a format to express one or more ContextObjects as an XML document.&lt;/dc:description&gt;</td>
</tr>
<tr>
<td><a href="">dcterms:created</a>2004-01-01&lt;/dcterms:created&gt;</td>
</tr>
<tr>
<td>&lt;/appinfo&gt;</td>
</tr>
<tr>
<td>&lt;element name=&quot;context-objects&quot;&gt;</td>
</tr>
<tr>
<td>&lt;annotation&gt;</td>
</tr>
<tr>
<td>&lt;documentation&gt;The 'context-objects' element is a wrapper holding one or more autonomous XML ContextObjects.&lt;/documentation&gt;</td>
</tr>
<tr>
<td>&lt;/annotation&gt;</td>
</tr>
<tr>
<td>&lt;complexType&gt;</td>
</tr>
<tr>
<td>&lt;complexContent&gt;</td>
</tr>
<tr>
<td>&lt;extension base=&quot;ctx:context-objects-type&quot;/&gt;</td>
</tr>
<tr>
<td>&lt;/complexContent&gt;</td>
</tr>
<tr>
<td>&lt;/complexType&gt;</td>
</tr>
<tr>
<td>&lt;/element&gt;</td>
</tr>
</tbody>
</table>
<complexType name="context-objects-type">
  <annotation>
    <documentation>The 'context-objects' element is a wrapper holding one or more autonomous ContextObjects.</documentation>
  </annotation>
  <sequence>
    <element name="administration" type="ctx:administration-type" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="ctx:context-object"/>
  </sequence>
</complexType>
<element name="context-object">
  <annotation>
    <documentation>The ContextObject is an information construct to represent an Entity that is referenced in a networked environment (the Referent) along with Entities that constitute the context in which the Referent is referenced. In the ContextObject, the Entities that describe the context are: the ReferringEntity, the Requester, the Resolver, the ServiceType, the Referrer. The ContextObject is represented by the 'context-object' element in this XML ContextObject Format.</documentation>
  </annotation>
  <complexType>
    <complexContent>
      <extension base="ctx:context-object-type"/>
    </complexContent>
  </complexType>
</element>
<complexType name="context-object-type">
  <annotation>
    <documentation>The ContextObject represented using the XML ContextObject Format contains descriptions of the following Entities: (1) exactly one Referent, (2) zero or one ReferringEntity, (3) zero or one Requester, (4) zero or more ServiceTypes, (5) zero or more Resolvers, and (6) zero or one Referrer. In the XML ContextObject Format, these Entities are represented by the elements 'referent', 'referring-entity', 'requester', 'service-type', 'resolver', and 'referrer', respectively.</documentation>
    <documentation>Each ContextObject has the following optional administrative attributes: (1) 'version' attribute - version of the ContextObject - fixed value Z39.88-2004 (optional), (2) 'identifier' attribute - identifier of the ContextObject (optional), and (3) 'timestamp' attribute - date/time of creation of the ContextObject (optional).</documentation>
    <documentation>The 'context-object' element has an optional administrative child element to hold Community-specific
administrative data. The name of that element is 'administration'.

<sequence>
  <element name="administration" type="ctx:administration-type" minOccurs="0"/>
  <element name="referent" type="ctx:descriptor-type"/>
  <element name="referring-entity" type="ctx:descriptor-type" minOccurs="0"/>
  <element name="requester" type="ctx:descriptor-type" minOccurs="0" maxOccurs="unbounded"/>
  <element name="service-type" type="ctx:descriptor-type" minOccurs="0" maxOccurs="unbounded"/>
  <element name="resolver" type="ctx:descriptor-type" minOccurs="0" maxOccurs="unbounded"/>
  <element name="referrer" type="ctx:descriptor-type" minOccurs="0"/>
</sequence>

<attribute name="version" use="optional" fixed="Z39.88-2004"/>
<attribute name="identifier" type="string" use="optional"/>
<attribute name="timestamp" type="ctx:utc-datetime-type" use="optional"/>
</complexType>
<br />
<complexType name="metadata-by-val-type">
  
  <annotation>
    <documentation>By-Value Metadata is provided through an XML description embedded in the ContextObject.</documentation>  
    <documentation>The By-Value Metadata is provided as the combination of (1) a 'format' element, which identifies the Metadata Format of the By-Value Metadata, and (2) a 'metadata' element in which the metadata corresponding to the identified Metadata Format is contained.</documentation>
  </annotation>
  
  <sequence>
    <element name="format" type="ctx:metadata-identifier-type"/>
    <element name="metadata" type="ctx:metadata-type"/>
  </sequence>
</complexType>

<complexType name="metadata-type">
  
  <sequence>
    <any namespace="##other" processContents="lax"/>
  </sequence>
</complexType>

<complexType name="metadata-by-ref-type">
  
  <annotation>
    <documentation>By-Reference Metadata is provided by means of the network-location of a document that contains the metadata.</documentation>  
    <documentation>By-Reference Metadata is provided as the combination of (1) a 'format' element, which identifies the Metadata Format of the By-Reference Metadata, and (2) a 'location' element that specifies the network-location of the By-Reference Metadata</documentation>
  </annotation>
  
  <sequence>
    <element name="format" type="ctx:metadata-identifier-type"/>
    <element name="location" type="ctx:network-location-type"/>
  </sequence>
</complexType>

<complexType name="private-data-type">
  
  <annotation>
    <documentation>Private Data is provided through an XML description that declares its XML Namespace URI and schemaLocation.</documentation>
  </annotation>
  
  <sequence>
  </sequence>
</complexType>
16.3.2 XML Schema Constraint Definitions for XML Metadata Formats

The XML Schema Constraint Language is also used to define XML Metadata Formats. Table 28 in Appendix D contains the list of XML Metadata Formats that are in the initial Registry. For each of these XML Metadata Formats, the Registry at <http://www.openurl.info/registry> contains a complete and authoritative Constraint Definition.
Table 19 is a **Constraint Definition** in the form of an XML Schema to describe a class of Entities of the type “journal” (which includes “journal article”). This Constraint Definition is registered and available in the Registry at <http://www.openurl.info/registry/docs/info:ofi/fmt:xml:xsd:journal>. Therefore, it defines a registered XML Metadata Format, which can be identified using the “info” URI scheme.

In principle, registration is **optional**, because an XML Serialization may use any XML Schema as a Metadata Format by identifying the XML Schema with its “http” URI in the format element. In practice, however, it is unlikely that Resolvers will be able to make sense of the metadata if an XML Metadata Format is not registered.

**Table 19: XML Schema Constraint Definition of XML Metadata Format for “journal”**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="info:ofi/fmt:xml:xsd:journal"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jo="info:ofi/fmt:xml:xsd:journal"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
  <xs:annotation>
  </xs:annotation>
  <xs:appinfo xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/">
    <dc:title>XML Format article</dc:title>
    <dc:creator>Committee NISO AX, OpenURL Standard Committee</dc:creator>
    <dc:description>This XML Schema defines a format to express properties of serial publications and their component parts</dc:description>
    <dcterms:created>2003-09-27</dcterms:created>
  </xs:appinfo>
  <xs:element name="journal" type="jo:journalType">
    <xs:annotation>
      <xs:documentation>The root element "journal" contains child elements that are used to express properties of serial publications</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="journalType">
    <xs:sequence>
      <xs:element name="authors" type="jo:authorType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
<xs:annotation><xs:documentation>The "authors" element contains child elements that are used to express authorship of an individual article in a serial publication. The "authors" element is not repeatable, it contains all authors, and allows for the indication of the position of the author in the publication's list of authors</xs:documentation></xs:annotation>

<xs:element name="atitle" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Article title</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="title" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Journal title. Provided for compatibility with OpenURL version 0.1. Usage of the "jtitle" element is preferred</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="jtitle" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Journal title. Use the most complete title available, e.g. "journal of the american medical association". Abbreviated titles, when known, are provided in the "stitle" element. Journal title information can also be provided in the "title" element, which is provided for compatibility with OpenURL version 0.1</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="stitle" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Abbreviated or short journal title. This is used for journal title abbreviations, e.g. "J Am Med Assn"</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="date" type="jo:dateType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Date of publication in ISO 8601 form YYYY, YYYY-MM or YYYY-MM-DD</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="chron" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Indications of chronology in a non ISO8601 form (like "Spring" or "1st quarter") should be carried in this element; the element content is not normalized. Where numeric ISO8601 dates are also available, they should be provided in the "date" element. As such, a recorded date of publication of "Spring, 1992" becomes "date=1992" and "chron=spring". Chronology information can also be provided in the "ssn" and "quarter" elements</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="ssn" type="jo:ssnType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Season (chronology). Legitimate values are "spring", "summer", "fall", "winter"</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="quarter" type="jo:quarterType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Quarter (chronology). Legitimate values are "1", "2", "3", "4"</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="volume" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Volume designation. Volume is usually expressed as a number but could be roman numerals or non-numeric, e.g. "124", or "VI","4"</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="part" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Part can be a special subdivision of a volume or it can be the highest level division of the journal. Parts are often designated with letters or names, e.g. "B", "Supplement"</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="issue" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>This is the designation of the published issue of a journal, corresponding to the actual
physical piece in most cases. While usually numeric, it could be non-numeric. Note that some publications use chronology in the place of enumeration, i.e. Spring, 1998.
ISSN for electronic version of the journal. Although there is no distinction by format in the assignment of ISSNs, some bibliographic services now carry both the ISSN for the paper version and a separate ISSN for the electronic version. This data element is included here to allow expression of both types of ISSN numbers.

International Standard Book Number (ISBN). The ISBN is usually presented as 9 digits plus a final check digit (which may be "X"), e.g. "057117678X". ISBN numbers may contain hyphens, e.g. "1-878067-73-7".

Serial Item and Contribution Identifier (SICI).

Genre of the document. Legitimate values for the "genre" element are: (1) "journal": for a serial publication issued in successive parts (2) "issue": for one instance of the serial publication (3) "article": for a document published in a journal. (4) "conference": for a record of a conference that includes one or more conference papers and that is published as an issue of a journal or serial publication (5) "proceeding": for a single conference presentation published in a journal or serial publication (6) "preprint": for an individual paper.
or report published in paper or electronically prior to its publication in a journal or serial (7) "unknown": use when the genre is unknown.

```xml
<xs:complexType name="authorType">
  <xs:choice maxOccurs="unbounded">
    <xs:element name="author" type="jo:detailedAuthorType" minOccurs="0"/>
    <xs:element name="au" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The author's full name, i.e. "Smith, Fred M", "Harry S. Truman"</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="aucorp" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Organization or corporation that is the author or creator of the book, i.e. "Mellon Foundation"</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:choice>
  <xs:attribute name="rank" type="xs:positiveInteger" use="optional">
    <xs:annotation>
      <xs:documentation>An integer indicating the position of the author in the publication's list of authors, e.g. "1" for first author, "2" for second author, etc.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
```

```xml
<xs:complexType name="detailedAuthorType">
  <xs:sequence>
    <xs:element name="aulast" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The author's family name. This may be more than one word. In many citations, the author's family name is recorded first and is followed by a comma, i.e. Smith, Fred James is recorded as "aulast=smith"</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="aufirst" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The author's first name. This may be more than one word. In many citations, the author's first name is recorded first and is followed by a comma, i.e. Smith, Fred James is recorded as "aufirst=fred"</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```
The author's given name or names or initials. This data element may contain multiple words and punctuation, i.e. "Fred F", "Fred James".

The author's first and middle initials.

The author's first initial.

The author's middle initial.

The author's name suffix. Qualifiers on an author's name such as "Jr.", "III" are entered here. i.e. Smith, Fred Jr. is recorded as "ausuffix=jr".

The author's first initial.

The author's middle initial.

The author's name suffix. Qualifiers on an author's name such as "Jr.", "III" are entered here. i.e. Smith, Fred Jr. is recorded as "ausuffix=jr".

The author's first initial.

The author's middle initial.

The author's name suffix. Qualifiers on an author's name such as "Jr.", "III" are entered here. i.e. Smith, Fred Jr. is recorded as "ausuffix=jr".

The author's first initial.

The author's middle initial.

The author's name suffix. Qualifiers on an author's name such as "Jr.", "III" are entered here. i.e. Smith, Fred Jr. is recorded as "ausuffix=jr".

The author's first initial.

The author's middle initial.

The author's name suffix. Qualifiers on an author's name such as "Jr.", "III" are entered here. i.e. Smith, Fred Jr. is recorded as "ausuffix=jr".

The author's first initial.

The author's middle initial.

The author's name suffix. Qualifiers on an author's name such as "Jr.", "III" are entered here. i.e. Smith, Fred Jr. is recorded as "ausuffix=jr".
17 XML ContextObject Representations

Using the XML Format, one or more ContextObjects are expressed in an XML Document. The XML ContextObject Format triple is as follows:

- The XML Serialization (see Section 16.1), recorded in the Registry under
  [Registry Identifier info:ofi/fmt:xml]
- The XML Schema Constrain Language (see Section 16.2), recorded in the Registry under
  [Registry Identifier info:ofi/fmt:xml:xsd]
- An XML Schema that specifies the actual constraints and structure for the XML ContextObject Format, recorded in the Registry under
  [Registry Identifier: info:ofi/fmt:xml:xsd:ctx]

Example 17 displays an XML ContextObject Representation that includes administrative data elements, two Identifier Descriptors to describe the Referent, Identifier Descriptors for the ReferringEntity, Requester, and Referrer. (Example 19 will show the use of By-Value Metadata in an XML ContextObject Representation.)
Example 17: XML ContextObject Representation

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ctx:context-objects
 xmlns:ctx="info:ofi/fmt:xml:xsd:ctx"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="info:ofi/fmt:xml:xsd:ctx
<ctx:context-object
 timestamp="2002-03-20T08:55:12Z"
 version="Z39.88-2004"
 identifier="456">
<ctx:referent>
<ctx:identifier>
 info:doi/10.1126/science.275.5304.1320
</ctx:identifier>
</ctx:referent>
<ctx:referring-entity>
<ctx:identifier>
 info:doi/10.1006/mthe.2000.0239
</ctx:identifier>
</ctx:referring-entity>
<ctx:requester>
<ctx:identifier>
 mailto:jane.doe@caltech.edu
</ctx:identifier>
</ctx:requester>
<ctx:referrer>
<ctx:identifier>
 info:sid/elsevier.com:ScienceDirect
</ctx:identifier>
</ctx:referrer>
</ctx:context-object>
</ctx:context-objects>
```

17.1 Cardinality Constraints on the XML ContextObject Format

The XML ContextObject Format restricts the number of Entities that may be present in each ContextObject, the number of Descriptors that may be used to describe Entities, and the number of ContextObjects that may be bundled in a single XML Representation. These constraints are specified and summarized in Table 20 (compare this with the fundamental restrictions of Table 1).
Table 20: XML ContextObject Format – Cardinality Constraints

<table>
<thead>
<tr>
<th>Entities per ContextObject</th>
<th>Number</th>
<th>Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identifier</td>
</tr>
<tr>
<td>Referent</td>
<td>1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Requester</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ServiceType</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Resolver</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Referrer</td>
<td>≤ 1</td>
<td>≥ 0</td>
</tr>
<tr>
<td>ContextObjects</td>
<td>≥ 1</td>
<td>N/A</td>
</tr>
</tbody>
</table>

17.2 Entity and Descriptor Elements in the XML ContextObject Format

Table 21 gives an overview of the XML ContextObject Format by listing the appropriate XML element for each combination of Entity and Descriptor. The Table lists those elements as XPath expressions [2]. These XPath expressions are relative to the ctx:context-object element, not to the ctx:context-objects element.

For example:

- The XPath expression //referent/identifier of Table 21 addresses XML elements that describe a Referent by means of Identifier Descriptors. In the XML ContextObject of Example 17, two XML elements match this XPath expression. Their respective content is info:doi/10.1126/science.275.5304.1320 and info:pmid/9036860.

- The XPath expression //referent/metadata-by-val/format of Table 21 addresses XML elements that identify Metadata Formats used for By-Value Metadata of a Referent. In the XML ContextObject of Example 22, one XML element matches this XPath expression. Its content is info:ofi/fmt:xml:xsd:journal. This particular Metadata Format is registered, as can be seen from its identifier in the info:ofi/ namespace.
### Table 21: XML ContextObject Format – Entities and Descriptors

<table>
<thead>
<tr>
<th>Entities</th>
<th>Nr.</th>
<th>Descriptors</th>
<th>Identifier</th>
<th>By-Value Metadata</th>
<th>By-Reference Metadata</th>
<th>Private Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referent</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>//referent/identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Referring-Entity</td>
<td>≤ 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>//referring-entity/identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requester</td>
<td>≤ 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>//requester/identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service-Type</td>
<td>≥ 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>//service-type/identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolver</td>
<td>≥ 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>//resolver/identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Referrer</td>
<td>≥ 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>//referrer/identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
17.3 Administrative Elements and Attributes in the XML ContextObject Format

Table 22 lists the administrative elements and attributes of an XML ContextObject Representation. The XML ContextObject Format provides three ways to specify administrative information:

- Three optional attributes of the context-object element
- Community-specific administrative data contained in the administration child element of the context-object element
- Community-specific administrative data contained in the administration child element of the context-objects element

<table>
<thead>
<tr>
<th>Item</th>
<th>Number</th>
<th>Element</th>
<th>Attribute</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration</td>
<td>≤ 1</td>
<td>//context-objects</td>
<td>attribute</td>
<td>Community-defined</td>
</tr>
<tr>
<td>Administration</td>
<td>≤ 1</td>
<td>//context-object/</td>
<td>attribute</td>
<td>Community-defined</td>
</tr>
<tr>
<td>Identifier</td>
<td>≤ 1</td>
<td>//context-object@</td>
<td>identifier</td>
<td>Identifier of ContextObject</td>
</tr>
<tr>
<td>Timestamp</td>
<td>≤ 1</td>
<td>//context-object@</td>
<td>timestamp</td>
<td>ISO8601datetime specifying the time of creation of the ContextObject</td>
</tr>
</tbody>
</table>

17.4 Character Encoding in the XML ContextObject Format

Character Encoding in the XML ContextObject Format follows the specifications provided by XML [1]. XML ContextObject Representations must use the UTF-8 encoding of Unicode. As is standard in XML Documents, character-encoding information is provided by the value of the encoding declaration in the XML declaration. Because UTF-8 is default for XML, the encoding declaration may be omitted. If the encoding declaration is present, it must specify UTF-8.

18 Entity Descriptors in the XML ContextObject Format

A Descriptor specifies information about an Entity. There are four types of Descriptors that may be used in the XML ContextObject Format: Identifier, By-Value Metadata, By-Reference Metadata, and Private Data.
18.1 Identifier Descriptors

An Identifier Descriptor specifies an Entity by means of a Uniform Resource Identifier (URI). This URI may be associated with the Entity itself or with metadata for the Entity. As described in Section 17.2, Identifier Descriptors in the XML ContextObject Format are represented using identifier elements.

Example 18 shows Identifier Descriptors for a Referent, a Requester, and a Resolver.

Example 18: Identifier Descriptors in an XML ContextObject Representation

```xml
<referent>
  <identifier>info:doi/10.1126/science.275.5304.1320</identifier>
</referent>

<referent>
  <identifier>info:pmid/9036860</identifier>
</referent>

<requester>
  <identifier>mailto:jane.doe@caltech.edu</identifier>
</requester>

<resolver>
  <identifier>http://links.caltech.edu/menu</identifier>
</resolver>
```

18.2 By-Value and By-Reference Metadata Descriptors

A Metadata Format provides a concrete set of descriptive elements for the purpose of representing an Entity. For compatibility, Metadata Formats and the ContextObject Format must be based on the same Serialization and Constraint Language. This compatibility rule is waived for By-Reference Metadata, provided the Metadata Format is registered (see Sections 9.2 and 18.2.1).

Metadata Formats used in the OpenURL Framework may be registered. Unregistered Metadata Formats must meet the requirements described in Section 9.2.

18.2.1 Rules Guiding By-Value and By-Reference Metadata Descriptors

The general rules for Metadata Formats are given in Section 9.2. This Section gives the rules for creating By-Value Metadata and By-Reference Metadata in the XML ContextObject Format. The XML ContextObject Format accommodates both registered and unregistered Metadata Formats.

- Registered Metadata Formats
  - Registered Metadata Formats must be identified by means of the Registry Identifier of the Metadata Format. The Registry maintains a one-to-one correspondence between the definition of a Metadata Format and its Registry Identifier. The identification of the Metadata Format must be provided as the content of the format element.
  - The corresponding By-Value Metadata Descriptor must use the XML Serialization and must conform to an XML Schema. This XML Schema must be in the Registry and correspond uniquely with the Registry Identifier used to identify the Metadata Format. The Registry Identifier of the Metadata Format must be of the form: info:ofi/fmt:xml:xsd:format_name.
  - The corresponding By-Reference Metadata Descriptor must be an instance document that conforms to the Metadata Format identified by the Registry Identifier. Because the Metadata Format is registered, the By-Reference Metadata Descriptor may use any registered Serialization, and the Metadata Format to which it conforms may use any
registered Constraint Language. The By-Reference Metadata Descriptor is not limited to the XML Serialization or the XML Schema Constraint Language.

- Unregistered Metadata Formats
  - Unregistered Metadata Formats **must** be identified by means of a URL that specifies the network location of the XML Schema that defines the XML Metadata Format. The identification of the Metadata Format **must** be provided as the content of the `format` element. For example, a Metadata Format could be identified as: http://www.example.net/x-service.xsd.
  - The corresponding By-Value Metadata or By-Reference Metadata Descriptor **must** use the XML Serialization: it **must** be an XML Document that conforms to the XML Schema at the network location specified by the aforementioned URL.

18.2.2 By-Value Metadata Descriptors

An XML By-Value Metadata Descriptor consists of a `metadata-by-val` element containing

- a `format` element containing the format identifier and
- a `metadata` element containing XML metadata conforming to the XML Schema identified in the `format` element.

**Example 19: Referent with a By-Value Metadata Descriptor**

```xml
<ctx:referent>
  <ctx:metadata-by-val>
    <ctx:metadata>
          xml:base="http://www.example.net" xml:lang="en"
          xml:space="preserve">
        <rft:authors>
          <rft:author>
            <rft:aulast>Bergelson</rft:aulast>
            <rft:auinit>J</rft:auinit>
          </rft:author>
        </rft:authors>
        <rft:atitle>Isolation of a common receptor for coxsackie B viruses and adenoviruses 2 and 5</rft:atitle>
        <rft:jtitle>Science</rft:jtitle>
        <rft:date>1997</rft:date>
        <rft:volume>275</rft:volume>
        <rft:spage>1320</rft:spage>
        <rft:epage>1323</rft:epage>
      </rft:journal>
    </ctx:metadata>
  </ctx:metadata-by-val>
</ctx:referent>
```
Example 19 shows a By-Value Metadata Descriptor for a Referent, an article in a journal.

- The XML Metadata Format is identified by info:ofi/fmt:kev:mtx:journal. White space is trimmed from the element data.
- The XML metadata in the ctx:metadata element conforms to the specified XML Schema.
- The requirement to provide a Format Identifier in the format element is separate from and independent of the requirement made by XML Schema for elements to declare their XML namespace.

18.2.3 By-Reference Metadata Descriptors

An XML By-Reference Metadata Descriptor consists of a metadata-by-ref element containing

- a format element that specifies the URI of a Metadata Format and
- a location element that specifies the URL of By-Reference Metadata that conform to the Constraint Definition identified in the format element.

Example 20: Requester with a By-Reference Metadata Descriptor

```xml
<ctx:requester>
  <ctx:metadata-by-ref>
    <ctx:format>http://my.example.org/eduperson.xsd</ctx:format>
    <ctx:location>ldap://ldap.caltech.edu:389/janed</ctx:location>
  </ctx:metadata-by-ref>
</ctx:requester>
```

Example 20 shows a By-Reference Metadata Descriptor for a Requester. In this example, the Requester is Jane Doe, a student at Caltech, identified by her LDAP record.

- The Metadata Format is identified by http://my.example.org/eduperson.xsd. Because this Metadata Format is not registered, this must be the URL of an XML Schema.

18.3 Private Data Descriptors

A Private Data Descriptor specifies information about the Entity using a method not defined in this Standard. This Standard does not provide any global mechanisms to interpret Private Data. Instead, it is assumed that the Resolver and the Referrer have a common understanding, based on a tacit or explicit bilateral agreement. To make it possible for the Resolver to interpret Private Data, a ContextObject that contains Private Data must identify the Referrer that created it.

Example 21: ReferringEntity with a Private Data Descriptor

```xml
<referring-entity>
  <private-data>
    <x:citdata xmlns:x="http://example.org/x" cites="8" citedby="12"/>
  </private-data>
</referring-entity>
```

In Example 21, the information in the referring-entity element is XML from an external unidentified scheme. The meaning of data in the private-data element is defined by the Referrer, which is identified in the referrer element (not shown in the example).
18.4 Example of an XML ContextObject Representation

Example 22 shows an XML Representation of a ContextObject that combines several of the previous examples.

The first line is a common XML introduction that specifies the XML version number and the XML character encoding.

The context-objects element is an optional container to hold multiple context-object elements. The context-objects element includes appropriate XML namespace declarations that indicate to XML processors how to validate the XML Document.

In this example, the context-objects element holds only one context-object element. The attributes of the context-object element specify the administrative data of the ContextObject: time of creation, version of this Standard, and an optional identifier for the ContextObject Representation. The optional identifier might be used to assist in the retrieval of ContextObject Representations.

The Referent is described in the referent element by means of a By-Value Metadata Descriptor in the metadata-by-val element. The format element specifies the XML Metadata Format for a journal (info:ofi/fmt:xml:xsd:journal). This is followed by the metadata element, which holds a journal element that contains the actual metadata for the Referent in the specified Metadata Format. The journal element holds an authors element that lists the authors, an atitle element to specify the title of the article, a jtitle element to specify the title of the journal, a date element to specify the date of publication, a volume element, a spage element for the starting page, and an epage element for the end page of the article.

A ReferringEntity is described in the referring-entity element by means of an Identifier Descriptor in the identifier element.

The Requester is described in the requester element by means of a By-Reference Metadata Descriptor in the metadata-by-ref element. The latter contains a format element and a location element to specify, respectively, the Metadata Format and the location of the actual metadata. The Metadata Format for the Requester is identified by the URL http://my.example.org/eduperson.xsd. For the ContextObject Representation to be valid, the LDAP URL ldap://ldap.caltech.edu:389/janed must point to an XML Document that conforms to the XML Schema located at the URL of the Metadata Format (http://my.example.org/eduperson.xsd).

Finally, the Referrer is described in the referrer element by means of an Identifier Descriptor in the identifier element. This element contains an identifier from the info:sid/ namespace used to identify sources of information.

Example 22: XML ContextObject Representation

```xml
<?xml version="1.0" encoding="UTF-8"?>
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
<ctx:context-object timestamp="2002-06-14T12:13:00Z" version="Z39.88-2004" identifier="125">
<ctx:referent>
<ctx:metadata>
<rft:journal
xmlns:rft="info:ofi/fmt:xml:xsd:journal"
xsi:schemaLocation="info:ofi/fmt:xml:xsd:journal
```
19 XML-Based Community Profiles

A Community Profile lists a selection of Registry entries. This selection specifies the ContextObject Format, the Metadata Format(s), and the Transport(s) that form the core properties of an OpenURL Application. Further information on the creation of Community Profiles is found in Section 11.

A Resolver that conforms to the XML ContextObject Format must process all items that conform to Registry entries specified in a Community Profile using the XML ContextObject Format. Communities may define additional conformance rules in their Community Profiles.

Appendix D describes the Level 2 San Antonio Community Profile, which is an example of a Community Profile based on the XML ContextObject Format. This Community Profile was developed by NISO Committee AX for the scholarly-information community. In the remainder of this Standard, this Community Profile will be referred to as the SAP2 Community Profile. Its Registry Identifier is info:ofi/pro:sap2-2004.
Other communities are encouraged to use the XML ContextObject Format to deploy their own OpenURL Framework Applications. As specified in Section 11, each Application must be defined in a Community Profile. A straightforward way to deploy XML-based Applications is to modify the SAP2 Community Profile to the needs of new communities.
The OpenURL Framework for Context-Sensitive Services

Part 4: OpenURL Transports

Part 1 (Sections 5 through 11) defines the core components of the OpenURL Framework: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, Transports, and Community Profiles.

Parts 2, 3, and 4 (Sections 12 through 22) define instances of these core components that illustrate the abstract concepts of Part 1. These instances form the initial content of the Registry. Each instance is described, given a Registry Identifier, and entered into the Registry at <http://www.openurl.info/registry/>. The initial Registry launches two Applications of the OpenURL Framework Standard intended for the scholarly-information community. The first Application provides a migration path from OpenURL 0.1 to the OpenURL Framework Standard. The second Application provides a path for future growth by harnessing the full expressive power of XML.

Part 2 defines a ContextObject Format inspired by the query string of the HTTP(S) GET request as specified in OpenURL 0.1. Part 3 defines a ContextObject Format based on XML.

Part 4 (Sections 20, 21, and 22) defines six methods to convey ContextObject Representations over a network. All six methods use the HTTP and HTTPS protocols defined in IETF RFC 2616 [14]. They are collectively called OpenURL Transports. Four of these Transports are generic and may be used with any ContextObject Format. Two of the Transports are developed specifically for the ContextObject Format defined in Part 2 to provide a migration path from OpenURL 0.1 to this Standard. Communities may use these transports in new Applications, and/or they may choose to create and register new instances of Transports. For example, a community could consider defining a SOAP-based Transport for XML ContextObject Representations.

Section 20 specifies By-Reference OpenURL Transports, which use HTTP(S) as the network protocol to transport network locations of ContextObject Representations.

Section 21 specifies the By-Value OpenURL Transports, which use HTTP(S) as the network protocol to transport ContextObject Representations.

By-Reference and By-Value OpenURL Transports are generic: they may be used to transport KEV ContextObject Representations, XML ContextObject Representations, and ContextObject Representations based on other, yet-to-be-registered, ContextObject Formats.

Section 22 specifies Inline OpenURL Transports, which use HTTP(S) as the network protocol to transport KEV ContextObject Representations carried as KEV pairs in the HTTP(S) query string.

Inline OpenURL Transports may only be used to transport KEV ContextObject Representations. Inline OpenURL Transports must not be used to transport ContextObject Representations in any other ContextObject Format. Inline OpenURL Transports are primarily introduced to provide a migration path from the OpenURL 0.1 specification to this Standard. The Implementation Guidelines available at <http://www.openurl.info/registry/docs/implementation_guidelines/> provide the details of this upgrade path for the scholarly-information community.

The OpenURL 0.1 specification, which is not a Transport as defined by this Standard, is available in the Registry at <http://www.openurl.info/registry/docs/pdf/openurl-01.pdf>.

Note: The terms “by reference” and “by value” refer to basic programming techniques that are widely applicable. In this Standard, both techniques are used independently in two contexts: Transports and Metadata. Both By-Reference and By-Value Transports may transport ContextObject Representations that contain By-Reference and/or By-Value Metadata.
20 By-Reference OpenURL Transports

A By-Reference OpenURL Transport transports the network location of a ContextObject Representation. The Representation itself is not transported, but resides at a network location. Depending on the constraints of the ContextObject Format, the Representation stored at a network location may contain the description of one or more ContextObjects. The By-Reference OpenURL Transport may be used for a ContextObject Representation that conforms to any registered ContextObject Format.

The By-Reference OpenURL Transport uses the HTTP network protocol or its secure sibling, HTTPS. The Registry Identifiers for these Transports are:

- By-Reference OpenURL Transport over HTTPS: `info:ofi/tsp:https:openurl-by-ref`

This Section describes both Transports, which are identical except for their use of HTTP or HTTPS as the respective network protocol.

For each transportation via the By-Reference OpenURL Transport, a base URL specifies the "Internet host and port, and path" of the target of the transportation, an HTTP(S)-based service called a Resolver.

A By-Reference OpenURL Transport may convey the network location of a ContextObject Representation via HTTP(S) GET or HTTP(S) POST.

Appendix E provides implementation guidelines for the By-Reference OpenURL Transports.

20.1 OpenURL Keys in By-Reference OpenURL Transports

A By-Reference OpenURL Transport uses KEV pairs with the following keys, either in the query string of an HTTP(S) GET request or in the message body of an HTTP(S) POST:

**url_ver**: OpenURL signature

- **Required**
- Maximum occurrence: 1
- Format: fixed value is the case-sensitive character string “Z39.88-2004”
- Character set and character encoding: value is US-ASCII
- Example: `url_ver=Z39.88-2004`

**url_tim**: Datetime of the creation of the OpenURL

- **Optional**
- Maximum occurrence: 1
- Format: ISO8601-conformant datetime in the YYYY-MM-DD or YYYY-MM-DDTHH:MM:SSZ representation
- Character set and character encoding: value is US-ASCII and may need URL-encoding
- Example (not URL-encoded for readability): `url_tim=2002-08-16T17:23:45Z`
url_ctx_fmt: Registry Identifier of the ContextObject Format of the referenced ContextObject Representation

- **Required**
- Maximum occurrence: 1
- Format: Registry Identifiers for ContextObject Formats (see Section 6.2)
- Character set and character encoding: value is US-ASCII and *may* need URL-encoding

url_ctx_ref: Network location of the ContextObject Representation

- **Required**
- Maximum occurrence: 1
- Dependency: requires `url_ctx_fmt`
- Format: network location (a URL)
- Character set and character encoding: value is US-ASCII and *may* need URL-encoding
- Example (not URL-encoded for readability):
  ```
  url_ctx_ref=http://www.example.org/temp/12587.xml
  ```

A foreign key in the *By-Reference OpenURL Transport* is any key that is not an OpenURL key. Foreign keys *may* be used in a *By-Reference OpenURL Transport*, but their meaning is not defined by the Transport. Resolvers *may* ignore KEV pairs with foreign keys.

### 20.2 By-Reference OpenURL Transports using HTTP(S) GET

In the HTTP(S) GET mode of the *By-Reference OpenURL Transport*, KEV pairs described in Section 20.1 are concatenated with the ampersand character (`&`) to form the query string of an HTTP(S) GET request. The resulting query string is appended to the base URL of the target Resolver, and separated from it by a question mark (`?`). As specified by the syntax rules for URIs [6], the query string following this question mark *must* be URL-encoded.

**Example 23: By-Reference OpenURL Transport using HTTP GET**

<table>
<thead>
<tr>
<th>Formatted for readability:</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.net/menu">http://www.example.net/menu</a>?</td>
</tr>
<tr>
<td>url_ver = Z39.88-2004</td>
</tr>
<tr>
<td>&amp; url_tim = 2002-08-16T17:23:45Z</td>
</tr>
<tr>
<td>&amp; url_ctx_fmt = info:ofi/fmt:kev:mtx:ctx</td>
</tr>
<tr>
<td>&amp; url_ctx_ref = <a href="http://www.example.org/temp/12587.txt">http://www.example.org/temp/12587.txt</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL-encoded:</th>
</tr>
</thead>
</table>

Example 23 illustrates the HTTP GET method of the *By-Reference OpenURL Transport* to transport the network location of a KEV ContextObject Representation. The first part is formatted for readability: the query string is not URL encoded, white space is introduced, and KEV pairs are on separate lines. The second part is formatted for actual use with a URL-encoded query string.

The base URL of the Transport (the network location of the Resolver) is http://www.example.net/menu. The value assigned to the `url_ctx_fmt` key is
info:ofi/fmt:kev:mtx:ctx. This declares that the referenced ContextObject Representation is based on the KEV ContextObject Format. The network location of the ContextObject Representation is the value assigned to the url_ctx_ref key: http://www.example.org/temp/12587.txt. This file must contain a set of ampersand-delimited KEV pairs that conform to the KEV ContextObject Format.

20.3 By-Reference OpenURL Transports using HTTP(S) POST

In the HTTP(S) POST mode of the By-Reference OpenURL Transport, the query string specified in Section 20.1 is carried in the message body of the HTTP(S) POST. The Content-Type of the HTTP(S) request must be application/x-www-form-urlencoded. Hence, the message body must be URL-encoded.

Example 24: By-Reference OpenURL Transport using HTTP POST

<table>
<thead>
<tr>
<th>Formatted for readability:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base URL: <a href="http://www.example.net/menu">http://www.example.net/menu</a></td>
</tr>
<tr>
<td>POST <a href="http://www.example.net/menu">http://www.example.net/menu</a> HTTP/1.0</td>
</tr>
<tr>
<td>Content-Length: 161</td>
</tr>
<tr>
<td>Content-Type: application/x-www-form-urlencoded</td>
</tr>
<tr>
<td>url_ver = Z39.88-2004</td>
</tr>
<tr>
<td>&amp; url_tim = 2002-08-16T17:23:45Z</td>
</tr>
<tr>
<td>&amp; url_ctx_fmt = info:ofi/fmt:xml:xsd:ctx</td>
</tr>
<tr>
<td>&amp; url_ctx_ref = <a href="http://www.example.net/temp/12587.xml">http://www.example.net/temp/12587.xml</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL-encoded:</th>
</tr>
</thead>
</table>

Example 24 illustrates the HTTP POST method of the By-Reference OpenURL Transport of the network location of a KEV ContextObject Representation. The first part is formatted for readability, and the second part is URL-encoded query string formatted for actual use. The base URL of the Transport (the network location of the Resolver) is http://www.example.net/menu.

21 By-Value OpenURL Transports

A By-Value OpenURL Transport transports the actual ContextObject Representation, not its network location. Depending on the constraints of the ContextObject Format, the Representation may contain the description of one or more ContextObjects. The By-Value OpenURL Transport may transport a ContextObject Representation that conforms to any registered ContextObject Format.

The By-Value OpenURL Transport uses the HTTP network protocol or its secure sibling, HTTPS. The Registry Identifiers for these Transports are:

- By-Value OpenURL Transport over HTTP: info:ofi/tsp:http:openurl-by-val
- By-Value OpenURL Transport over HTTPS: info:ofi/tsp:https:openurl-by-val

This Section describes both Transports, which are identical except for their use of HTTP or HTTPS as the respective network protocol.
For each transportation via the *By-Value OpenURL Transport*, a base URL specifies the “internet host and port, and path” of the target of the transportation, an HTTP(S)-based service called a *Resolver*.

A *By-Value OpenURL Transport* may convey a *ContextObject Representation* via HTTP(S) GET or HTTP(S) POST.

Appendix E provides implementation guidelines for the *By-Value OpenURL Transports*.

### 21.1 OpenURL Keys in By-Value OpenURL Transports

A *By-Value OpenURL Transport* uses KEV pairs with the following keys, either in the query string of an HTTP(S) GET request or in the message body of an HTTP(S) POST:

- **url_ver**: OpenURL signature
  - **Required**
  - Maximum occurrence: 1
  - Format: fixed value is the case-sensitive character string “Z39.88-2004”
  - Character set and character encoding: value is US-ASCII
  - Example: `url_ver=Z39.88-2004`

- **url_tim**: Datetime of the creation of the OpenURL
  - **Optional**
  - Maximum occurrence: 1
  - Format: ISO8601-conformant datetime, in the YYYY-MM-DD or YYYY-MM-DDTHH:MM:SSZ representation
  - Character set and character encoding: value is US-ASCII and may need URL-encoding
  - Example (not URL-encoded for readability): `url_tim=2002-08-16T17:23:45Z`

- **url_ctx_fmt**: Registry Identifier of the *ContextObject Format* of the transported *ContextObject Representation*
  - **Required**
  - Maximum occurrence: 1
  - Dependency: requires `url_ctx_fmt`
  - Format: Registry Identifiers for *ContextObject Formats* (see Section 6.2)
  - Character set and character encoding: value is US-ASCII and may need URL-encoding

- **url_ctx_val**: The actual *ContextObject Representation* expressed according to a registered *ContextObject Format*
  - **Required**
  - Maximum occurrence: 1
  - Character set and character encoding: The character set and character encoding of the value is the *Character Encoding* applied by the *ContextObject Format* used in the
transported ContextObject Representation. In the KEV ContextObject Format, the default Character Encoding is info:ofi/enc:UTF-8. The ContextObject Representation may specify other Character Encodings in the value associated with the ctx_enc key. However, because values are URL-encoded in the KEV ContextObject Format, the ContextObject Representation provided as the value of the url_ctx_val key must be US-ASCII. When provided on a By-Value OpenURL Transport, the value of the url_ctx_val key may need further URL-encoding.

- Example (not URL-encoded for readability):
  url_ctx_val= rft_id=info:doi/10.1126/science.275.5304.1320

A foreign key in the By-Value OpenURL Transport is any key that is not an OpenURL key. Foreign keys may be used in a By-Value OpenURL Transport, but their meaning is not defined by the Transport. Resolvers may ignore KEV pairs with foreign keys.

21.2 By-Value OpenURL Transports using HTTP(S) GET

In the HTTP(S) GET mode of the By-Value OpenURL Transport, KEV pairs described in Section 21.1 are concatenated with the ampersand character ('&') to form the query string of an HTTP(S) GET request. The resulting query string is appended to the base URL of the target Resolver, and separated from it by a question mark ('?'). As specified by the syntax rules for URIs [6], the query string following this question mark must be URL-encoded.

Example 25: By-Value OpenURL Transport using HTTP GET

<table>
<thead>
<tr>
<th>Formatted for readability:</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.net/menu">http://www.example.net/menu</a>?</td>
</tr>
<tr>
<td>url_ver = Z39.88-2004</td>
</tr>
<tr>
<td>&amp; url_tim = 2002-08-16T17:23:45Z</td>
</tr>
<tr>
<td>&amp; url_ctx_fmt = info:ofi/fmt:kev:mtx:ctx</td>
</tr>
<tr>
<td>&amp; url_ctx_val = rft_id=info:doi/10.1126/science.275.5304.1320</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL-encoded:</th>
</tr>
</thead>
<tbody>
<tr>
<td>url_ver=Z39.88-2004&amp;url_tim=2002-08-16T17%3A23%3A45Z&amp;url_ctx_fmt=info%3Aofi%2Ffmt%3Akev%3Amtx%3Actx&amp;url_ctx_val=rft_id%3Dinfo%253Adoi%252F10.1126%252Fscience.275.5304.1320</td>
</tr>
</tbody>
</table>

Example 25 illustrates the HTTP GET method of the By-Value OpenURL Transport of a KEV ContextObject Representation. The first part is formatted for readability: the query string is not URL encoded, white space is introduced, and KEV pairs are on separate lines. The second part is formatted for actual use with a URL-encoded query string.

The base URL of the Transport (the network location of the Resolver) is http://www.example.net/menu. The value assigned to the url_ctx_fmt key is info:ofi/fmt:kev:mtx:ctx. This declares that the transported ContextObject Representation is based on the KEV ContextObject Format.

The value assigned to the url_ctx_val key is the actual KEV ContextObject Representation. Note how this value is URL-encoded twice. The first URL-encoding is required by the KEV ContextObject Format (see Section 13.4). It encodes the values assigned to the keys. The second URL-encoding is required by the syntax rules for URIs (see IETF RFC 2396 [6]). It encodes the KEV ContextObject Representation. The first encoding of

   rft_id=info:doi/10.1126/science.275.5304.1320

replaces the colon character (':') with the character string "%3A" and the forward slash character ('/') with the character string "%2F" to obtain

   rft_id=info%3Adoi%2F10.1126%2Fscience.275.5304.1320
The second URL-encoding replaces the equals character (‘=’) with the character string “%3D” and the percent character (‘%’) with the character string “%25” to obtain

\[ \text{rft_id} \%3D \text{info} \%253Adoi\%252F10.1126\%252Fscience.275.5304.1320 \]

### 21.3 By-Value OpenURL Transports using HTTP(S) POST

In the HTTP(S) POST mode of the *By-Value OpenURL Transport*, the query string specified in Section 21.1 is carried in the message body of the HTTP(S) POST. The Content-Type of the HTTP(S) request *must* be application/x-www-form-urlencoded. Hence, the message body *must* be URL-encoded.

#### Example 26: By-Value OpenURL Transport using HTTP POST

<table>
<thead>
<tr>
<th>Formatted for readability:</th>
</tr>
</thead>
<tbody>
<tr>
<td>base URL : <a href="http://www.example.net/menu">http://www.example.net/menu</a></td>
</tr>
<tr>
<td>POST <a href="http://www.example.net/menu">http://www.example.net/menu</a> HTTP/1.0</td>
</tr>
<tr>
<td>Content-Length: 1279</td>
</tr>
<tr>
<td>Content-Type: application/x-www-form-urlencoded</td>
</tr>
<tr>
<td>&amp;<a href="">ctx:referent</a></td>
</tr>
<tr>
<td>&amp;<a href="">ctx:identifier</a>info:doi/10.1126/science.275.5304.1320 &lt;/ctx:identifier&gt;</td>
</tr>
<tr>
<td>&amp;<a href="">ctx:identifier</a>info:pmid/9036860&lt;/ctx:identifier&gt;</td>
</tr>
<tr>
<td>&lt;/ctx:referent&gt;</td>
</tr>
<tr>
<td>&amp;<a href="">ctx:referring-entity</a></td>
</tr>
<tr>
<td>&lt;/ctx:referring-entity&gt;</td>
</tr>
<tr>
<td>&amp;<a href="">ctx:requester</a></td>
</tr>
<tr>
<td>&amp;<a href="">ctx:identifier</a><a href="mailto:jane.doe@caltech.edu">mailto:jane.doe@caltech.edu</a>&lt;/ctx:identifier&gt;</td>
</tr>
<tr>
<td>&lt;/ctx:requester&gt;</td>
</tr>
<tr>
<td>&amp;<a href="">ctx:referrer</a></td>
</tr>
<tr>
<td>&amp;<a href="">ctx:identifier</a>info:sid/elsevier.com:ScienceDirect&lt;/ctx:identifier&gt;</td>
</tr>
<tr>
<td>&lt;/ctx:referrer&gt;</td>
</tr>
<tr>
<td>&lt;/ctx:context-object&gt;</td>
</tr>
</tbody>
</table>
Example 26 illustrates the HTTP POST method of the By-Value OpenURL Transport of an XML ContextObject Representation. The first part is formatted for readability, and the second part is formatted for actual use with a double URL-encoding as explained in Section 21.2.

The base URL of the Transport (the network location of the Resolver) is http://www.example.net/menu. The value assigned to the url_ctx_fmt key is info:ofi/fmt:xml:xsd:ctx. This declares that the transported ContextObject Representation is based on the XML ContextObject Format. The XML ContextObject Representation is provided as the value assigned to the url_ctx_val key.

As noted in Table 20, the XML ContextObject Format allows bundling multiple ContextObjects into one XML ContextObject Representation. An XML ContextObject Representation may, therefore, contain the description of multiple ContextObjects, all of which are conveyed in a single transportation.

22 Inline OpenURL Transports

An Inline OpenURL Transport transports exactly one KEV ContextObject Representation as part of the query string used in an HTTP(S) GET request or in the message body of an HTTP(S) POST. This differs from the By-Value OpenURL Transport, where the KEV ContextObject Representation is the value associated with the url_ctx_val key.

The Inline OpenURL Transport strongly resembles OpenURL 0.1. The Inline OpenURL Transport may be used only for the transportation of one, and only one, KEV ContextObject Representation. It must not be used for the transportation of ContextObject Representations that conform to any other ContextObject Format.

The Inline OpenURL Transport uses the HTTP network protocol or its secure sibling, HTTPS. The Registry Identifiers for these Transports are:

- Inline OpenURL Transport over HTTP  info:ofi/tsp:http:openurl-inline
- Inline OpenURL Transport over HTTPS info:ofi/tsp:https:openurl-inline

This Section describes both Transports, which are identical except for their use of HTTP or HTTPS as the respective network protocol.
For each transportation of a KEV ContextObject Representation via the Inline OpenURL Transport, a base URL specifies the “Internet host and port, and path” of the target of the transportation, an HTTP(S)-based service called a Resolver.

An Inline OpenURL Transport conveys exactly one KEV ContextObject Representation via HTTP(S) GET and HTTP(S) POST.

The KEV ContextObject Format supports Character Encodings other than the default UTF-8 encoded Unicode. As a result, it is possible to submit KEV ContextObjects Representations via HTML forms. The Character Encoding is declared by assigning a value to the ctx_enc key. This value must be a Registry Identifier of a registered Character Encoding.

Appendix E provides implementation guidelines for the Inline OpenURL Transports.

### 22.1 OpenURL Keys in Inline OpenURL Transports

An Inline OpenURL Transport uses KEV pairs with the following keys, either in the query string of an HTTP(S) GET request or in the message body of an HTTP(S) POST:

- **url_ver**: OpenURL signature
  - **Required**
  - Maximum occurrence: 1
  - Format: fixed value is the case-sensitive character string “Z39.88-2004”
  - Character set and character encoding: value is US-ASCII
  - Example: `url_ver=Z39.88-2004`

- **url_tim**: Datetime of the creation of the OpenURL
  - **Optional**
  - Maximum occurrence: 1
  - Format: ISO8601-conformant datetime, in the YYYY-MM-DD or YYYY-MM-DDTHH:MM:SSZ representation
  - Character set and character encoding: value is US-ASCII and may need URL-encoding
  - Example (not URL-encoded for readability): `url_tim=2002-08-16T17:23:45Z`

- **url_ctx_fmt**: Registry Identifier of the ContextObject Format of the transported ContextObject Representation, which must be the KEV ContextObject Format
  - **Optional**
  - Maximum occurrence: 1
  - Format: fixed value `info:ofi/fmt:kev:mtx:ctx`
  - Character set and character encoding: value is US-ASCII and may need URL-encoding

A foreign key in the Inline OpenURL Transports is any key that is not:

- One of the above OpenURL keys
- A key from the KEV ContextObject Format, which are:
  - Administrative keys (prefixed by `ctx_`). The first encoding is called for by the KEV ContextObject Format; see Section 13.4. The second encoding is called for by the syntax rules for URIs; see IETF RFC 2396 [6].
− Entity Keys (prefixed by rft_, rfe_, req_, rfr_, res_, or svc_).
− Keys from KEV Metadata Formats (prefixed by rft., rfe., req., rfr., res., or svc.).

Foreign keys may be used in an Inline OpenURL Transport, but their meaning is not defined by the Transport. Resolvers may ignore KEV pairs with foreign keys.

22.2 Inline OpenURL Transports using HTTP(S) GET

In the HTTP(S) GET mode of the Inline OpenURL Transport, the query string of an HTTP(S) GET request is the union of the following three sets of KEV pairs:

- A set of KEV pairs with keys from the list of the OpenURL keys described in Section 22.1
- A set of KEV pairs from one, and only one, KEV ContextObject Representation
- A set of KEV pairs with foreign keys, which have no meaning assigned by the OpenURL Framework and may be ignored by Resolvers

It is recommended to strip the query string from a leading ampersand (if there is one).

The resulting set of KEV pairs is expressed as an ampersand-delimited string. The order in which the KEV pairs happen to be concatenated in that string is insignificant, and no meaning should be inferred from the order.

The resulting query string is appended to the base URL of the target Resolver, and separated from it by a question mark ("?"). As specified by the syntax rules for URIs [6], the query string following this question mark must be URL-encoded. Note that, by definition of the KEV ContextObject Format, the values of all KEV pairs in a KEV ContextObject Representation are URL-encoded.

Example 27: Inline OpenURL Transport using HTTP GET

Formatted for readability:

http://www.example.net/menu?
url_ver = 239.88-2004
&url_tim = 2002-03-20T08:55:12Z
&url_ctx_fmt = info:ofi/fmt:kev:mtx:ctx
&rft_id = info:doi/10.1126/science.275.5304.1320
&rft_id = info:pmid/9036860
&rft_val_fmt = info:ofi/fmt:kev:mtx:journal
&rft.jtitle = Science
&rft.atitle = Isolation of a common receptor for coxsackie B viruses and adenoviruses 2 and 5
&rft.aufirst = Bergelson
&rft.aulast = J
&rft.date = 1997
&rft.volume = 275
&rft.spage = 1320
&rft.epage = 1323
&rfe_id = info:doi/10.1006/mthe.2000.0239
&rfr_id = info:sid/elsevier.com:ScienceDirect
&req_id = mailto:jane.doe@caltech.edu
&ctx_tim = 2002-03-20T08:55:12Z
&ctx_enc = info:ofi/enc:UTF-8
Example 27 illustrates the HTTP GET method of the Inline OpenURL Transport of a KEV ContextObject Representation. The first part is formatted for readability: the query string is not URL encoded, white space is introduced, and KEV pairs are on separate lines. The second part is formatted for actual use with a URL-encoded query string.

The base URL of the Transport (the network location of the Resolver) is http://www.example.net/menu. The value assigned to the url_ctx_fmt key is info:ofi/fmt:kev:mtx:ctx. This declares that the transported ContextObject Representation is based on the KEV ContextObject Format (as is required for an Inline OpenURL Transport). The absence of both the url_ctx_ref and url_ctx_val keys indicates that this is an Inline OpenURL Transport. (The presence of the url_ctx_ref key would have indicated a By-Reference OpenURL Transport. The presence of the url_ctx_val key would have indicated a By-Value OpenURL Transport.)

The KEV pairs starting with & rft_id = info:doi/10.1126/science.275.5304.1320 and ending with & ctx_enc = info:ofi/enc=UTF-8 form the KEV ContextObject Representation, consisting of two Identifier Descriptors for the Referent (rft_id), a By-Value Metadata Descriptor for the Referent (rft_val_fmt, keys with a rft. prefix), one Identifier Descriptor for a ReferringEntity (rfe_id), one Identifier Descriptor for a Referrer (rfr_id), and one Identifier Descriptor for a Requester (req_id). The last two KEV pairs specify the time of creation of the ContextObject Representation (ctx_tim) and the Character Encoding used (ctx_enc). The By-Value Metadata Descriptor of the Referent consists of two parts. The first part is a KEV pair that declares the Metadata Format (rft_val_fmt = info:ofi/fmt:kev:mtx:journal), in this case a journal publication in the KEV Metadata Format. The second part is a set of KEV pairs that specify the actual metadata in the specified KEV Metadata Format. These KEV pairs have keys with the rft. prefix to indicate that they represent the Referent.

22.3 Inline OpenURL Transports using HTTP(S) POST

In the HTTP(S) POST mode of the Inline OpenURL Transport, the query string specified in Section 22.2 is carried in the message body of the HTTP(S) POST. The Content-Type of the HTTP(S) request must be application/x-www-form-urlencoded. Hence, the message body must be URL-encoded. Note that the KEV ContextObject Format already requires that values of all KEV pairs occurring in a KEV ContextObject Representation be URL-encoded.

Example 28 shows an HTML form that uses the POST method. It is assumed that the form is inserted in an HTML page that uses UTF-8 for character encoding. The result of submitting the form is the Inline OpenURL Transport of Example 29. It illustrates the HTTP POST method of the Inline OpenURL Transport of a KEV ContextObject Representation. The base URL of the Transport is the network location of the Resolver: http://www.example.net/menu.
Example 28: An HTML Form (POST Method) to generate an Inline OpenURL Transport

```html
<form method="POST" action="http://www.example.net/menu">
  <input type="hidden" name="url_ver" value="Z39.88-2004">
  <input type="hidden" name="url_ctx_fmt" value="info:ofi/fmt:kev:mtx:ctx">
  <input type="hidden" name="rft_id" value="info:doi/10.1126/science.275.5304.1320">
  <input type="hidden" name="rft_id" value="info:pmid/9036860">
  <input type="hidden" name="rft_val_fmt" value="info:ofi/fmt:kev:mtx:journal">
  <input type="hidden" name="rft.jtitle" value="Science">
  <input type="hidden" name="rft.atitle" value="Isolation of a common receptor for coxsackie B viruses and adenoviruses 2 and 5">
  <input type="hidden" name="rft.aulast" value="Bergelson">
  <input type="hidden" name="rft.auinit" value="J">
  <input type="hidden" name="rft.date" value="1997">
  <input type="hidden" name="rft.volume" value="275">
  <input type="hidden" name="rft.spage" value="1320">
  <input type="hidden" name="rft.epage" value="1323">
  <input type="hidden" name="req_id" value="mailto:jane.doe@caltech.edu">
  <input type="hidden" name="rfr_id" value="info:sid/elsevier.com:ScienceDirect">
  <input type="hidden" name="ctx_tim" value="2002-03-20T08:55:12Z">
  <input type="hidden" name="ctx_enc" value="info:ofi/enc:UTF-8">
  <input type="submit" value="send OpenURL">
</form>
```

Example 29: Inline OpenURL Transport using HTTP POST

Formatted for readability:

```
base URL : http://www.example.net/menu

POST http://www.example.net/menu HTTP/1.0
Content-Length: 1480
Content-Type: application/x-www-form-urlencoded

url_ver = Z39.88-2004
& url_tim = 2002-03-20T08:55:12Z
& url_ctx_fmt = info:ofi/fmt:kev:mtx:ctx
& rft_id = info:doi/10.1126/science.275.5304.1320
& rft_id = info:pmid/9036860
& rft_val_fmt = info:ofi/fmt:kev:mtx:journal
& rft.jtitle = Science
& rft.atitle = Isolation of a common receptor for coxsackie B viruses and adenoviruses 2 and 5
& rft.aulast = Bergelson
& rft.auinit = J
& rft.date = 1997
```
The URL-encoded message body is the same as the URL-encoded part of Example 27.

| & rft.volume = 275  |
| & rft.spage = 1320 |
| & rft.epage = 1323 |
| & rfe_id = info:doi/10.1006/mthe.2000.0239 |
| & rfr_id = info:sid/elsevier.com:ScienceDirect |
| & req_id = mailto:jane.doe@caltech.edu |
| & ctx_tim = 2002-03-20T08:55:12Z  |
| & ctx_enc = info:ofi/enc:UTF-8 |

The URL-encoded message body is the same as the URL-encoded part of Example 27.
Appendix A
Responsibilities of the Maintenance Agency for the OpenURL Framework Standard

(This appendix is not part of The OpenURL Framework for Context Sensitive Services, ANSI/NISO Z39.88-2004. It is included for information only.)

Upon approval of this Standard, NISO will establish one or more Maintenance Agencies for the OpenURL standard. The primary responsibility of a Maintenance Agency is to provide ongoing maintenance of the Registry to guarantee stability. Specifically, a Maintenance Agency is responsible to:

- Develop processes and procedures for Registry maintenance and updating consistent with this Standard.
- Facilitate the registration of new entries. When introducing new items into the Registry, a Maintenance Agency should maintain the Registry structure described in Section 6.3. If necessary, a Maintenance Agency may create new areas in the Registry to accommodate new types of Registry entries.
- Correct registry errors. Registered entries are fixed and unchangeable to the maximum practical extent possible. Under exceptional circumstances and with adequate community notification, a Maintenance Agency may correct errors in registered entries. However, a Maintenance Agency must not alter entries for the purpose of introducing new features or accommodating evolving usage. Instead, such evolution must be implemented through the registration of new entries.
- Provide an appropriate machine interface for downloading Registry materials. The initial Registry developed by the Committee supports the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [17]. Systems may cache Registry materials locally to ensure reliable operation, whether or not the Registry is available. It is the responsibility of system developers to update their cached copies.
- Create and maintain an area in the Registry dedicated to security considerations. In this area, the Maintenance Agency should post implementation guidelines and/or requirements to prevent abuse of the OpenURL Framework.
Appendix B
(normative)

B.1 The Z39.88-2004 Matrix Constraint Language

Registry Identifier info:ofi/fmt:kev:mtx

The Z39.88-2004 Matrix Constraint Language is used to specify constraints for descriptions of resources expressed using the KEV Serialization. The Z39.88-2004 Matrix Constraint Language is used to define the syntax and semantics of the KEV ContextObject Format and KEV Metadata Formats.

The Z39.88-2004 Matrix document is expressed in XHTML using a table format to define keys and data types of potential values for the keys. Table 24 displays the complete XHTML underlying the construction of Z39.88-2004 Matrices. This is also available in the Registry at <http://www.openurl.info/registry/docs/html/mtx.html>.

Table 23: Structure of the Z39.88-2004 Matrix

<table>
<thead>
<tr>
<th>Delim</th>
<th>Key</th>
<th>Equals</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>[** Key **]</td>
<td>=</td>
<td>&lt;[** Value **]&gt;</td>
<td>0</td>
<td>1</td>
<td>[** Item definition **]</td>
</tr>
<tr>
<td>#</td>
<td>[** ... **]</td>
<td>[** ... **]</td>
<td>[** ... **]</td>
<td>[** ... **]</td>
<td>[** ... **]</td>
<td>[** This is a comment row **]</td>
</tr>
</tbody>
</table>

Table 23 shows the structure of a Z39.88-2004 Matrix. It consists of the following columns:

- Delim: the ampersand character ('&') delimiter for rows containing syntax rules or the hash character ('#') for comment rows
- Key: the key being defined
- Equals character ('=')
- Value: the data type for the value associated with the key
- Min: the minimum occurrence allowed for the key; an integer
- Max: the maximum occurrence allowed for the key; an integer or an asterisk character (**) to denote 'unbounded'
- Description: a full name of the key, a semantic definition of the key, and any further information

Each row of the Z39.88-2004 Matrix with an ampersand character ('&') in the first column describes the construction of a valid KEV pair. Rows of the Z39.88-2004 Matrix that have a hash character ('#') in the first column are comment rows and **must** be ignored.

One valid KEV pair is obtained by concatenating table entries from the first four columns of a Z39.88-2004 Matrix row that begins with an ampersand character ('&'). Several valid KEV pairs may be concatenated to obtain a description of a resource compliant with a Z39.88-2004 Constraint Definition. The order in which KEV pairs are concatenated is not important.

In comment rows, replace the character string ["** ... **"] with descriptive text. Descriptive text **must not** occur in the Delim column. Usually, only the Description column contains descriptive text.

In the Key column of non-comment rows, the character string ["** Key **"] **must be** replaced with the name of a valid key.
The Value column of a non-comment row of the Z39.88-2004 Matrix assigns a data type to the key, and 
**Value** should be replaced with one of the following available data types:

- `<data>`: character string
- `<id>`: character string for an Identifier (Section 5.2.1)
- `<fmt-id>`: character string for a Format Identifier (Sections 8.2 and 9.2)
- `<m-key>`: character string for a metadata key (Section 14.2)
- `<url>`: character string for a URL [6]
- `<date>`: character string of the form [YYYY-MM-DD| YYYY-MM | YYYY], which represents a date formatted according to the W3C DTF profile of ISO 8601 [12]
- `<time>`: character string of the form [YYYY-MM-DDThh:mm:ssTZD], which represents a complete date plus hours, minutes, and seconds formatted according to the W3C DTF profile of ISO 8601 [12]

In the Description column, **Item definition** should be replaced with descriptive text containing the full name of the key, a semantic definition of the key, and any additional useful information.

**B.2 Constraint Definitions in the KEV ContextObject Format**

The main Constraint Definition associated with the KEV Serialization and the Z39.88-2004 Matrix Constraint Language is the KEV ContextObject Format. This Format defines the Representation of a ContextObject as a concatenation of KEV pairs of the form `&key=value`.

In addition, there are Constraint Definitions known as KEV Metadata Formats that define the Representation of Entities of ContextObjects as a concatenation of KEV pairs. These Representations may be used for both By-Value and/or By-Reference Metadata Descriptors.

In the Registry, a Constraint Definition for a Format expressed in the Z39.88-2004 Matrix Constraint Language is described by the following metadata:

- `dc:title`: the title of the Format
- `dc:creator`: the name of the community that defined the Format
- `dc:description`: a brief description of the Format
- `dc:identifier`: a locator of the Z39.88-2004 Matrix that defines the Format
- `dcterms:created`: the date when the Format was created
- `dcterms:modified`: the date when the Format was modified

Z39.88-2004 Matrix definitions are primarily intended for human reading. To this end, the XHTML Matrix has an associated style sheet that displays the first four rows of each column in bold type to highlight the syntax embedded in the Matrix. However, machine reading is supported, and each cell of the Matrix has an associated class attribute. The W3C XHTML validator button at the foot of the page should be used to validate the XHTML Matrix.

The template for the Z39.88-2004 Matrix displayed in Table 24, which may be used in the creation of KEV ContextObject and Metadata Formats, is also available in the Registry at <http://www.openurl.info/registry/docs/html/mtx.html>.
<table>
<thead>
<tr>
<th>dc:title</th>
<th>KEV [** XX **] Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc:creator</td>
<td>NISO Committee AX, OpenURL Standards Committee</td>
</tr>
<tr>
<td>dc:description</td>
<td>This Matrix represents the [** XX **] Format as a string of ampersand-delimited Key/Encoded-Value pairs</td>
</tr>
<tr>
<td>dc:identifier</td>
<td>info:ofi/fmt:kev:mtx:[<strong>format-id</strong>]</td>
</tr>
<tr>
<td>dcterms:created</td>
<td>[** YYYY-MM-DD **]</td>
</tr>
<tr>
<td>dcterms:modified</td>
<td> </td>
</tr>
</tbody>
</table>
A representation of a Key/Encoded-Value pair is generated by concatenating the contents of the first four columns of a row that begins with an ampersand in the <a href="#theMtx">Matrix</a> below. The ordering of KEV pairs is not important. Rows which have '#' in the first column are comments and <strong>should not</strong> be included in the representation.

The following data types are provided for the values of the Keys, which must be URL-encoded:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;data&gt;</td>
<td>Character string</td>
</tr>
<tr>
<td>&lt;id&gt;</td>
<td>Character string for an Identifier (Z39.88-2004, Part 1, Section 7)</td>
</tr>
<tr>
<td>&lt;fmt-id&gt;</td>
<td>Character string for a Format Identifier (Z39.88-2004, Part 1, Section 12)</td>
</tr>
<tr>
<td>&lt;m-key&gt;</td>
<td>Character string for a Metadata Key (Z39.88-2004, Part 2, Section 8.1)</td>
</tr>
<tr>
<td>&lt;url&gt;</td>
<td>Character string for a &lt;a href=&quot;http://www.ietf.org/rfc/rfc2396.txt&quot;&gt;URL&lt;/a&gt;</td>
</tr>
<tr>
<td>&lt;date&gt;</td>
<td>Character string representing a date to the complete date level of the</td>
</tr>
<tr>
<td></td>
<td>&lt;a href=&quot;http://www.w3.org/TR/NOTE-datetime&quot;&gt;W3CDTF&lt;/a&gt; profile of ISO 8601, of the form: [ YYYY-MM-DD</td>
</tr>
<tr>
<td>&lt;time&gt;</td>
<td>Character string representing a date to the seconds level of the &lt;a href=&quot;http://www.w3.org/TR/NOTE-datetime&quot;&gt;W3CDTF&lt;/a&gt; profile of ISO 8601, of the form: [ YYYY-MM-DDThh:mm:ssTZD</td>
</tr>
</tbody>
</table>

Abbreviations in column headings:

- Delim - Delimiter
- Min - minimum occurrence
- Max - maximum occurrence ('*' = unbounded)
<table>
<thead>
<tr>
<th>Delim</th>
<th>Key</th>
<th>Equals</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>[** Key **]</td>
<td>=</td>
<td>&lt;[** data **]&gt;</td>
<td>0</td>
<td>1</td>
<td>[** Item definition and comment **]</td>
</tr>
</tbody>
</table>

<!-- This is an example comment row -->

<table>
<thead>
<tr>
<th>Comment</th>
<th>ComKey</th>
<th>ComEquals</th>
<th>ComValue</th>
<th>ComMin</th>
<th>ComMax</th>
<th>ComDesc</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>[<strong>...</strong>]</td>
<td> </td>
<td> </td>
<td>0</td>
<td>1</td>
<td>[<strong>This is a comment row</strong>]</td>
</tr>
</tbody>
</table>

</table>

<hr />

<p><a href="http://validator.w3.org/check/referer"><img src="http://www.w3.org/Icons/valid-xhtml10.gif" width="88" height="31" border="0" alt="[Valid XHTML 1.0!]"></a></p>
Appendix C

The Level 1 San Antonio Community Profile

(informative)

An Example of a Community Profile based on the KEV ContextObject Format

(This appendix is not part of The OpenURL Framework for Context Sensitive Services, ANSI/NISO Z39.88-2004. It is included for information only.)

C.1 History

NISO Committee AX created the Level 1 San Antonio Community Profile (SAP1) to support the deployment of an OpenURL Framework Application in the scholarly-information community. SAP1 is built on the KEV ContextObject Format and the OpenURL Transports specified in Part 4. The Registry Identifier of the SAP1 Community Profile is info:ofi/pro:sap1-2004. The mandatory XML Document that defines SAP1 is available at < http://www.openurl.info/registry/docs/pro/info:ofi/pro:sap1-2004 >.

By including the Inline OpenURL Transport as a selected core component, SAP1 provides an elegant migration path from the OpenURL 0.1 specification to this Standard. A description of the upgrade process is presented in Appendix A of the Implementation Guidelines for the KEV ContextObject Format, available in the Registry at <http://openurl.info/registry/docs/implementation_guidelines/>.

C.2 Maintenance of SAP1

NISO Committee AX acts in an advisory capacity until a permanent Maintenance Agency for SAP1 is appointed by NISO. The Maintenance Agency will assume overall responsibility for the further development and maintenance of the SAP1 Community Profile.

C.3 Introduction to SAP1

SAP1 consists of those core components of the OpenURL Framework Standard that were selected by NISO Committee AX on behalf of the scholarly-information community. As required by the OpenURL Framework Standard, the selections are entries from the Registry for the following components: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, and Transports.

As creator of this Standard, NISO Committee AX also specified the initial content of the Registry. Although the initial Registry entries are targeted at the scholarly-information community, Registry entries used by SAP1 may also be valuable for other communities.
C.4 Purpose and Scope

For the scholarly-information community, the major application of the OpenURL Framework is to provide context-sensitive linking from a reference in online scholarly-information systems to resources and services relevant to the referenced item. Generally, the OpenURL Framework is used as follows:

When a user clicks a link or button on an HTML page, information about a scholarly resource (a journal article, for example) and about the context in which it is referenced is transported to a linking server. The transportation mechanism is based on HTTP(S) GET or POST, and is referred to as “an OpenURL”. The purpose of the transportation is to obtain services relevant to the referenced scholarly resource and its context. The transported descriptions of the referenced item and the context are contained in a ContextObject Representation. The ContextObject has six possible Entities, one of which — the Referent — conveys information about the referenced item; the others — the ReferringEntity, Requester, Resolver, ServiceType, and Referrer — convey information about the context of the reference.

Table 25 shows these six Entities together with typical examples from the scholarly-information community. The Table also shows that the Referent is mandatory and that the other five Entities are optional in the KEV ContextObject Format, which is used by SAP1.

Table 25: Use of ContextObject Entities in the Scholarly-Information Community

<table>
<thead>
<tr>
<th>Entity</th>
<th>Definition</th>
<th>Mandatory</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referent</td>
<td>The Entity about which the ContextObject was created—a referenced resource</td>
<td>M</td>
<td>A referenced journal article</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>The Entity that references the Referent</td>
<td>O</td>
<td>A referencing article on EBSCOhost</td>
</tr>
<tr>
<td>Requester</td>
<td>The Entity that requests services pertaining to the Referent</td>
<td>O</td>
<td>The user clicking an OpenURL</td>
</tr>
<tr>
<td>ServiceType</td>
<td>The Entity that defines the type of service requested</td>
<td>O</td>
<td>Fulltext, ILL, etc.</td>
</tr>
<tr>
<td>Resolver</td>
<td>The Entity at which a request for services is targeted</td>
<td>O</td>
<td>A library’s OpenURL linking server</td>
</tr>
<tr>
<td>Referrer</td>
<td>The Entity that generated the ContextObject</td>
<td>O</td>
<td>EBSCOhost</td>
</tr>
</tbody>
</table>

As specified by this Standard, a Community Profile must list Registry selections for the following core components:

- One, and only one, ContextObject Format. This choice implies a selection of:
  - A set of constraints on the type and number of Entities and Descriptors used in ContextObject Representations
  - A constraint on the number of ContextObjects that may be represented in an instance document that conforms to the ContextObject Format
  - One Serialization
− One Constraint Language
− One or more Character Encodings

• Metadata Formats that may be used for By-Value Metadata and/or By-Reference Metadata descriptions. This choice implies a selection of:
  − One or more Serializations
  − One or more Constraint Languages
  − One or more Character Encodings

• Namespaces that may be used to describe Entities with an Identifier Descriptor.

• One or more Transports that specify how ContextObject Representations in the chosen ContextObject Format must be transported.

SAP1 is built around the KEV ContextObject Format. It selects Metadata Formats and Namespaces that meet the needs of the scholarly-information community, and it uses the OpenURL Transports. The SAP1 Community Profile is identified in the Registry as info:ofi/pro:sap1-2004.

C.5 Registry Entries in SAP1

The SAP1 Community Profile is composed of the registered elements listed in Table 26:

Table 26: SAP1 Registered Elements

<table>
<thead>
<tr>
<th>Core Component</th>
<th>Registry Entry</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespaces</td>
<td>Namespace for “ftp” URI Scheme</td>
<td>info:ofi/nam:ftp:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “http” URI Scheme</td>
<td>info:ofi/nam:http:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “https” URI Scheme</td>
<td>info:ofi/nam:https:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “ldap” URI Scheme</td>
<td>info:ofi/nam:ldap:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “mailto” URI Scheme</td>
<td>info:ofi/nam:mailto:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “ISSN” URN Namespace</td>
<td>info:ofi/nam:urn:ISSN:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “NBN” URN Namespace</td>
<td>info:ofi/nam:urn:NBN:</td>
</tr>
<tr>
<td></td>
<td>Namespace for Astrophysics Bibcodes</td>
<td>info:ofi/nam:info:biocode:</td>
</tr>
<tr>
<td></td>
<td>Namespace for Digital Object Identifiers</td>
<td>info:ofi/nam:info:doi:</td>
</tr>
<tr>
<td></td>
<td>Namespaces for CNRI Handles</td>
<td>info:ofi/nam:info:hdl:</td>
</tr>
<tr>
<td></td>
<td>Namespaces for Library of Congress</td>
<td>info:ofi/nam:info:lccn:</td>
</tr>
<tr>
<td></td>
<td>Control Numbers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Namespace for OAI Identifiers</td>
<td>info:ofi/nam:info:oai:</td>
</tr>
<tr>
<td></td>
<td>Namespace for identifiers assigned by</td>
<td>info:ofi/nam:info:oclcnum:</td>
</tr>
<tr>
<td></td>
<td>OCLC to records in the WorldCat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>database</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Namespace for PubMed Identifiers</td>
<td>info:ofi/nam:info:pmid:</td>
</tr>
</tbody>
</table>

97
<table>
<thead>
<tr>
<th>Core Component</th>
<th>Registry Entry</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Namespace for identifiers that follow the info:sid scheme, mainly used for the identification of the Referrer Entity</strong></td>
<td><strong>Namespace for SICI identifiers</strong></td>
<td>info:ofi/nam:info:sid: info:ofi/nam:info:sici:</td>
</tr>
<tr>
<td><strong>UTF-8 Unicode</strong></td>
<td><strong>ISO Latin 1</strong></td>
<td>info:ofi/enc:UTF-8 info:ofi/enc:ISO-8859-1</td>
</tr>
<tr>
<td><strong>Inline OpenURL</strong></td>
<td><strong>By-Value OpenURL</strong></td>
<td>info:ofi/tsp:http:openurl-inline info:ofi/tsp:http:openurl-by-val</td>
</tr>
<tr>
<td><strong>By-Reference OpenURL</strong></td>
<td></td>
<td>info:ofi/tsp:http:openurl-by-ref</td>
</tr>
</tbody>
</table>
Appendix D

The Level 2 San Antonio Community Profile

(informative)

An Example of a Community Profile based on the XML ContextObject Format

(This appendix is not part of The OpenURL Framework for Context Sensitive Services, ANSI/NISO Z39.88-2004. It is included for information only.)

D.1 History

NISO Committee AX created the Level 2 San Antonio Community Profile (SAP2) to support the deployment of an OpenURL Framework Application in the scholarly-information community. SAP2 is built on the XML ContextObject Format and the OpenURL Transports specified in Part 4. The Registry Identifier of the SAP2 Community Profile is info:ofi/pro:sap2-2004. The mandatory XML Document that defines SAP2 is available at <http://www.openurl.info/registry/docs/pro/info:ofi/pro:sap2-2004>.

D.2 Maintenance of SAP2

NISO Committee AX acts in an advisory capacity until a permanent Maintenance Agency for SAP2 is appointed by NISO. The Maintenance Agency will assume overall responsibility for the further development and maintenance of the SAP2 Community Profile.

D.3 Introduction to SAP2

SAP2 consists of those core components of the OpenURL Framework Standard that were selected by NISO Committee AX on behalf of the scholarly-information community. As required by the OpenURL Framework Standard, the selections are entries from the Registry for the following components: Namespaces, Character Encodings, Serializations, Constraint Languages, ContextObject Formats, Metadata Formats, and Transports.

As creator of this Standard, NISO Committee AX also specified the initial content of the Registry. Although the initial Registry entries are targeted at the scholarly-information community, Registry entries used by SAP2 may also be valuable for other communities.

D.4 Purpose and Scope

For the scholarly-information community, the major application of the OpenURL Framework is to provide context-sensitive linking from a reference in online scholarly information systems to resources and services relevant to the referenced item. Generally, the OpenURL Framework is used as follows:
When a user clicks a link or button on an HTML page, information about a scholarly resource (a journal article, for example) and about the context in which it is referenced is transported to a linking server. The transportation mechanism is based on HTTP(S) GET or POST, and is referred to as "an OpenURL". The purpose of the transportation is to obtain services relevant to the referenced scholarly resource and its context. The transported descriptions of the referenced item and the context are contained in a ContextObject Representation. The ContextObject has six possible Entities, one of which — the Referent — conveys information about the referenced item; the others — the ReferringEntity, Requester, Resolver, ServiceType, and Referrer — convey information about the context of the reference.

Table 27 shows these six Entities together with typical examples from the scholarly-information community. The Table also shows that the Referent is mandatory and that the other five Entities are optional in the XML ContextObject Format used by SAP2.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Definition</th>
<th>Mandatory</th>
<th>Optional</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referent</td>
<td>The Entity about which the ContextObject was created—a referenced resource</td>
<td>M</td>
<td></td>
<td>A referenced journal article</td>
</tr>
<tr>
<td>ReferringEntity</td>
<td>The Entity that references the Referent</td>
<td>O</td>
<td></td>
<td>A referencing article on EBSCOhost</td>
</tr>
<tr>
<td>Requester</td>
<td>The Entity that requests services pertaining to the Referent</td>
<td>O</td>
<td></td>
<td>The user clicking an OpenURL</td>
</tr>
<tr>
<td>ServiceType</td>
<td>The Entity that defines the type of service requested</td>
<td>O</td>
<td></td>
<td>Fulltext, ILL, etc.</td>
</tr>
<tr>
<td>Resolver</td>
<td>The Entity at which a request for services is targeted</td>
<td>O</td>
<td></td>
<td>A library’s OpenURL linking server</td>
</tr>
<tr>
<td>Referrer</td>
<td>The Entity that generated the ContextObject</td>
<td>O</td>
<td></td>
<td>EBSCOhost</td>
</tr>
</tbody>
</table>

As specified by this Standard, a Community Profile must list Registry selections for the following core components:

- One, and only one, ContextObject Format. This choice implies a selection of:
  - A set of constraints on the type and number of Entities and Descriptors used for ContextObject Representations
  - A constraint on the number of ContextObjects that may be represented in an instance document that conforms to the ContextObject Format
  - One Serialization
  - One Constraint Language
  - One or more Character Encodings

- Metadata Formats that may be used for By-Value Metadata and/or By-Reference Metadata. This choice implies a selection of:
  - One or more Serializations
One or more Constraint Languages
One or more Character Encodings

- Namespaces that may be used to describe Entities with an Identifier Descriptor.
- One or more Transports that specify how ContextObject Representations in the chosen ContextObject Format must be transported.

SAP2 is built around the XML ContextObject Format. It selects Metadata Formats and Namespaces that meet the needs of the scholarly-information community and it uses the OpenURL Transports. The SAP2 Community Profile is identified in the Registry as info:ofi/pro:sap2-2004.

### D.5 Registry Entries in SAP2

The SAP2 Community Profile is composed of the registered elements listed in Table 28:

<table>
<thead>
<tr>
<th>Core Component</th>
<th>Registry Entry</th>
<th>Registry Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespaces</td>
<td>Namespace for “ftp” URI Scheme</td>
<td>info:ofi/nam:ftp:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “http” URI Scheme</td>
<td>info:ofi/nam:http:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “https” URI Scheme</td>
<td>info:ofi/nam:https:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “ldap” URI Scheme</td>
<td>info:ofi/nam:ldap:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “mailto” URI Scheme</td>
<td>info:ofi/nam:mailto:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “ISSN” URN Namespace</td>
<td>info:ofi/nam:urn:ISSN:</td>
</tr>
<tr>
<td></td>
<td>Namespace for “NBN” URN Namespace</td>
<td>info:ofi/nam:urn:NBN:</td>
</tr>
<tr>
<td></td>
<td>Namespace for Astrophysics Bibcodes</td>
<td>info:ofi/nam:info:bibcode:</td>
</tr>
<tr>
<td></td>
<td>Namespace for Digital Object Identifiers</td>
<td>info:ofi/nam:info:doi:</td>
</tr>
<tr>
<td></td>
<td>Namespaces for CNRI Handles</td>
<td>info:ofi/nam:info:hdl:</td>
</tr>
<tr>
<td></td>
<td>Namespaces for Library of Congress Control Numbers</td>
<td>info:ofi/nam:info:lccn:</td>
</tr>
<tr>
<td></td>
<td>Namespace for OAI Identifiers</td>
<td>info:ofi/nam:info:oai:</td>
</tr>
<tr>
<td></td>
<td>Namespace for identifiers assigned by OCLC to records in the WorldCat database</td>
<td>info:ofi/nam:info:oclcnum:</td>
</tr>
<tr>
<td></td>
<td>Namespace for PubMed Identifiers</td>
<td>info:ofi/nam:info:pmid:</td>
</tr>
<tr>
<td></td>
<td>Namespace for identifiers that follow the info:sid scheme, mainly used for the identification of the Referrer Entity</td>
<td>info:ofi/nam:info:sid:</td>
</tr>
<tr>
<td></td>
<td>Namespace for SICI identifiers</td>
<td>info:ofi/nam:info:sici:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character Encodings</th>
<th>UTF-8 Unicode</th>
<th>info:ofi/enc:UTF-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialization</td>
<td>W3C XML 1.0</td>
<td>info:ofi/fmt:xml</td>
</tr>
<tr>
<td>Core Component</td>
<td>Registry Entry</td>
<td>Registry Identifier</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>XML Metadata Format</strong> for Patents</td>
<td>info:ofi/fmt:xml:xsd:patent</td>
</tr>
<tr>
<td></td>
<td><strong>XML Metadata Format for ServiceTypes for the scholarly-information community</strong></td>
<td>info:ofi/fmt:xml:xsd:sch_svc</td>
</tr>
<tr>
<td>Transports</td>
<td>By-Value OpenURL</td>
<td>info:ofi/tsp:http:openurl-by-val</td>
</tr>
<tr>
<td></td>
<td>By-Reference OpenURL</td>
<td>info:ofi/tsp:http:openurl-by-ref</td>
</tr>
</tbody>
</table>
Appendix E
Implementation Guidelines for the OpenURL Transports
(informative)

(This appendix is not part of The OpenURL Framework for Context Sensitive Services, ANSI/NISO Z39.88-2004. It is included for information only.)

E.1 Length of HTTP(S) GET URIs

Transport techniques based on HTTP(S) GET are subject to length limitations on the GET URI. The OpenURL Standard does not place any a priori limit on the length of an OpenURL. However, Resolvers must be able to accept OpenURLs as long as 255 bytes after encoding and should be able to accept OpenURLs as long as 2048 bytes.

E.2 URL-Encoding and URL-Decoding

URL-encoding and decoding of HTTP(S) GET and POST query string values prevent the misinterpretation of special characters occurring in these values.

To form an encoded value from a value, a procedure called URL-encoding is used:

1. The alphanumeric characters 'a' through 'z', 'A' through 'Z', and '0' through '9' remain unchanged.
2. The period character ('.'), the hyphen character ('-'), the asterisk character ('*'), and the underscore character ('_') remain unchanged.
3. The space character ( ' ') is replaced with a plus-sign character ('+') or with the character string '%20'.
4. All other characters (the unsafe characters) are first converted into one or more bytes using the UTF-8 encoding method (or another encoding if specified by the ContextObject Format). Then, each byte is represented by the 3-byte string '%XY', where XY is the two-digit hexadecimal representation of the byte.

To form a value from an encoded value, a procedure called URL-decoding is used. It reverses the URL-encoding procedure:

1. The plus-sign character ('+') is replaced by the space character (' ').
2. Each instance of a three-byte string '%XY', where XY is a hexadecimal number, is replaced with the corresponding byte.
3. The bytes are converted to Unicode characters using UTF-8, unless otherwise specified by a ContextObject Format.

E.3 Parsing of HTTP(S) Query Strings

Upon receiving an OpenURL request, the Resolver may parse and URL-decode the query string into a set of KEV pairs. Depending on the type of Transport, a Resolver may encounter three types of keys: OpenURL keys, KEV ContextObject keys, and foreign keys.

1. All OpenURL Transports use OpenURL keys. They are defined in Sections 20.1, 21.1, and 22.1. All OpenURL keys share the prefix url_.
2. In addition to the OpenURL keys, the Inline OpenURL Transport also uses keys from the KEV ContextObject Format. These keys are defined in Section 13.2. They are:
   - Administrative keys (prefixed by \texttt{ctx\_}).
   - Entity keys (prefixed by \texttt{rft\_}, \texttt{rfe\_}, \texttt{req\_}, \texttt{rfr\_}, \texttt{res\_}, or \texttt{svc\_}).
   - Keys from KEV Metadata Formats (prefixed by \texttt{rft\_}, \texttt{rfe\_}, \texttt{req\_}, \texttt{rfr\_}, \texttt{res\_}, or \texttt{svc\_}).

3. All Transports \textit{may} use foreign keys. Foreign keys are keys that do not reside under the categories specified by 1 and 2 above. The OpenURL Transports do not define their meaning.