ANSI/NISO Z39.50-2003 **ISSN: 1041-5653**
(revision of Z39.50-1995)

Information Retrieval (Z39.50): Application Service Definition and Protocol Specification

Abstract: This standard defines a client/server based service and protocol for Information Retrieval. It specifies procedures and formats for a client to search a database provided by a server, retrieve database records, and perform related information retrieval functions. The protocol addresses communication between information retrieval applications at the client and server; it does not address interaction between the client and the end-user.

An American National Standard
Developed by the
National Information Standards Organization

Approved November 27, 2002
by the
American National Standards Institute

Published by the National Information Standards Organization
Bethesda, Maryland

NISO Press, Bethesda, Maryland, U.S.A.

Published by
NISO Press
4733 Bethesda Avenue, Suite 300
Bethesda, MD 20814
www.niso.org

Printed in the United States of America
ISSN: 1041-5653 National Information Standard Series
ISBN: 1-880124-55-6

∞ This paper meets the requirements of ANSI/NISO Z39.48-1992 Permanence of Paper.

Library of Congress Cataloging-in-Publication Data

National Information Standards Organization (U.S.)
Information retrieval (Z39.50) : application service definition and protocol specification : an American national standard / developed by the National Information Standards Organization.
p. cm. -- (National information standards series. ISSN 1041-5653)
"Approved November 27, 2002 by the American National Standards Institute."
"ANSI/NISO Z39.50-2003 (maintenance revision of Z39.50-1995.)"
ISBN 1-880124-55-6 (alk. paper)
1. Library information networks--Standards--United States. 2. Information storage and retrieval systems--Standards-- United States. 3. Computer network protocols--Standards--United States. I. Title: Z39.50. II American National Standards Institute. III. Title. IV. Series.

Z674.8.N44 2003
025'.0028546--dc21 200342115

TABLE OF CONTENTS

Foreword

(This foreword is not a part of NISO Z39.50-2003 Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. It is included for information only.)

This standard is a maintenance revision of Z39.50-1995. Appendix 19 describes how this version differs from Z39.50-1995, and the reasons for these changes.

This standard was processed and approved for submittal to ANSI by the National Information Standards Organization. It was balloted by the NISO Voting Members March 29, 2002 - May 13, 2002. It will next be reviewed in 2007. 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 necessarily imply that all Voting Members voted for its approval. At the time it approved this standard, NISO had the following members:

NISO Voting Members

3M
Jerry Karel
Susan Boettcher (Alt)

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

American Chemical Society
Robert S. Tannehill, Jr.

American Library Association
Paul J. Weiss

American Society for Information Science and Technology
Mark H. Needleman

American Society of Indexers
Judith Gibbs
Jacqueline Rodebaugh (Alt)

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
David L. Austin

Association for Information and Image Management
Betsy A. Fanning

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

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

BiblioMondo Inc.
Martin Sach

Book Industry Communication
Brian Green

Broadcast Music Inc.
Edward Oshanani
Robert Barone (Alt)

Cambridge Information Group
Michael Cairns
Matthew Dunie (Alt)

Checkpoint Systems, Inc.

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

Congressional Information Service, Inc.
Robert Lester

divine, inc.
Robert Boissy

Elsevier Science Inc.
Anthony Ross
John Mancia (Alt)

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

epixtech, inc.
John Bodfish
Ricc Ferante (Alt)

Ex Libris
James Steenbergen
Carl Grant (Alt)

Follett Corp.
D. Jeffrey Blumenthal
Don Rose (Alt)

Fretwell-Downing Informatics
Robin Murray

Gale Group
Katherine Gruber
Justine Carson (Alt)

Gaylord Information Systems
William Schickling
Linda Zaleski (Alt)

GCA Research Institute
Jane Harnad

H.W. Wilson Company
Ann Case

IBM
David M. Choy
Chuck Brink (Alt)

Information Use Management & Policy Institute
Charles McClure
John Carlo Bertot (Alt)

Infotrieve
Jan Peterson

Innovative Interfaces, Inc.
Gerald M. Kline
Sandy Westall (Alt)

Institute for Scientific Information

The International DOI Foundation
Norman Paskin

Library Binding Institute
Donald Dunham

The Library Corporation
Mark Wilson
Joe Zeeman (Alt)

Library of Congress
Winston Tabb
Sally H. McCallum (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
Cameron Bardrick (Alt)

Motion Picture Association of America
William M. Baker
Axel aus der Muhlen (Alt)

Music Library Association
Mark McKnight
Lenore Coral (Alt)

National Agricultural Library
Gary K. McCone

National Archives and Records Administration
Mary Ann Hadyka

National Federation of Abstracting and Information Services
Marion Harrell

National Library of Medicine
Betsy L. Humphreys

Nylink
Mary-Alice Lynch
Jane Neale (Alt)

OCLC, Inc.
Donald J. Muccino

Openly Informatics
Eric Hellman (Alt)

Proquest Information and Learning
Todd Fegan
James Brei (Alt)

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

Research Libraries Group
Lennie Stovel
Joan Aliprand (Alt)

SIRS Mandarin, Inc.
Leonardo Lazo
Harry Kaplanian (Alt)

SIRSI Corporation
Greg Hathorn
Slavko Manojlovich (Alt)

Society for Technical Communication
Annette Reilly
Kevin Burns (Alt)

Society of American Archivists
Lisa Weber

Special Libraries Association
Marcia Lei Zeng

Triangle Research Libraries Network
Jordan M. Scepanski
Mona C. Couts (Alt)

U.S. Department of Commerce, National Institute of Standards and Technology, Office of Information Services

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

U.S. National Commission on Libraries and Information Science
Denise Davis

VTLS, Inc.
Vinod Chachra

NISO Board of Directors

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

Beverly P. Lynch, Chair
University of California, Los Angeles

Jan Peterson, Vice Chair/Chair-Elect
Infotrieve, Inc.

Donald J. Muccino, Immediate Past Chair
OCLC

Jan Peterson, Treasurer
Infotrieve, Inc.

Patricia R. Harris, Executive Director
National Information Standards Organization

Pieter S. H. Bolman
Academic Press

Albert Simmonds
OCLC, Inc.

Priscilla Caplan
Florida Center for Library Automation

Carl Grant
Ex Libris (USA), Inc.

Brian Green
BIC/EDItEUR

Jose-Marie Griffiths
University of Pittsburgh

Richard E. Luce
Los Alamos National Laboratory

Sally McCallum
Library of Congress

Norman Paskin
The International DOI Foundation

Steven Puglia
U.S. National Archives and Records Administration

Information Retrieval (Z39.50): Application Service Definition and Protocol Specification

1. Introduction

This standard, ANSI/NISO Z39.50-2003, Information Retrieval (Z39.50) Application Service Definition and Protocol Specification, defines an application protocol for search and retrieval of information in databases.

1.1 Scope and Field of Application

This standard describes the Information Retrieval Application Service (section 3) and specifies the Information Retrieval Application Protocol (section 4). The service definition describes services that support capabilities within an application; the services are in turn supported by the Z39.50 protocol. The description neither specifies nor constrains the implementation within a computer system. The protocol specification includes the definition of the protocol control information, the rules for exchanging this information, and the conformance requirements to be met by implementation of this protocol.

This standard is intended for systems supporting information retrieval services for organizations such as information services, universities, libraries, and union catalogue centers. It addresses connection oriented, program - to - program communication. It does not specify a user interface.

1.2 Version

Z39.50-1995 specifies versions 2 and 3 for the Z39.50 service and protocol. This standard Z39.50-2003 also specifies versions 2 and 3, and additionally, incorporates many clarifications, amendments, defect corrections, and implementer agreements, all of which have been endorsed by the Z39.50 Implementers Group.

Z39.50-1992 specifies version 2 only. Version 2 of Z39.50 is assumed identical to version 1 of Z39.50; thus implementations that support version 2 automatically support version 1. Implementations that support version 3 are required to support version 2 (and thus version 1 as well).

Certain procedures specified within the standard apply specifically to version 2 or version 3 and are noted as such.

1.3 References

ANSI/NISO Z39.53-2001	Codes for the Representation of Languages for Information Interchange.
ISO 2709	Documentation - Format for Bibliographic Information Interchange on Magnetic Tape.

ISO 4217	Codes for the representation of currencies and funds 1990.
ISO 8777	Information and Documentation - Commands for Interactive Text Searching.
ISO 8824:1990	Specification of Abstract Syntax Notation one (ASN.1).
ISO 8825:1990	Specification of basic Encoding Rules for Abstract Syntax Notation one (ASN.1/BER).
ISO 10160	Information and Documentation - Interlibrary Loan Application Service Definition for Open Systems Interconnection 1991.
ISO 10161	Information and Documentation - Interlibrary Loan Application Protocol Specification for Open Systems Interconnection 1991.

2. Definitions

Abstract database record	An abstract representation of the information in a database record. An abstract database record may be formed by the application of an abstract record structure (defined by a schema) to the database record. An element specification may be applied to an abstract database record forming another instance of the abstract database record.
Abstract record structure	The primary component of a database schema. An abstract record structure applied to database record results in an abstract database record.
Abstract syntax	A description of a particular data type using an abstract syntax notation. It can be referenced by an OID (object identifier).
Abstract syntax notation	A language that allows the description of data types in a representation-independent manner. ASN.1 is an example.
Access point	A unique or non-unique key that can be specified (either alone or in combination with other access points) in a search for records. An access point may or may not correspond to one or more elements (defined by an abstract syntax),.
Access point clause	An operand of a type-1 query (informal).
Aggregate present response	Segment requests (if any) together with the Present response, for a Present operation.
APDU	See Application Protocol Data Unit.
Application Protocol	The rules governing the format and exchange of information between a client and server.
Application Protocol Data Unit	A unit of information, transferred between client and server, whose format is specified by the Z39.50 protocol, consisting of application-protocol-information and possibly application-user-data.
Application ProtocolControl Information	Information conveyed by an application protocol data unit.
AppliedVariant	One of three usages for a variant specification. The applied variant is the variant specification that the server applied to an element included in a retrieval record. See also variantRequest and supportedVariant.

ARS	See Abstract record structure.
ASN.1	Abstract Syntax Notation One, as specified in ISO 8824.
Attribute	A characteristic of a search term, or one of several characteristic components which together form a characteristic of a search term.
Attribute element	An attribute represented by a pair of components: an attribute type and a value of that type.
Attribute list	A set of attribute elements and the attribute set id to which it belongs. An attribute list is combined with a search term to form an operand of a type-1 query. Usually, one of the attribute elements from the set corresponds to a normalized access point, against which the term (as qualified by the other attribute elements) is matched.
Attribute set	A set of attribute types, and for each, a list of attribute values. Each type is represented by an integer, unique within that set (as identified by its attribute set id), and each value for a given type is unique within that type.
Attribute set id	An OID that identifies an attribute set, to which an attribute element (within an attribute list) belongs.
Attribute type	A component of an attribute element. An attribute set defines one or more attribute types and assigns an integer to each type (it also defines values specific to each type). For example, bib-1 assigns the integer 1 for the attribute type "Use."
Attribute value	A component of an attribute element. An attribute set defines one or more values for each attribute type that it defines. For example, bib-1 defines the Use attribute "personal name."
bib-1	An attribute set, formerly defined in Z39.50-1995. Its definition is available at http://lcweb.loc.ogv/z3950/agency/defns/bib1.html
Client	The initiating application.
Client system	The system on which the client resides.
Composition specification	A specification that may be included in a Present request to indicate the desired composition (elements and record syntax) of the retrieval records. It includes a schema identifier, element specification, and record syntax identifier.
Conditionally confirmed service	A service that may be invoked as confirmed or non-confirmed. It is defined in terms of a request (from the

	client or server) followed possibly by a response (from the peer). For example, Resource-control is a conditionally confirmed service, initiated by the server. See also Non-confirmed service and Confirmed service.
Confirmed service	A service that is defined in terms of a request (from the client or server) followed by a response (from the peer). For example, Search is a confirmed service, initiated by the client; Access-control is a confirmed service initiated by the server. See also Non-confirmed service and Conditionally-confirmed service.
Database	A collection of information units containing related information. Each unit is a database record.
Database record	A local data structure representing an information unit in a database.
Database schema	A common understanding shared by the client and server of the information contained in the records of the database, which allows the subsequent selection of portions of that information via an element specification. A schema defines an abstract record structure, which, when applied to a database record, results in an abstract database record.
Data element	See Element.
Element	A unit of information defined by a schema.
ElementRequest	A request, included with an element specification, for the retrieval of a specific element. The element request may include a variantRequest, indicating the desired variant form of the element.
Element set name	An element specification in the form of a primitive name.
Element specification	An instance of an element specification format, or an element set name. An element specification transforms an abstract database record into another instance of the abstract database record (this may be a null transformation). The element specification selects elements from the abstract database record, and possibly also specifies variant forms for those elements.
Element specification format	A structure used to express an element specification.
Element specification identifier	The object identifier of an element specification format, or an element set name.
Exceptional record size	The maximum size of the record that may be included in a Present response, in the special case when a single, exceptionally large record (i.e. larger than preferred-message-size) is requested.

Facility	A logical group of Z39.50 services; in some cases, a single service. For example, the Retrieval facility consists of the Present service and the Segment service; the Search facility consists of the Search service. Alternatively, a facility might not consist of services, but instead might use services of other facilities. For example, the Explain facility does not define any services, but uses the Search and Present services.
Final fragment	A fragment that ends at the end of a record. See Fragment.
Fragment	A proper substring of a record. (This definition is meaningful only in the context of level-2 segmentation, described in section 3.3.3; within that section, a record is considered to be a string of bytes.)
GRS	Generic Record Syntax.
Initiating request	A request that initiates an operation.
Intermediate fragment	A fragment that neither starts at the beginning nor ends at the end of a record. See Fragment.
IR	Information Retrieval.
Item	(1) A result set item. (2) A bibliographic item; see ISO 10160.
Leaf-node	A terminal node in a tree. A node with no children.
Maximum segment size	The largest allowable segment of an aggregate Present response (when segmentation is in effect).
Non-confirmed service	A service that is defined in terms of a request from the client or server, with no corresponding response. For example, Segment is a non-confirmed service initiated by the server. See also Confirmed service.
Object identifier	An unambiguous, globally-recognized, registered identifier for a data object, assigned by a registration authority.
OID	See Object identifier.
Operation	An initiating request and the corresponding terminating response, along with intervening related messages. For example, a Search operation always includes a Search request and Search response, and may also include access control and resource control messages. Multiple concurrent operations may occur within a Z-association.
Operation type	The name of an initiating request. For example, a Search request initiates an operation whose type is "search."

Preferred message size	The maximum size of a Search response or Present response when no segmentation is in effect. It is expressed in terms of the sum of the sizes (in bytes) of the response records, not including protocol control information.
Primitive Name	A name whose internal structure is not required to be understood or have significance to users of the name.
Record syntax	An abstract syntax requested by the client or used by the server to represent retrieval records. For a complete definition, see section 3.6.3.
Response record	A retrieval record or a surrogate diagnostic record, representing a database record, in a Search response or (aggregate) Present response.
Result set	A local data structure used as a selection mechanism for the transfer of records, identified by a query. Its logical structure is a named, ordered list of result set items, and possibly, unspecified information which may be used as a surrogate for the search that created the result set.
Result set item	A database name, a pointer to a record within the database, and possibly, additional, unspecified information associated with the record.
Result set record	(Informal) The database record represented by a result set item. See Result set.
Retrieval record	The exportable structure defined by the application of a record syntax to an abstract database record.
RPN	Reverse Polish Notation.
Schema	See Database schema.
Segment	A message that is sent (or is in preparation for transmission) by the server as part of an aggregate Present response, i.e. a Segment request or Present response.
Server	The responding application, associated with one or more databases.
Server system	The system on which the server resides.
Service	(1) A Z39.50 service, as in the "search" service; (2) an extended service, as in the "persistent result set extended service".
Simple Present response	An aggregate Present response consisting of a single segment, i.e., consisting of a Present response only, and no Segment requests.

Starting fragment	A fragment that starts at the beginning of a record. See Fragment.
SupportedVariant	One of three usages for a variant specification. A supportedVariant is a variant specification that the server lists as supported for a particular element. See also appliedVariant and variantRequest.
Surrogate diagnostic Record	A diagnostic record supplied in place of a retrieval record, representing a database record.
Tag	The identifier of an element (or of a node of the tagPath representing an element). It consists of a tagType and a tagValue.
TagPath	A sequence of nodes from the root of a tree to the node that the tagPath represents (when the elements of a record are represented hierarchically, as a tree). Each node of a tagPath is represented by a tag. The end-node might be a leaf-node, in which case the tagPath represents an element; otherwise the tagPath represents the subtree whose root is that node.
TagSet	The tagValues (and recommended data types) for a set of elements.
TagSetId	An object identifier serving as a persistent identifier for a tagSet.
TagType	A short-hand (integer) identifier for a tagSet. A schema definition may assign a tagType to a TagSetId, to identify a particular tagSet (within the context of the schema definition).
TagValue	The identifier of an element (or of a node of the tagPath representing an element). It may be either integer or string, and it is qualified by a tagType.
Terminating response	A response that ends an operation.
Transfer syntax	A syntax that when paired with an abstract syntax forms a record syntax.
Triple	A 3-tuple. (I.e. an n-tuple, where n = 3.)
Variant	One of possibly several forms in which an element is available for retrieval. The client may request, or the server present, an element according to a specific variant. The server may indicate what variants are available for an element.
Variant list	A list provided by the server of the supportedVariants for a particular element.
VariantRequest	One of three usages for a variant specification. A

	variantRequest is a variant specification occurring within an element request. See also appliedVariant and supportedVariant.
Variant set	A definition of a set of classes; for each class, a set of types; and for each type, a set of values. A variant specification consists of a set of variantSpecifiers from a particular variant set.
Variant set identifier	An OID identifying a variant set.
Variant Specification	A variantRequest, appliedVariant, or supportedVariant. A variant specification is a sequence of triples, each of which is a variantSpecifier.
Variant Specifier	A component of a variant specification. It consists of a class, a type defined for that class, and a value defined for that type.
Z-association	See Z39.50-association.
Z39.50-association	A session, explicitly established by the client and either explicitly terminated by the client or server, or implicitly terminated by loss of connection. Communication between client and server is via a Z39.50-association.

3. Information Retrieval Service

The Information Retrieval service definition describes an activity between two applications: an initiating application, the client, and a responding application, the server. The server is associated with one or more databases.

Communication between the client and server is carried out by the Z39.50 protocol. The specification is logically divided into procedures pertaining to the client and procedures pertaining to the server.

3.1 Model and Characteristics of the Information Retrieval Service

Communication between client and server is via a Z39.50-Association (Z-association). A Z-association is explicitly established by the client and may be explicitly terminated by either client or server, or implicitly terminated by loss of connection.

There may be multiple consecutive Z-associations for a connection. There may be multiple consecutive as well as concurrent operations (see 3.1.2) within a Z-association.

The roles of client and server may not be reversed within a Z-association. A Z-association cannot be restarted, thus once a Z-association is terminated no status information is retained, except information that is explicitly saved.

The service definition describes services and operations; models for these are described in 3.1.1 and 3.1.2. Services are grouped by facilities; the Z39.50 facilities and services are defined in 3.2.

3.1.1 Z39.50 Services

Z39.50 services are carried out by the exchange of messages between the client and server. A message is a request or a response. Services are defined to be confirmed, non-confirmed, or conditionally-confirmed. A confirmed service is defined in terms of a request (from the client or server) followed by a response (from the peer). For example, Search is a confirmed service, initiated by the client; the Search service is defined in terms of a Search request from the client followed by a Search response from the server. Access-control is an example of a confirmed service initiated by the server.

A non-confirmed service is defined in terms of a request from the client or server, with no corresponding response. For example, TriggerResourceControl is a non-confirmed service initiated by the client; Segment is a non-confirmed service initiated by the server.

A conditionally-confirmed service is a service that may be invoked as either a confirmed or non-confirmed service. It is defined in terms of a request (from the client or server) followed possibly by a response (from the peer). For example, Resource-control is a conditionally-confirmed service, initiated by the server.

3.1.2 Z39.50 Operations

This standard describes nine operation types: Init, Search, Present, Delete, Scan, Sort, Resource-report, Extended-services, and Duplicate Detection.

A request from the client of a particular operation type initiates an operation of that type (for example a Search request initiates a Search operation) which is terminated by the respective response from the server. Only the client may initiate an operation, and not all client requests do so (see 3.4).

A request that initiates an operation is called an initiating request and a response that ends an operation is called a terminating response. From the client perspective, an operation begins when it issues the initiating request, and ends when it receives the terminating response. From the server perspective, the operation begins when it receives the initiating request and ends when it sends the terminating response. An operation consists of the initiating request and the terminating response, along with any intervening related messages (see 3.4).

3.1.3 Model of a Database

The objective of this standard is to facilitate interconnection of clients and servers for applications where clients search and retrieve information from server databases. The ways in which databases are implemented differ considerably; different systems have different styles for describing the storage of data and the means by which it can be accessed. A common, abstract model is therefore used in describing databases, to which an individual system can map its implementation. This enables different systems to communicate in standard and mutually understandable terms, for the purpose of searching and retrieving information from a database. The search and retrieval models are described in 3.1.4 and 3.1.5.

The term database, as used in this standard, refers to a collection of records. Each record is a collection of related information. The term database record refers to a local data structure representing the information in a particular record. Associated with a database are one or more sets of access points that can be specified in a search for database records (see 3.1.4), and one or more sets of elements that may be retrieved from a database record (see 3.1.5). An access point is a unique or non-unique key that can be specified (either alone or in combination with other access points) in a search for records. An access point may or may not correspond to one or more elements (defined by an abstract syntax), For example, the (abstract) access point “title” might be used to search a particular database and might correspond to the Main Title data element for that database. The same access point might be used to search a different database, and for that database it might correspond to the Series Title data element.

3.1.4 Searching a Database

A query is applied to a database, specifying values to be matched against the access points of the database. The subset of records formed by applying a query is called the result set (see 3.1.6). A result set may itself be referenced in a subsequent query and manipulated to form a new result set.

A search request specifies one or more databases and includes a query. The type-1 query defined in this standard (see 3.7) consists of either a single access point clause, or several access point clauses linked by logical operators. For example:

In the database named "science fiction" find all records for which 'title' contains "galaxy" AND 'author' contains "adams". (" 'title' contains "galaxy" " is an access point clause, as is " 'author' contains "adams" ". "AND" is a logical operator.)

Each access point clause consists of a search term and attributes. The attributes qualify the term; usually, one of the attributes corresponds to a normalized access point, against which the term (as qualified by the other attributes) is matched. Each attribute is a pair representing an attribute type and a value of that type (for example, type might be "usage" and value "author"; or type might be "truncation" and value "left").

Each attribute is qualified by an attribute set id, which identifies the attribute set to which the attribute belongs. An attribute set specifies a set of attribute types, and for each, a list of attribute values.

3.1.5 Retrieving Records from a Database

Following the processing of a search, the result set is available at the server, for reference by the client, for subsequent searches or retrieval requests. When requesting the retrieval of a record from a result set, the client may supply a database schema identifier, element specification, and record syntax identifier.

For the purpose of retrieving records from a result set, associated with each database are one or more schemas. A schema represents a common understanding shared by the client and server of the information contained in the records of the database, to allow the subsequent selection of portions of that information via an element specification.

A schema defines an abstract record structure which, when applied to a database record results in an abstract database record, which is an abstract representation of the information in the record. An element specification applied to an abstract database record result in another instance of the abstract database record (the latter may be a null transformation). The element specification selects elements from the abstract database record, and may also specify variant forms for those elements.

The server applies a record syntax to an abstract database record, resulting in an exportable structure referred to as a retrieval record.

3.1.6 Model of a Result Set

Logically, a result set is an ordered list of items, each of which is a pointer to a database record; it is used as a selection mechanism for the transfer of database records identified by a query. A result set itself is considered to be a purely local data structure and is not transferred (that is, records are transferred, but not the local pointers to the records).

In general, it is assumed that query processing does not necessarily require physical access to records; a result set is thus assumed to be the identification of (e.g., pointers to) records, as opposed to the actual set of records, selected by a query.

It is important to distinguish the physical implementation from the abstract model. How a server chooses to implement result sets is an implementation matter; a result set may be a copy of the

database records, a table of pointers, or there may not even be a physical result set (the server might execute the query every time the result set is referenced, package up and send the requested records, and otherwise immediately discard the results; however, the result set model does require that a result continue to refer to the same records in the same order). But from the client point of view (and the abstract model) the result set is a set of items, or vectors, where each includes a database name and a pointer to a record within the database.

3.1.6.1 Concurrency and Update Considerations

It is not assumed that the database records are locked. Methods of concurrency control, which would prevent modification or deletion of result set records, are not addressed by this standard.

The server could potentially modify a record pointed to by a result set. For example suppose the client requests record 1, and then subsequently requests record 1 again; the second time, the record may have changed. There is no direct mechanism provided by Z39.50 to prevent this (though Extended Services might be used to lock a record). When a client does modify a record referenced by a result set, then when the client subsequently requests the record, the server might refuse to supply it and instead supply a surrogate diagnostic, but there is no such requirement.

A successful search (see 3.2.2.1.10) results in the creation of a result set. However, a server may unilaterally delete a result set at any time for no specified reason; the next time that the client attempts to refer to that result set (for example in a Present request) the server might send a diagnostic to the effect "result set does not exist". (The server may instead send a diagnostic to the effect that "the result set no longer exists because it was unilaterally deleted", however, the server is not obliged to do so and may simply take the position that the result set never existed.) Therefore, although in the abstract a successful search results in the creation of a result set, as a practical matter, if a server does not actually create a result set, it is not violating the Z39.50 protocol.

3.1.6.2 Order of Result Set Records

The records in a result set are not necessarily ordered according to any specific or predictable scheme (although, whatever the order, it is assumed static). Thus assume for example there is a result set of 100 records, and the client wants the three most recent (i.e. based for example on 'date of publication'). There is no simple way to find those records, short of retrieving all the records in the result set. The Z39.50 Sort service however allows the client to request that a particular result set be sorted based on a specific key (e.g. ‘date of publication’, descending). If the server supports the sort service (and also supports sorting on the requested key, in this example, ‘date of publication’) then following the Sort, the client may subsequently retrieve the first three records, and they will be the three most recent.

3.1.6.2 Logical Structure of a Result Set

For the purpose of retrieving records, the logical structure of a result set is that of a named, ordered list of items. Each item is a triple consisting of:

(a) An ordinal number corresponding to the position of the triple in the list

(b) A database name

(c) A unique identifier (of local significance only) of a record within the database named in (b)

A result set item is referenced by its position within the result set, that is, by (a).

For the purpose of searching, when a result set is used as an operand in a query, the logical structure is one of the following:

Basic model	A set of pairs, each consisting of (b) and (c) of the above model for retrieval.
Extended model	A set of triples, each consisting of (b) and (c) of the retrieval model; and including unspecified information associated with each record, which may be used as a surrogate for the search that created the result set.

Note: Query specifications may indicate that the basic model applies, or under what condition the extended model applies and the nature of the unspecified information. For the type-1 query, when version 2 is in effect, the basic model applies.

3.1.7 Model of Extended Services

The family of Z39.50 services includes the Extended Services (ES) service. "Extended services" refers to a class of services recognized by this standard, but which are not Z39.50 services (as described in 3.1.1). The ES service ***is*** a Z39.50 service, and an ES operation results in the initiation of an extended services task. The ***task*** is ***not*** considered part of the Z39.50 ES operation.

An ES operation is initiated by the client, via an ES request. The ES response, which completes the operation, does not (necessarily) signal completion of the task; it may indicate for example that the task has started or is queued (or it might indicate that the task has been completed; in fact the ES request may specify that the task should be completed prior to the ES response). An ES task may have a lifetime beyond the Z-association.

Examples of extended services are: saving a result set or query, and exporting or ordering a document.

Each ES task is represented by a database record, called a task package, maintained by the server in a special database, the "extended services database". The client uses the ES request to cause creation of a task package on the ES database. The database may be searched, and records retrieved, by the Z39.50 Search and Retrieval facilities. The client may search for packages of a particular type, or created by a particular user, or of a particular status (i.e. pending, active, or complete), or according to various other criteria. In particular, the client may search the database after submitting an ES request (during the same or a subsequent Z-association), for a resulting task package, to determine status information pertaining to the task, for example, to determine whether the task has started.

3.1.8 Explain

The client may obtain details of the server implementation, including databases, attribute sets, diagnostic sets, record syntaxes, and element specifications supported. The client obtains these details through the Z39.50 Explain facility. The server maintains this information in a database that the client may access via the Z39.50 Search and Present facilities.

This "explain" database appears to the client as any other database supported by the server, but it has a well-known name and a pre-defined record syntax. Also, certain search terms, corresponding to information categories, are predefined in order to allow a semantic level of interoperability. Each information category has its own record layout, and all are included in the Explain syntax.

3.2 Facilities of the Information Retrieval Service

Sections 3.2.1 through 3.2.11 describe the eleven facilities of the Information Retrieval service. Most consist of logical groups of services; in several cases, a facility consists of a single service. Additional services may be added to any facility in future versions of this standard. Following is a summary description of the eleven facilities.

Initialization Facility

Init Service: A confirmed service invoked by the client to initiate an Init operation.

Search Facility

Search Service: A confirmed service invoked by the client to initiate a Search operation.

Retrieval Facility

The Retrieval facility consists of two services:

- Present Service: A confirmed service invoked by the client to initiate a Present operation.
- Segment Service: A non-confirmed service initiated by the server, during a Present operation.
  Note: a Present operation thus consists of a Present request followed by zero or more Segment requests followed by a Present response.

Result-set-delete Facility

Delete Service: A confirmed service invoked by the client to initiate a Delete operation.

Browse Facility

Scan Service: A confirmed service invoked by the client to initiate a Scan operation.

Sort Facility

The Sort facility consists of two services:

- Sort Service: A confirmed service invoked by the client to initiate a Sort operation.
- Duplicate Detection Service: A confirmed service invoked by the client to initiate a Duplicate Detection operation.

Access Control Facility

Access-control service: A confirmed service initiated by the server. It does not initiate an operation, and it might or might not be part of an active operation.

Accounting/Resource Control Facility

The Accounting/ Resource Control facility consists of three services:

- Resource-control Service: A conditionally-confirmed service initiated by the server. It does not initiate an operation, and it might or might not be part of an

active operation.

- Trigger-resource-control Service: A non-confirmed service initiated by the client during an operation.
- Resource-report Service: A confirmed service invoked by the client to initiate a Resource report operation.

Explain Facility — The Explain facility does not include any services, but uses the services of the Search and Retrieval facilities.

Extended Services Facility — Extended-services Service: A confirmed service invoked by the client to initiate an Extended-services operation.

Termination Facility — Close Service: A confirmed service initiated by the client or server. It does not initiate nor is it part of any operation. It allows a client or server to abruptly terminate all active operations and to initiate termination of the Z-Association. (Following termination of the Z-Association the client may subsequently attempt to initialize another Z-Association using the Init service.)

3.2.1 Initialization Facility

The Initialization facility consists of the single service, Init.

3.2.1.1 Init Service

The Init service enables the client to establish a Z-association. In the Init request, the client proposes values for initialization parameters. In the Init response, the server responds with values for the initialization parameters; those values, which may differ from the client-proposed values, are in effect for the Z-association.

If the server responds affirmatively (Result = 'accept'), the Z-association is established. If the client then does not wish to accept the values in the server response, it may terminate the Z-association, via the Close service (and may subsequently attempt to initialize again). If the server responds negatively, the client may attempt to initialize again.

Parameters of the Init Service

Parameter Name	Client Request	Server Response	Reference
Version	m	m	3.2.1.1.1
Id/authentication	o		3.2.1.1.2
Options	m	m	3.2.1.1.3
Preferred-message-size	m	m	3.2.1.1.4
Exceptional-record-size	m	m	3.2.1.1.4
Result		m	3.2.1.1.5
Implementation-id	o	o	3.2.1.1.6

Parameter Name	Client Request	Server Response	Reference
Implementation-name	o	o	3.2.1.1.6
Implementation-version	o	o	3.2.1.1.6
User-information-field	o	o	3.2.1.1.7
Other-information	o	o	3.2.1.1.8
Reference-id	o	ia	3.4

Key:

- **m** Mandatory
- **o** Optional
- **ia** If applicable

3.2.1.1.1 Version

Both the client and server indicate all versions that they support. The highest common version is selected for use, and is said to be 'in force,' for the Z-association. If there are no versions in common, the server should indicate 'reject' for the parameter Result.

Notes:

1. Version numbers higher than the highest known version should be ignored.
2. Versions 1 and 2 are identical. Systems supporting version 2 should indicate support for version 1 as well, for interoperability with systems that indicate support for version 1 only. (**Note:** version 1 is not defined by this standard, however there may be implementations that indicate support for version 1, based on other, obsolete standards.)

3.2.1.1.2 Id/Authentication

The client and server agree, outside the scope of the standard, whether or not this parameter is to be supplied by the client, and if so, to the value. This value is used by the server to determine if the client is authorized to enter into communication with the server.

3.2.1.1.3 Options

For each of the capabilities (represented by an “option bit”) in the table below, the client proposes either 'on' or 'off' (meaning 'in effect' or 'not in effect' respectively) and the server responds correspondingly for each. The response determines whether the capability is in effect.

Option Bit	Option	Description
0	Search	See Note 1
1	Present	See Note 1
2	Delete Result Set	See Note 1
3	Resource Report	See Note 1
4	Trigger Resource Control	See Note 2
5	Resource Control	See Note 3

Option Bit	Option	Description
6	Access Control	See Note 3
7	Scan	See Note 1
8	Sort	See Note 1
9	Unused	
10	Extended Services	See Note 1
11	Level 1 Segmentation	See Note 4
12	Level 2 Segmentation	See Note 4
13	Concurrent Operations	See Note 6
14	Named Result Sets	See Note 5
15	Encapsulation	See 4.3
16	resultCount parameter in Sort Response	See Note 8
17	Negotiation Model	See Note 9
18	Duplicate Detection	See Note 1
19	Query type 104	See Note 10
20	PeriodicQuery ES Correction	See Note 11
21	Use of string values for schema in compSpec	See Note 12

Notes:

1. For each Z39.50 operation type -- search, present, delete, resource-report, scan, sort, extended-services, duplicate detection -- the client indicates that it may choose to initiate operations of that type by setting the value for that type to ‘in effect’; if so, the server indicates whether it is willing to process an operation of that type. If the client proposes 'not in effect' for a particular operation type, the server must also specify 'not in effect'.
2. The client may propose to submit Trigger-resource-control requests; if so, the server indicates whether it will accept Trigger-resource-control requests. If the client proposes 'not in effect,' the server must also specify 'not in effect'. If the server specifies 'in effect' for Trigger-resource-control, but 'not in effect' for 'resource-control,' then the client may use only the Cancel function of Trigger-resource-control. The server may indicate unwillingness to accept Trigger-resource-control requests even if it specifies 'in effect' for 'resource-control’. The server's indication of willingness to accept Trigger-resource-control requests does not imply that the server will take any action as a result of a Trigger-resource-control request.
3. The client indicates whether it proposes to permit the server to invoke Resource-control and/or Access-control (i.e. send Resource-control and/or Access-control requests). The server specifies that it may choose to (or will not) invoke Resource-control and/or Access-control. If the server specifies 'not in effect' for resource-control (or access-control) then it will not invoke resource-control (or access control) even if the client has proposed 'in effect’. If the client proposes 'not in effect' for resource-control, and the server indicates 'in effect' for resource-control, indicating that it is not willing to suppress Resource-control requests, and if indeed the client cannot accept Resource-control requests, the client should terminate the Z-association. If the client

proposes 'not in effect' for access-control, and if security requirements on the server system mandate that security (other than that which might be provided by the parameter Id/authentication) be invoked at the outset of a Z-association, then the server should reject the Z-association (by setting the parameter Result to 'reject,' and specifying 'in effect' for 'access-control'). However, security may be invoked at different levels. In addition to authentication at the outset of a Z-association, security might be invoked to control access to a particular database, record, result-set, resource-report format, or use of an operation. Thus if the client proposes 'not in effect' for access-control, and the server normally invokes security (other than at the Z-association level), the server need not necessarily reject the Z-association. The server might wish to invoke a security challenge during an Init operation to determine whether the client is authorized to use a capability it has proposed. If the client has proposed 'not in effect' for access-control, the server may simply refuse the use of that particular operation via the Options parameter. If the client proposes 'not in effect' for access-control, and the server chooses to accept the Z-association, and if the client subsequently initiates an action that would precipitate an Access-control request (for example, if the client issues a Search specifying a database for which it has not yet established credentials), the server should suppress the Access-control request and instead respond with an error status indicating that a security challenge was required but could not be issued.

4. The client proposes one of the following:

    1. "no segmentation", by specifying 'not in effect' for both level 1 and level 2 segmentation;
    2. "level 1 segmentation", by specifying 'in effect' for level 1 and 'not in effect' for level 2 segmentation; or
    3. "level 2 segmentation", by specifying 'in effect' for level 2 segmentation.

    If the client proposes 'in effect' for level 2 segmentation then it may also propose 'in effect' for level 1 segmentation to indicate that if the server is unable to support level 2 segmentation, the client wishes level 1 segmentation to be in effect.” Segmentation" is said to be 'in effect' if either level 1 or level 2 segmentation is in effect. Segmentation may be in effect only when version 3 is in force. The server response indicates which, if either, form of segmentation it intends to perform. If the server specifies neither level 1 nor level 2 then 'no segmentation' is in effect, regardless or what the client has proposed. If the server specifies level 1 (but not level 2) segmentation, it will not perform level 2 segmentation, and the client must be prepared to accept level 1 segmentation, regardless of what the client has proposed. If the server specifies level 2 segmentation, the client must be prepared to accept level 2 segmentation regardless of what it has proposed (the server value for level 1 should be 'not in effect'). When 'no segmentation' is in effect, the server response to a Present request must consist of a single message (a single "segment", i.e. a Present response only, with no intervening Segment requests), containing an integral number of records. When 'Level 1 segmentation' is in effect the server may respond to a Present request with multiple segments (i.e. a Present response, with possibly one or more intervening Segment requests); each must contain an integral number of records. When 'Level 2 segmentation' is in effect the server may respond to a Present request with multiple segments, and individual records may span segments. Segmentation procedures are

detailed in section 3.3.

5. The client may propose to be permitted to use named-result sets (i.e. to specify result set names other than "default" as the value of Result-set-name within a Search request); if so, the server specifies whether it will support named-result-sets. If the client proposes 'not in effect,' the server must also specify 'not in effect'.
6. The client may propose to be permitted to initiate concurrent operations; if so, the server indicates whether it will accept concurrent operations. If the client proposes 'not in effect,' the server must also specify 'not in effect'.
7. Note 7 Deleted.
8. Rules for negotiation of resultCount parameter in Sort Response are as follows:
   - If the client sets bit 8 (proposing that it wishes to initiate Sort operations) then it may also set bit 16 indicating that it supports the resultCount parameter in the response.
   - The client must not set bit 16 if it does not set bit 8.
   - If the client does not set bit 16, then the server must not set bit 16 under any circumstance.
   - If the client sets bit 8 and not bit 16, and if the server sets bit 8, accepting the proposal that Sort operations may be submitted (the server must not set bit 16 as noted above), the server must not include the resultCount parameter in a Sort response.
   - If the client sets bits 8 and 16, and the server also sets bits 8 and 16, the server may include the resultCount parameter in Sort responses. However, the server, by setting bit 16, does not obligate itself to supplying the resultCount parameter.
   - If the client sets bits 8 and 16, and the server sets bit 8 but not bit 16, the server indicates that it will accept Sort requests, but will not include the resultCount parameter in a Sort response.
9. When the client sets the "negotiation model" option bit, it signifies adherence to the negotiation model. If the client and server both set the option bit (in the InitRequest and Response respectively) both may assume that negotiation is carried out in accordance with the model.

   If the client sets this option bit and the server does not, the client should assume that negotiation has not been carried out in accordance with the model.

   If the client does not set this option bit, but the server requires that negotiation be carried out in accordance with this model, the server may reject the Z-association and supply diagnostic 1055: "negotiation option required".

   An option bit corresponding to the negotiation model is necessary for this reason. Suppose a server is unaware of the negotiation model and suppose further that that server routinely echos in the InitResponse all of the information supplied in the InitRequest. This behavior may lead the client to believe (falsely) that negotiation has been carried out. However, because the server is unaware of the negotiation model, it will not set that bit, and so the client will know that the negotiation model is not in effect.
10. When the client sets the "query type 104" bit it proposes to submit queries of type 104. If the server also sets this bit, then it will accept queries of type 104 -- this means the server will not consider it to be a protocol error if the client submits a type 104 query; it does not mean that the server agrees to support any specific external query-definition.
11. If this option is negotiated, then in the event that the PeriodicQuery Extended Service is

used, the definition in this standard will be in effect; otherwise the definition in Z39.50-1995 will be in effect:

- databaseNames must not occur in ClientPartToKeep if option bit 20 is set,
- databaseNames must not occur in ClientPartNotToKeep unless option bit 20 is set,
- additionalSearchInfo must not occur in ClientPartNotToKeep unless option bit 20 is set,
- databaseNames must occur in ServerPart if bit 20 is set and must not occur if option bit 20 is not set,
- lastQueryTime and lastResultNumber are optional if bit 20 is set and mandatory otherwise,
- additionalSearchInfo must not occur in ServerPart unless bit 20 is set.

12. If this option is negotiated, then Schema within Specification (in the ASN.1 APDU definitions) is defined as in this version of the standard, that is it may take on the additional choice of a string. Otherwise it must be an object identifier.

3.2.1.1.4 Preferred-message-size and Exceptional-record-size

The Init request contains the client's proposed values of Preferred-message-size and Exceptional-record-size, specified in bytes. The Init response contains the Preferred-message-size and Exceptional-record-size that the server is going to use; these may be different from (and override) the values proposed by the client. For both the request and response, Preferred-message-size must be less than or equal to Exceptional-record-size.

Exceptional-record-size is meaningful during a Present operation, and only in the special case when a single, exceptionally large record (i.e. larger than preferred-message-size) is requested in the Present request. In this special case, preferred-message-size may be overridden (for the present operation), so that a single record may be presented whose size may be as large as Exceptional-record-size. The fact that a single record is requested is how the client signals that preferred-message-size may be overridden. Thus Exceptional-record-size must be greater than or equal to preferred-message-size. In the case where they are equal, Exceptional-record-size has no meaning (this is the way to signify that the special case will not apply during the Z-association).

If the client supplies values of zero for both parameters, then the client is explicitly indicating "no preference". If the server responds with zero for both parameters (regardless of what the client proposed), the server is indicating that the client must be prepared to accept arbitrarily large records and arbitrarily large messages.

Note that "message size", as in Preferred-message-size, etc., means the sum of the sizes (in bytes) of the response records (retrieval and diagnostic records) in the Present or Search Response APDU. As a practical matter this standard does not prescribe that message sizes be strictly enforced; thus if the negotiated Preferred-message-size is 32767, the server does not need to ensure that the message size does not exceed that value, but rather that it not exceed it substantially. Moreover, message size is the sum of the sizes of the records not including protocol control information. However, this standard does not attempt to distinguish what is protocol control information versus record content. Thus, if the server thinks it sent 32,767 bytes, but the client thinks it sent 32,770 bytes (because the server considered something to be data that the client thought was protocol control information) the server complies in spirit, and the client

should be forgiving and not consider this to be a protocol error. It is recommended that the client propose a value of Preferred-message-size that is less than the actual limit that the client can support.

The use of these parameters is detailed in 3.3.

3.2.1.1.5 Result

The server indicates whether or not it accepts the Z-association by specifying a value of 'accept' or 'reject' in the parameter Result. (If 'reject' is indicated, the client may send another Init request.)

3.2.1.1.6 Implementation-id, Implementation-name, and Implementation-version

The request or response may optionally include any of these three parameters. They are, respectively, an identifier (unique within the client or server system), descriptive name, and descriptive version, for the client or server implementation. These three implementation parameters are provided solely for the convenience of implementers, for the purpose of distinguishing implementations.

3.2.1.1.7 User-information-field

This parameter may be used by the client or server for additional information not specified by this standard.

3.2.1.1.8 Other-information

This parameter may be used by the client or server for additional information not specified by the standard. This parameter may be used only if version 3 is in force.

Note*:* The use of Other-information during initialization (i.e. within the Init request or response, but particularly in the request) is not recommended, because of the uncertainty of what protocol version, 2 or 3, is in force during initialization See Appendix USR (USR.2 Use of Init Parameters for User Information).

3.2.1.1.9 Reference-id

See 3.4.

3.2.2 Search Facility

The Search facility consists of the single service, Search.

3.2.2.1 Search Service

The Search service enables a client to specify a query to be applied to databases at a server system, and to receive information about the results of the query.

The search request allows the client to request that the server apply a query to a specified set of databases at the server, to identify records with the properties indicated by the query. The server

creates a result set, which represents the set of records identified by the query. The result set is an ordered set; a record identified by an entry in the result set is referenced by the position of the entry within the result set (beginning with 1). The server maintains the result set for subsequent retrieval requests.

Depending on the parameters of the search, one or more records identified by the result set may be immediately retrieved as part of the search response. (This is referred to informally as "piggybacking". Retrieval is in general a function of the Present service but in special cases may be carried out as part of the Search operation.)

Parameters of the Search Service

Parameter Name	Client Request	Server Response	Reference
Query-type	m		3.2.2.1.1
Query	m		3.2.2.1.1
Database-names	m		3.2.2.1.2
Result-set-name	m		3.2.2.1.3
Replace-indicator	m		3.2.2.1.3
Small-set-element-set-names	o		3.2.2.1.4
Medium-set-element-set-names	o		3.2.2.1.4
Preferred-record-syntax	o		3.2.2.1.5
Small-set-upper-bound	m		3.2.2.1.6
Large-set-lower-bound	m		3.2.2.1.6
Medium-set-present-number	m		3.2.2.1.6
Response-records		ia	3.2.2.1.7
Result-count		m	3.2.2.1.8
Number-of-records-returned		m	3.2.2.1.9
Next-result-set-position		m	3.2.2.1.10
Search-status		m	3.2.2.1.11
Result-set-status		ia	3.2.2.1.11
Present-status		ia	3.2.2.1.11
Additional-search-information	o	o	3.2.2.1.12
Other-information	o	o	3.2.2.1.13
Reference-id	o	ia	3.4

3.2.2.1.1 Query-type and Query

The parameter Query-type identifies the type of query, i.e. the syntax of parameter Query. Six types are defined:

- Type-0 may be used only when the client and server have a priori agreement outside of the standard.
- Type-1 is the Reverse Polish Notation (RPN) query (so-called because RPN is sometimes

used as an abstract representation; note however that RPN is not used as an encoding of the Type-1 query). It is specified in 3.7.

- Type-2 is the ISO8777 type query, specified in ISO 8777.
- Type-100 is the Common Command Language query; its syntax is not specified by this standard.
- Type-101 is the extended RPN (ERPN) query; an extension to the type-1 query to allow proximity searching and restriction of result sets by attributes. It is specified in 3.7.
  Note: The type-101 query is identical to the type-1 query with the following exception: For type-1, proximity and restriction are valid only when version 3 is in force. For type-101, proximity and restriction are valid both for version 3 and version 2 as well. (The definition of the type-101 query is independent of version.)
- Type-102 is the Ranked List query, to be defined in a later version of this standard.
- Type-104 is used to identify externally defined queries.

3.2.2.1.2 Database-names

The client indicates the set of databases to which the Query applies.

Notes:

1. **Database Combinations**. The server designates (through the Explain facility or through some mechanism outside of the standard) what databases may be specified on a Search request, and in what combinations they may be specified. For example, a server might specify that databases **A**, **B**, and **C**, may be searched individually, and that **A** and **B** may be searched in combination (but not **A** and **C**, nor **B** and **C**).
2. **Multi-database Searching.** Z39.50 allows but does not require support for, multi-database searching in a single Search request. A client may simply choose not to send Search requests that list more than a single database. A server, upon receipt of a multi-database search request may simply fail the search (diagnostic 111: "Too many databases specified"; with an addinfo "maximum value" 1 supplied). A server that chooses to support multi-database search request, may limit such support to any specific combinations it chooses. For example suppose a server provides 100 databases, named "1", "2", ... , "100". It may declare that each database may be searched singly, and databases "1" and "2" may be searched together. (It may declare this via Explain, or, more dynamically, when it receives a search with an unsupported combination of databases, it may fail the search with diagnostic 23: "Specified combination of databases not supported".) A server may define virtual databases corresponding to supported combinations rather than support multi-database search requests. For example, suppose there are 3 real databases (A, B, and C) to which a server will allow access in all possible combinations; the server might expose 7 virtual databases (A, B, C, AB, AC, BC, and ABC), thus in effect providing all possible combinations but allowing only a single (virtual) database to be specified in a Search request.
3. **No Default Database.** There is no defined concept in Z39.50 of a “default” database. If a server considers a particular database to be the default database, it could express that fact via Explain (using the “description” field of DatabaseInfo) however that information would be for human consumption only.
4. **Case Insensitive.** Each database name specified by the server is a string of

characters, and the string is case-insensitive. Thus for any character that is a letter, the client may use either upper or lower case, regardless of how the server has specified the name.

3.2.2.1.3 Result-set-name and Replace-indicator

The parameter Result-set-name specifies a name to be given to the result set (to be created by the query) so that it may be subsequently referenced (within the same Z-association).

If a result set with the same name already exists at the server, the action taken depends on the value of the parameter Replace-indicator, as follows:

- If the value of Replace-indicator is 'on,' then after processing the query, the existing result set whose name is specified by the parameter Result-set-name will be deleted, and a new result set by that name created. If the search cannot be processed, the content of the result set will be empty.
- If the value of Replace-indicator is 'off,' the search is not processed, an error diagnostic is returned by the server, and the existing result set whose name is specified by the parameter Result-set-name is left unchanged.

If a result set does not exist with the name specified by the parameter Result-set-name, then a result set by that name is created by the server, and the parameter Replace-indicator is ignored. The initial content of the result set is empty. If no records are found by the query, the result set remains empty.

A server need not support, in general, the naming of result sets by the client (see Notes below.). However, a server must support at least the result set whose name is "default." If the client specifies "default" as Result-set-name, then Replace-indicator must be 'on'.

A result set created by a Search request (that is, specified by the parameter Result-set-name) may be referenced in a subsequent Present request or as an operand in a subsequent Search request (for example, in a type-1 query). If a result set named "default" is created, it remains available for reference from the time it is created until the end of the Z-association during which it is created, or until either:

- Another default result set is created, because the name "default" is specified as Result-set-name in a subsequent Search request
- It is unilaterally erased or deleted by the server

Any result set other than the result set named "default" remains available for reference from the time it is created until it is deleted in one of the following ways:

- By a Delete operation
- Implicitly, because a result set was specified by the same name in a Search request, and the value of the parameter Replace-indicator was 'on'
- Unilaterally by the server (at any time)
- By termination of the Z-association

Notes:

1. A server need not support, in general, the naming of result sets by the client. The server may always choose to:

(a) Fail the search, supplying diagnostic 22 (Result set naming not supported).

Alternatively, in certain circumstances (detailed below) the server may (but need not):

(b) Treat the condition as a protocol error: issues a Close, with Close-reason 'protocol error', if version 3 is in effect, or terminate the connection if version 2 is in effect.

Result set naming is a negotiated feature in thus a server who doesn't support it should so inform the client during initialization (via option bit 14). If so, then a subsequent attempt by the client to name a result set (i.e. assign a name other than 'default') may be construed by the server to constitute a protocol error.

However the mechanism, by which the server informs the client that result set naming is not suported, is not reliable unless either version 3 is in effect or if the client sets bit 14 during initialization (implicitly acknowledging that it knows the meaning of this option bit). Z39.50-1992 did not define this option bit, so by 1992 interpretation, the condition is not a protocol error. If version 2 is in force, the client may have implemented Z39.50-1992. If version 3 is in force, the server may be sure that the client has implemented Z39.50-1995 or later.

In general, the server may always choose behavior (a) above, and more specifically: If version 2 is in force, and if the client did not attempt during initialization to negotiate result set naming (via option bit 14), the server should choose behavior (a). If version 2 is in force, but the client did attempt during initialization to negotiate result set naming, and the server rejected that option, the server may choose (a) but may alternatively choose (b).

If version 3 is in force, and the server had set bit 14 off indicating that result set naming is not supported (regardless of whether the client set bit 14), then similarly, the server may choose (a) but may alternatively choose (b).

2. Result set names are ***case sensitive***. This is in contrast to database names, which are case-***in***sensitive (see 3.2.2.1.2 Database-names note 4). Database names are passed around outside of the protocol, often by humans, and transcend Z-associations. Result set names have none of these characteristics; they are referred to only by the client (the mapping between user specified names and actual result set names used in the protocol are not within the scope of the Z39.50 protocol), within a Z-association, and the client is expected to use a given result set name consistently within a Z-association. (Element set names are case-***in***sensitive; see 3.6.2.)

3.2.2.1.4 Small-set-element-set-names and Medium-set-element-set-names

These parameters describe the preferred composition of the records expected in the search response (see 3.2.2.1.6). If the query results in a small-set then Small-set-element-set-names pertains. If the query results in a medium-set, then Medium-set-element-set-names pertains.

3.2.2.1.5 Preferred-record-syntax

The client may specify a preferred record syntax for retrieval records.

See 3.2.3.1.5.

3.2.2.1.6 Small-set-upper-bound, Large-set-lower-bound, and Medium-set-present-number

The result set is considered a "small-set," "medium-set," or "large-set," depending on the values of parameters Small-set-upper-bound and Large-set-lower-bound of the Search request, and Result-count of the Search response (see 3.2.2.1.8). The result set is a small-set if Result-count is not greater than small-set-upper-bound. The result set is a large-set if Result-count is larger than or equal to Large-set-lower-bound. Otherwise, the result set is a medium-set. If the query results in a small-set, response records corresponding to all database records identified by the result set are to be returned to the client (subject to possible message size constraints). If the query results in a large-set, no response records are to be returned. If the query results in a medium-set, the maximum number of response records to be returned is specified by Medium-set-present-number.

Notes:

1. The result set may be a medium-set only when Result-count is greater than small-set-upper-bound and less than Large-set-lower-bound, and this can occur only if Large-set-lower-bound is at least 2 greater than Small-set-upper-bound; i.e. the result set cannot be a medium-set if Large-set-lower-bound exceeds Small-set-upper-bound by 1. For example, if Large-set-lower-bound is 11 and Small-set-upper-bound is 10, the intent is "if 10 or less database records are found, return response records for them all, otherwise do not return any," and medium-set-present-number would not apply.
2. Small-set-upper-bound may be zero. Large-set-lower-bound must be greater than Small-set-upper-bound.
3. If the client does not want any response records returned regardless of the value of Result-count, Large-set-lower-bound should be set to 1 and Small-set-upper-bound to zero.

3.2.2.1.7 Response-records

The server processes the search, creating a result set that identifies a set of database records. It cannot be assumed however that search processing requires physical access to the database records. A particular database record might not be accessible but this circumstance might not be recognized until an attempt is made to access the record for the purpose of forming a retrieval record.

After processing the search, the server attempts to create retrieval records to be included in the Search response, corresponding to the first N database records identified by the result set (N depends on the request parameters and Result-count, as described in 3.2.2.1.6). For each database record for which a retrieval record cannot be included, a surrogate diagnostic record is substituted.

The term response record refers to a retrieval record or a surrogate diagnostic record. The parameter Response-records is one of the following:

- N response records
- A number of response records, which is less than N because of message size constraints (see 3.3)
- One or more non-surrogate diagnostic records (see note) indicating that the search cannot be processed, and why it cannot be processed

- One or more non-surrogate diagnostic records (see note) indicating that records cannot be presented, and why not, e.g. "element set name not valid for database"
  Note: If version 2 is in force, the server returns a single non-surrogate diagnostic record. If version 3 is in force, the server returns one or more non-surrogate diagnostic records.

The order of occurrence of response records within the parameter Response-records is according to the order in which they are identified by the result set. Each may optionally be accompanied by the name of the database in which the record resides. However, the database name must accompany the first response record being returned, and must accompany any record from a database different from its immediate predecessor. The database name appended to a record need not be one of the database names that was included in the original search request (the request that caused the creation of the result set from which the record is being presented).

When Server Does Not Support Response Records in a Search Response

If a server does not support “piggybacking”, i.e. supplying response records in a Search response, and when the Search request parameter values for small-set-upper-bound and large-set-lower-bound and the Search response parameter value for Result-count are such that the response is expected to include one or more records as specified in 3.2.2.1.7 (for example, if a search results in 5 records and small-set-upper-bound on the search request was 10, the server is expected to attempt to return all five records), the server should respond with a Search-status of 'success' and a Present-status of 'failure', and include a non-surrogate-diagnostic, for example General Diagnostic “1005: Response records in Search response not supported”, or “1006: Response records in Search response not possible for specified database or database combination”.

Note that Diagnostic 1006 is intended for the case of an intermediary providing access to multiple servers, some of which may support piggybacking and some which do not, and is for the intermediary to use in case the particular end server does not support piggybacking, to distinguish from the case where the intermediary itself does not support piggybacking.

When Query is not supported for a Database

When the query is not supported for one of the databases (that is, when there is at least one database listed by the Client in the Database-names parameter, 3.2.2.1.2, for which the query is not supported, even though the query may be supported for other listed databases), the server should set 'search-status' to ‘failure’, and supply an appropriate diagnostic. (For example, when an attribute is not supported for a database, supply diagnostic 1056: "Attribute not supported for database". Multiple instances of diagnostic 1056 may be supplied for multiple attribute/database combinations.)

When a query is supported for one or more but not all of the databases, and if the server wishes to make the result set available to the client (the result set produced by applying the query to the subset of supported databases) the server may set Search-status to ‘failure’ and Result-set-status to ‘subset: partial, valid results available’. The reasoning for this prescribed behavior is as follows. Setting Search-status to 'success' would give the client the false impression that the search was executed across all of the databases. For those databases for which the query is not supported, the client would be led to believe instead that the query is indeed supported and that no records were found. But while a status of 'success' is misleading when part of the search fails, 'failure', too, is misleading when part of the search succeeds. Thus the parameter Result-set-status helps disambiguate a 'failure' status. Furthermore,

Result-set-status may be supplied only when Search-status is 'failure'. Thus, setting status to 'failure' is prescribed, not because of any implied semantics of a failure status but (more pragmatically) because it provides the client with access to the result set.

3.2.2.1.8 Result-count and Number-of-records-returned

The parameter Result-count is the number of database records identified by the result set. If the result set is empty, result-count is zero.

A negative value for resultCount in a Search Response is invalid and constitutes a protocol violation. A client that receives a negative value of Result-count may treat this as a protocol error (if version 3 is in force, may issue a Close Request with Close-reason ‘protocol error’).

The parameter Number-of-records-returned is the total number of records returned in the Search response, including diagnostic records, surrogate or non-surrogate.

3.2.2.1.9 Next-result-set-position

The parameter Next-result-set-position takes on the value M+1, where M is the position of the result set item which identifies the database record corresponding to the last response record among those returned; or zero if M = Result-count.

3.2.2.1.10 Search-status

The parameter Search-status, returned in the response, assumes one of the following two values:

Success	The search completed successfully.
Failure	The search did not complete successfully.

A value of 'success' does not imply that the expected response records are being returned as part of the response (see Present-status, 3.2.2.1.11). Note also, a value of 'success' does not imply that any database records were located by the search. A value of 'failure' ***does*** imply that ***none*** of the expected response records is being returned. In the latter case, the server returns one or more non-surrogate diagnostic records (see note) indicating that the search cannot be processed.

Note: If version 2 is in force, the server returns a single non-surrogate diagnostic record. If version 3 is in force, the server returns one or more non-surrogate diagnostic records.

3.2.2.1.11 Result-set-status and Present-status

These are status descriptors necessary to distinguish potentially ambiguous situations that can occur during search and present operations.

Result-set-status occurs if and only if the value of Search-status is 'failure,' and its value is one of the following:

Subset	Partial, valid results available.
Interim	Partial results available, not necessarily valid.
None	No result set.

Present-status occurs if and only if the value of Search-status is 'success,' and its value is one of the following:

Success	All of the expected response records are available. See note 1.
Partial-1	Not all of the expected response records can be returned because the request was terminated by access control.
Partial-2	Not all of the expected response records can be returned because they will not fit within the preferred message size.
Partial-3	Not all of the expected response records can be returned because the request was terminated by resource control, at client request. See note 2.
Partial-4	Not all of the expected response records can be returned because the request was terminated by resource control, by the server. See Note 3.
Failure	None of the expected response records can be returned. One or more non-surrogate diagnostic records are returned (see note in 3.2.2.1.7).

Notes:

1. In the case where the Search Request specified Small-set-upper-bound = 0 and Large-set-lower-bound = 1 (i.e. don't present records under any circumstances) and where Search-status = 'success'; Present-status (which must be supplied, since Present-status must occur if Search-status is 'success') should be 'success', where in this case “success” may be interpreted to mean "you asked for no records, and no records were sent, so therefore consider the disposition of the Present part of the request to be successful".
2. If partial-3 is indicated, this implies that either (1) the server sent a resource control request to which the client responded "do not continue", or (2) the client issued a TriggerResourceControl request to terminate the operation. This means that either Resource Control or Trigger Resource control (for cases 1 and 2 respectively) must have been negotiated.
3. Partial-4 may be indicated when the server simply terminates the operation, unilaterally, perhaps because resources at the server are limited (but not as a result of a Resource-control response indicating "do not continue".) Partial-4 does not require that resource control be negotiated.

Notes Pertaining to the Relationship among Search-Status, Present-Status, and Result-Set-Status

Informally, in a Z39.50 Search operation, the client supplies search criteria and requests the server to:

(a) Identify records that meet the criteria

(b) Report the number of records identified

(c) Establish a result set corresponding to those records (see note 1)

(d) (Conditionally) return some or all of the records (see note 3)

Notes:

1. If no records were identified (see note 2), the server might not physically create a result set, but in the abstract, an empty result set is assumed to have been created.
2. The qualification "no records were identified" means that the server performed the search and is stating that "there are no records meeting the criteria"; it does not mean the server was unable to determine the number (in which case the server should fail the search).
3. The number of records returned, which may be zero, depends on the result count and the values of the Search Request parameters Small-set-upper-bound, Large-set-lower-bound, and Medium-set-present-number.

The search operation consists of a search phase and a retrieval phase, where (a), (b), and (c) correspond to the search phase and (d) to the retrieval phase. The search is considered to succeed if and only if the server is able to perform (a), (b), and (c).

Thus if a search results in the identification of zero records (that is, the server determines that no records meet the criteria), this does not constitute failure. (In Z39.50, "search" does not so much connote "locate", as in the classical usage of the term "search", as it means "identify how many and which items meet specified criteria". "None" is a valid answer to "how many", and in that case, the "which" part does not apply.) The search-status values of 'success' and 'failure' correspond respectively to whether the search succeeds or not.

When the search succeeds, a result set is created. If the search fails, a result set may or may not be created, and its existence may be determined by the value of Result-set-status (see table below). Result-set-status occurs if and only if the search fails; if the search succeeds, the existence of the result set is implicitly known (it exists) and therefore the parameter is not necessary.

There is a retrieval phase if and only if the search succeeds, and if so, the response parameter Present-status occurs.

Therefore exactly one of the parameters Result-set-status and Present-status occurs in the response. Result-set-status corresponds to a search that fails, and Present-status to a search that succeeds.

Present-status has values of 'success' or 'failure', similar to Search-status (though, in contrast, Present-status has additional intermediate statuses as well). 'Success' means that the server presented all of the response records (retrieval or surrogate diagnostic records) that it attempted to present; 'failure' means that it was unable to present any of the records (cases where the server presents some, but not all, are covered by the intermediate statuses).

Whenever either Search-status or Present-status is 'failure', the server must provide one or more non-surrogate diagnostics, supplying diagnostic information associated with the failure.

The following table summarizes the relationship among the statuses, the presence of non-surrogate diagnostics, and the existence of a result set.

If Search Status is:	And Present Status is:	Then Result Set Status:	And a Non-surrogate Diagnostic:	And a Result Set:
'success'	'success' (or 'partial')	Must not occur	Must not be supplied	Exists
'success'	'failure'	Must not occur	Must be supplied (at least one)	Exists
'failure'	*(Present Status must not occur)*	Must occur	Must be supplied (at least one)	Exists if result-set-status is 'subset' or 'interim'; Does not exist if 'none'.

3.2.2.1.12 Additional-search-information

On the response the server may use this parameter to convey information, which is a by-product of the search process, including, for example, intermediate result counts, why particular records were returned, or whether a particular attribute was used for searching a database. On the request the client may use this parameter to indicate the preferred format or content of that information. User Information format SearchResponse-1 is defined in Appendix USR. This parameter may be used only when version 3 is in force.

3.2.2.1.13 Other-information

This parameter may be used by either the client or server for additional information not specified by the standard. This parameter may be used only when version 3 is in force.

3.2.2.1.14 Reference-id

See 3.4.

3.2.3 Retrieval Facility

The Retrieval facility consists of two services: Present and Segment.

The client sends a Present request-to-request response records according to position within a result set maintained by the server. The server responds by sending a Present response, containing the requested response records. Alternatively, if segmentation is in effect and the requested response records will not fit within the Present response message, the server may segment the response by sending one or more Segment requests before the Present response. The procedures for segmentation are described in 3.3.

The Segment requests (if any) together with the Present response are referred to as the aggregate Present response. Each Segment request as well as the Present response is referred to as a segment of the Present response. If an aggregate Present response consists of a single segment (i.e. only a Present response) it is called a simple Present response.

3.2.3.1 Present Service

The Present service allows the client to request response records corresponding to database records represented by a specified result set. Database records are referenced by relative

position within the result set. The client specifies a range and may follow with subsequent requests specifying different ranges.

Note: If version 3 is in force, a single request may include more than one range.

The client may request, for example, records one through five and follow with a request for records four through six.

Note: In this section, "record N" means "the response record corresponding to the database record identified by result set entry N."

Parameters of the Present Service

Parameter Name	Client Request	Server Response	Reference
Number-of-records- requested	m		3.2.3.1.1
Result-set-start-position	m		3.2.3.1.1
Additional-ranges	o		3.2.3.1.2
Result-set-id	m		3.2.3.1.3
Element-set-names	o		3.2.3.1.4
Preferred-record-syntax	o		3.2.3.1.5
Comp-spec	o		3.2.3.1.6
Max-segment-count	o		3.2.3.1.7
Max-segment-size	o		3.2.3.1.7
Max-record-size	o		3.2.3.1.7
Response-records		ia	3.2.3.1.8
Number-of-records- returned		m	3.2.3.1.9
Next-result-set-position		m	3.2.3.1.9
Present-status		m	3.2.3.1.10
Other-information	o	o	3.2.3.1.11
Reference-id	o	ia	3.4

3.2.3.1.1 Number-of-records-requested and Result-set-start-position

The client requests a range of records: N records beginning at record M. M = Result-set-start-position, N = Number-of-records-requested.

Note: The 1995 version of this standard assumed that the client always know the size of (i.e. number of records in) the result set, and therefore considered it to be a protocol error if N is greater than (Result-count - M) + 1; that is, if the range requested includes records whose ordinal result set position would exceed the number of items in the result set (for example, requesting result set records 7 through 10 when there are only 8 records in the result set). However, if version 3 is in effect, this need not be considered a protocol error, because the client might not always be expected to know the size of the result set (for example when the result set is sorted,

or de-duplication is performed, or the result set is a re-activated persistent result set). The server may chose to treat it as a protocol error (and may issue a non-surrogate diagnostic, such as diagnostic 13) or may honor the request and supply a surrogate diagnostics for each of the non-existent records (and similarly, diagnostic 13 may be used as a surrogate diagnostic). Note however, if version 2 is in effect, then this condition must be treated as a protocol error.

3.2.3.1.2 Additional-ranges

The client may request additional ranges of records by including this parameter, which consists of one or more pairs (M, N) where M and N are as described in 3.2.3.1.1. For the first pair (M, N) M must be greater than or equal the sum of Result-set-start-position and Number-of-records-requested. For any consecutive pairs (M1, N1) and (M2, N2), M1 + N1 must be less than M2. This parameter may occur only when version 3 is in force.

If the client includes this parameter and the server does not support Additional-ranges, it should fail the Present (Present-status of 'failure' with a non-surrogate diagnostic 243).

3.2.3.1.3 Result-set-id

The client indicates the name of a transient result set, created during this Z-association, from which records are to be retrieved.

3.2.3.1.4 Element-set-names

The client may indicate the desired composition of the retrieved records. See 3.6.2.

3.2.3.1.5 Preferred-record-syntax

The client may specify a preferred record syntax for retrieval records.

3.2.3.1.6 Comp-spec

This parameter may be included only if the parameter Element-set-names is omitted, and only if version 3 is in force. If included, Comp-spec provides an alternative means of specifying the desired composition of retrieved records. See 3.6.

3.2.3.1.7 Max-segment-count, Max-segment-size, and Max-record-size

These three parameters may be used only when version 3 is in force.

Max-segment-count may be included when level-1 or level-2 segmentation is in effect; it specifies the maximum number of segments the server may include in the aggregate Present response. If its value is 1, no segmentation is applied for the operation and Max-record-size should not be included.

Max-segment-size and/or Max-record-size may be included only when level 2 segmentation is in effect. Max-segment-size is the largest allowable segment; if included, it overrides Preferred-message-size (for this Present operation only); if not included it assumes the value of Preferred-message-size. Max-record-size is the largest allowable retrieval record within the aggregate Present response; if included, it must equal or exceed Max-segment-size.

These three parameters are further detailed in 3.3.3.2.

3.2.3.1.8 Response-records

This parameter consists of a sequence of response records, or possibly, if 'level 2 segmentation' is in effect, a final fragment (see 3.3.3) followed by zero or more response records. Alternatively (if the operation included no Segment requests) the parameter consists of one or more non-surrogate diagnostic records indicating that the request cannot be processed, and why not (see note below).

A response record will be returned in the aggregate Present response for each record requested in the request (subject to message size, access-control, and resource-control constraints). Each response record corresponds to a result set entry, and the result set ordinal positions represented by the response records will be ascending and consecutive, unless the request included the parameter Additional-ranges. In this case, the positions will be ascending but may have gaps, which will correspond, exactly to the gaps in the requested ranges.

Each response record may optionally be accompanied by the name of the database to which it corresponds. However, the database name must accompany the first response record (or starting fragment) within the first segment of the aggregate Present response, and must accompany any response record (or starting fragment of a response record) from a database different from its immediate predecessor within the aggregate Present response.

When the client has received the aggregate Present response, the result (if all of the segments are re-assembled, and segmented response records re-assembled from their fragments) will be one of the following:

- N response records, where N = Number-of-records-requested
- A number of response records, which is less than N (reason specified by Present-status)
- One or more diagnostic records (see note) indicating that the request cannot be processed, and why not.

Note: If version 2 is in force, the server returns a single non-surrogate diagnostic record. If version 3 is in force, the server returns one or more non-surrogate diagnostic records.

3.2.3.1.9 Number-of-records-returned and Next-result-set-position

The parameter Number-of-records-returned is the total number of records in the aggregate Present response. Next-result-set-position is the value M+1, where M is the position of the result set item corresponding to the last record among those included in the response; or zero if M is the position of the last result set item.

3.2.3.1.10 Present-status

Present-status is mandatory in a Present response and its values are the same as those listed for Present-status in 3.2.2.1.11. Present-status refers to the aggregate Present response.

3.2.3.1.11 Other-information

This parameter may be used by the client or server for additional information, not specified by the standard. This parameter may be used only if version 3 is in force.

3.2.3.1.12 Reference-id

See 3.4.

3.2.3.2 Segment Service

If the records requested by a Present request will not fit in a single segment, and if segmentation is in effect, the server returns multiple segments, each of which contains a portion of the records. Each except the last segment is returned as a Segment request (the last segment is returned as a Present response).

Notes:

1. The segment service is modeled as a request, even though, logically, the server is not making a request. The reason is that (for purposes of abstract service definition and resultant protocol specification) any message is a request or a response, a response must be preceded by a request of the same type, and there may be at most one response to a given request. Because of these modeling constraints: the Segment service cannot be modeled as a response (because if it were, it would necessarily respond to a segment request, and it is a non-confirmed service); and the present operation cannot be modeled as a Present request followed by multiple Present responses.
2. This service may be used only when version 3 is in force.
3. If segmentation is not in effect, the server does not send any Segment requests and the aggregate Present response consists of a simple Present response. If the records requested will not fit in a segment, the procedures described in 3.3.1 apply.
4. If the records requested will fit in a single segment (whether or not segmentation is in effect) the server does not send any Segment requests and the aggregate Present response consists of a simple Present response.

Parameters of the Segment Service

Parameter Name	Server Request	Reference
Segment-records	m	3.2.3.2.1
Number-of-records-returned	m	3.2.3.2.2
Other-information	o	3.2.3.2.3
Reference-id	ia	3.4

3.2.3.2.1 Segment-records

If level 1 segmentation is in effect, the parameter Segment-records consists of a sequence of response records.

If level 2 segmentation is in effect, the parameter Segment-records may include response records as well as fragments (see 3.3.3). It may be composed of a final fragment (except within the first segment of the aggregate Present response), followed by zero or more response records, followed by a starting fragment. Neither fragment need occur, however if neither occurs there must be at least one response record. (Note that fragments pertain only to retrieval records; a diagnostic record may not be segmented.)

The order of occurrence of a response record or fragment of a retrieval record is according to the order in which the record is identified by the result set. Each response record or starting fragment may optionally be accompanied by the name of the database to which it pertains. However, the database name must accompany the first response record (or starting fragment) within the first segment of the aggregate Present response, and must accompany any response record (or starting fragment of a retrieval record) from a database different from its immediate predecessor within the aggregate Present response.

3.2.3.2.2 Number-of-records-returned

This is the total number of response records and starting fragments included within the segment.

3.2.3.2.3 Other-information

This parameter may be used by the server for additional information, not specified by the standard.

3.2.3.2.4 Reference-id

See 3.4.

3.2.4 Result-set-delete Facility

The Result-set-delete facility consists of a single service, Delete.

3.2.4.1 Delete Service

The Delete service enables a client to request that the server delete specified result sets, or all result sets, created during the Z-association. The server responds by reporting information pertaining to the result of the operation.

Although a result set is deleted automatically after the Z-association is closed. A client may wish to delete a result set explicitly during the Z-association. A server might have a limit to the number of result sets it can maintain, and might unilaterally delete one or more result sets when that limit is approached. Therefore by deleting a result set it no longer needs, a client might forestall the possibility that a result set that it does still need may be unilaterally deleted. Moreover, some servers charge to maintain results, so deleting a result set when it is no longer needed may help reduce cost.

Parameters of the Delete Service

Parameter Name	Client Request	Server Response	Reference
Delete-function	m		3.2.4.1.1
Result-set-list	ia		3.2.4.1.2
Delete-operation-status		m	3.2.4.1.3
Delete-list-status		ia	3.2.4.1.4
Number-not-deleted		ia	3.2.4.1.5
Bulk-statuses		ia	3.2.4.1.5
Delete-msg		ia	3.2.4.1.6
Other-information	O	o	3.2.4.1.7
Reference-id	O	ia	3.4

3.2.4.1.1 Delete-function

The client specifies one of the following:

list	delete specified result sets (see 3.2.4.1.2), or
bulk-delete	delete all result sets currently on the server created during this Z-association.

3.2.4.1.2 Result-set-list

This parameter occurs if and only if Delete-function is 'list'. It contains a list of result sets (created during this Z-association) to be deleted.

3.2.4.1.3 Delete-operation-status

Delete-operation-status is the status of the delete request. It assumes one of the values 'success' or 'failure-3' through 'failure-9' in the table below.

3.2.4.1.4 Delete-list-statuses

Delete-list-statuses is present in a Delete response if Delete-function in the request was 'list'. Delete-list-statuses contains the same list of result sets as in the Result-set-list parameter of the Delete request, each paired with a status. Possible status values are 'success,' 'failure-1' through 'failure-6,' and 'failure-10'.

Status	Description
success	Result set(s) deleted.
failure-1	Result set did not exist.
failure-2	Result set previously unilaterally deleted by server.
failure-3	System problem at server (optional text message may be included in the Delete-msg

	parameter).
failure-4	Access-control failure: the delete request caused the server to issue an Access-control request, which the client failed to satisfy, or the client could not accept an Access-control request.
failure-5	Operation terminated by resource control at client request.
failure-6	Operation terminated by server due to resource constraints.
failure-7	Bulk delete of result sets not supported by server. See Note 1.
failure-8	Not all result sets deleted (on a bulk-delete request) (see 3.2.4.1.5). See Note 1.
failure-9	Not all requested result sets deleted (on a list request).
failure-10	Result-set in use. See Note.2.

Notes:

1. Failure-7 and failure-8 can occur only if Delete-operation is Bulk-delete.
2. Failure-10 may be used only when version 3 is in force.

3.2.4.1.5 Number-not-deleted and Bulk-statuses

These two parameters occur only if Delete-function is Bulk-delete and if Delete-operation-status = 'failure-8'. The parameter Number-not-deleted indicates how many result sets were not deleted, and the parameter Bulk-statuses gives individual statuses for those not deleted.

Note, however, that a server is not obligated to provide statuses for each result set not deleted on a bulk delete. For example, a server may abort a bulk delete when the first failure to delete a result set that is part of the bulk delete fails; in this case only a single status might be included in the parameter Bulk-statuses.

If a bulk delete results in more statuses than can fit into a single Delete-response message, the server may discard those that do not fit.

3.2.4.1.6 Delete-msg

Delete-msg, if present, contains an optional text message.

3.2.4.1.7 Other-information

This parameter may be used by either the client or server for additional information not specified by the standard. This parameter may be used only when version 3 is in force.

3.2.4.1.8 Reference-id

See 3.4.

3.2.5 Access Control Facility

The Access Control facility consists of a single service, Access-control.

3.2.5.1 Access-control Service

The Access-control service allows a server to challenge a client. The challenge might pertain to a specific operation or to the Z-association. The Access-control request/response mechanism can be used to support access control challenges or authentication, including password challenges, public key cryptosystems, and algorithmic authentication.

A client must be prepared to accept and respond to Access-control requests from the server if access control is in effect. A server may issue an Access-control request which is either part of a specific (active) operation, or which pertains to the Z-association.

- If concurrent operations is in effect:
  - If the Access-control request includes a Reference-id: The supplied Reference-id must correspond to an active operation; the Access-control request is part of that operation. The Access-control response must also include that Reference-id.
  - If the Access-control request does not include a Reference-id: The Access-control request and response are not part of any operation, they pertain to the Z-association.
- If serial operations is in effect: The server may issue an Access-control request only when there is an active operation; the Access-control request and subsequent response are part of that operation and must include the Reference-id of the operation (which is assumed 'null' if not present in the initiating request).

The following procedures pertain to access control as it applies to an operation:

1. After sending an initiating request, the client must be prepared to receive an Access-control request (for that operation), respond with an Access-control response, then later receive another Access-control request, etc., before receiving a terminating response. The server might suspend processing of the operation from the time that it sends the Access-control request until it receives the Access-control response. The challenge does not interrupt any other operation. If the client response is acceptable to the server, the operation proceeds as if the challenge has never taken place. If the client fails to respond correctly to the challenge then the server's terminating response to the interrupted operation may indicate that the operation was terminated due to an Access-control failure.
2. If the client fails to respond correctly to a challenge during an Init operation, the server may reject the Z-association (by setting the Result parameter to 'reject,' and optionally supplying an explanatory message in the User-information-field of the Init response). However, the server need not necessarily reject the Z-association. For example the server might wish to invoke a security challenge during an Init operation to determine whether the client is authorized to use a capability it has proposed. If the client fails to respond properly, the server may simply refuse the use of that particular operation (via the Options parameter).
3. During a Search or Present operation, while the server is preparing records for presentation, it might send an Access-control request pertaining to a particular record. If the client fails to respond correctly to the challenge, the server may simply substitute a surrogate diagnostic: "security challenge failed; record not included."

The following procedures pertain to access control as it applies to the Z-association:

1. If concurrent operations is in effect, following initialization the client must be prepared at any time during the Z-association, whether or not operations are active, to receive an

Access-control request pertaining to the Z-association, to respond with an Access-control response, then later to receive another Access-control request, etc.

2. The server might suspend processing of some or all of the active operations from the time that it sends the Access-control request until it receives the Access-control response. If the client response is acceptable to the server, the suspended operations proceed as if the challenge had never taken place.
3. If the client fails to respond correctly to the challenge, the server might decide to terminate one or more operations but to leave open the Z-association. In that case, the server's terminating response to any such operations may indicate that the operation was terminated because of an Access control failure. Alternatively, the server may close the Z-association.

Parameters of the Access Control Service

Parameter Name	Server Request	Client Response	Reference
Security-challenge	m		3.2.5.1.1
Security-challenge-response		m	3.2.5.1.1
Other-information	o	o	3.2.5.1.2
Reference-id	ia	ia	3.4

3.2.5.1.1 Security-challenge and Security-challenge-response

Definitions for format and content of the challenge and response are subject to registration; several definitions are defined in Appendix ACC. Alternatively, the contents of these two parameters may be established by prior agreement between a given server/client pair.

3.2.5.1.2 Other-information

This parameter may be used by either the client or server for additional information not specified by the standard. This parameter may be used only when version 3 is in force.

3.2.5.1.3 Reference-id

If serial operations is in effect, or if concurrent operations is in effect and the challenge pertains to a particular operation, then the use of Reference-id is governed by section 3.4. If 'concurrent operations' is in effect and the challenge pertains to the Z-association, then the Reference-id is to be omitted from both the request and response.

3.2.6 Accounting/Resource Control Facility

The Accounting/Resource Control facility consists of three services:

- The Resource-control service, invoked by the server, either as part of an active operation (of any type) or pertaining to the Z-association
- The Trigger-resource-control service, invoked by the client as part of an active operation (of any type except Init)

- The Resource-report service, invoked by the client to initiate a Resource-report operation

The Resource-control service permits the server to send a Resource-control request, which might include a resource report. The report might notify the client that either actual or predicted resource consumption will exceed agreed upon limits (or limits built into the server), and request the client's consent to continue an operation, via the Resource-control response. The server might, for example, inform the client about the current status of a result set being generated on the server during a Search operation, and indicate information about the progress of the operation.

The Trigger-resource-control service permits the client to request that the server initiate the Resource-control service, or cancel the operation.

The Resource-report service permits the client to request that the server send a Resource-report pertaining to a completed operation or to the Z-association.

3.2.6.1 Resource-control Service

A client must be prepared to accept and respond to Resource-control requests from the server if resource control is in effect. A server may issue a Resource-control request which is either part of a specific (active) operation or which pertains to the Z-association.

- If concurrent operations is in effect:
  - If the Resource-control request includes a Reference-id: The supplied Reference-id must correspond to an active operation; the Resource-control request is part of that operation. The Resource-control response (if any) must also include that Reference-id.
  - If the Resource-control request does not include a Reference-id: The Resource-control request and response are not part of any operation, they pertain to the Z-association.
- If serial operations is in effect: The server may issue a Resource-control request only when there is an active operation; the Resource-control request and (possible) subsequent response are part of that operation and must include the Reference-id of the operation (which is assumed 'null' if not present in the initiating request).

The Resource-control request indicates whether a response is required:

- If so, the client must issue a Resource-control response. If the Resource-control request was part of an operation: the response is part of the same operation; the server awaits the Resource-control response, and subsequently issues a terminating response after processing of the operation is concluded.
- If not, the client must not issue a Resource-control response. If the Resource-control request was part of an operation: the server subsequently issues the terminating response, after processing of the operation is concluded.

A client should be prepared to receive, and (conditionally) respond to, multiple Resource-control requests as part of an operation (while the operation is active), or pertaining to the Z-association.

If the client responds to a Resource-control request with a Resource-control response saying to terminate an operation, it can expect to receive a terminating response. This response might

indicate that the operation was terminated at client request. However, the response might alternatively indicate that the operation completed, since the operation at the server may continue to execute and subsequently complete before the Resource-control response reaches the server.

Parameters of the Resource Control Service

Parameter Name	Server Request	Client Response	Reference
Resource-report	o		3.2.6.1.1
Partial-results-available	ia		3.2.6.1.2
Suspended-flag	ia		3.2.6.1.3
Response-required	m		3.2.6.1.4
Triggered-request-flag	o		3.2.6.1.5
Continue-flag		m	3.2.6.1.6
Result-set-wanted		ia	3.2.6.1.7
Other-information	o	o	3.2.5.1.8
Reference-id	ia	ia	3.4

3.2.6.1.1 Resource-report

This parameter may be used to convey information about the current and estimated resource consumption at the server. The format of Resource-report resource-1 and resource-2 are defined in Appendix RSC.

3.2.6.1.2 Partial-results-available

The server indicates the status of the result set via the flag Partial-results-available, whose value is one of the following:

Value of Partial-results-available	Description
subset	Partial, valid results available.
interim	Partial results available, not necessarily valid.
none	No results available.

This parameter is meaningful only as part of a search operation. If its value is 'subset' or 'interim,' then the server will accept subsequent Present requests against the result set if the client indicates (via the Continue-flag) that the operation is to be terminated and if the value of the parameter Result-set-wanted is 'on'.

If the value of Partial-results-available is 'none' then the server need not accept subsequent Present requests in the event that the client indicates (via the Continue-flag) that the operation is to be terminated.

Note that if the Suspended-flag is off, the partial results available situation may change because processing of the Search operation may continue. In all cases, the values of Search-status and

Result-set-status in the Search response should be treated as the authoritative information.

Access to Incomplete Results

This parameter Partial-results-available may be used not only for post-search information, but during a search as well. The server may notify the client of the availability of partial (i.e. incomplete) results by sending a resource report that may include the name of a result set (different from the result set specified in the Search Request) that contains the partial results, and the client may begin retrieving records, if concurrent operations is in effect. The client may send a trigger-resource-control request during the search, specifying the searchResult-1 format (see Appendix USR, USR.1), and so indicate that it wants partial results.

3.2.6.1.3 Suspended-flag

This parameter is valid only when the request pertains to an operation. The server indicates whether or not processing of the operation has been suspended pending the Resource-control response. This flag occurs if and only if the value of Response-required is 'yes'.

3.2.6.1.4 Response-required

The server indicates whether or not a response (from the client) to this request is required.

3.6.2.1.5 Triggered-request-flag

This parameter is valid only when the request pertains to an operation. The server may optionally indicate whether or not this request resulted from a Trigger-resource-control request from the client.

3.2.6.1.6 Continue-flag

This parameter is valid only when the request pertains to an operation. The client indicates to the server whether or not to continue processing the operation.

3.2.6.1.7 Result-set-wanted

This flag is valid only:

- During a Search operation,
- When the value of Partial-results-available is 'subset' or 'interim,' and
- When the value of the parameter Continue-flag is 'do not continue'.

If the value of this flag is 'yes,' the server will maintain the (possibly partial) result set for subsequent Present operations. If the value of the flag is 'no,' the server may delete the result set. A result set status of 'none' on the subsequent Search response indicates that the server has discarded the result set. In all cases, the values of Search-status and Result-set-status in the Search response describe the actual decisions made by the server and the way in which the search terminated.

3.2.6.1.8 Other-information

This parameter may be used by either the client or server for additional information, not specified by the standard. This parameter may be used only when version 3 is in force.

3.2.6.1.9 Reference-id

See 3.4.

3.2.6.2 Trigger-resource-control Service

A client may issue Trigger-resource-control requests during an operation (except during an Init operation), as part of that operation. It serves as a signal to the server that the client wishes the server to:

a) Simply send a Resource-report, i.e. issue a Resource-control request with Response-required 'off';
b) Invoke full resource control, i.e. issue a Resource-control request with Response-required 'on'; or
c) Cancel the operation.

The server is not obliged to take any specific action upon receipt of a Trigger-resource-control request. For the purpose of procedure description, there is no response to the request; if the server wishes to issue a Resource-control request it does so unilaterally. (If the client issues a Trigger-resource-control request and subsequently receives a Resource-control request as part of the same operation, the client cannot necessarily determine whether the latter resulted from the Trigger-resource-control request. However, the server may include Triggered-request-flag in the Resource-control-request to so indicate.)

If the client issues a Trigger-resource-control request saying to cancel the operation, and if the server honors the request, the client can expect to receive a terminating response indicating that the operation was terminated at client request.

Although a client may issue a Trigger-resource-control request as part of an active operation, the server might receive the request after the operation terminates. In that case, the server will ignore the Trigger-resource-control request. Furthermore, the server might receive a Trigger-resource-control request after issuing a Resource-control request for the same operation, while awaiting a Resource-control response. In that case, again, the server should ignore the Trigger-resource-control request. (Note that in general, the server may ignore any Trigger-resource-control request.)

Parameters of the Trigger-resource-control Service

Parameter Name	Client Request	Reference
Requested-action	m	3.2.6.2.1
Preferred-resource-report-format	ia	3.2.6.2.2
Result-set-wanted	ia	3.2.6.2.3
Other-information	o	3.2.3.2.4
Reference-id	ia	3.4

3.2.6.2.1 Requested-action

The client indicates one of the following:

resource-report	Issue a Resource-control request and set Response-required to 'off'
resource-control	Issue a Resource-control request and set Response-required to 'on'
cancel	Terminate the operation

3.2.6.2.2 Preferred-Resource-report-format

The client may indicate a resource report format that it prefers.

3.2.6.2.3 Result-set-wanted

This flag is meaningful only for a Search operation, and when the requested action is 'cancel'. If the value of the flag is 'yes,' the client requests that the server maintain the (possibly partial) result set for subsequent Present operations. See 3.2.6.1.7.

3.2.6.2.4 Other-information

This parameter may be used by the client for additional information, not specified by the standard. This parameter may be used only when version 3 is in force.

3.2.6.2.5 Reference-id

See 3.4.

3.2.6.3 Resource-report Service

The Resource-report service allows a client to request a Resource-report, pertaining to a specified, completed operation, or to the entire Z-association.

Note: The Resource-report service differs from the Trigger-resource-control service, in this respect: Trigger-resource-control is a non-confirmed service; there is a request, but no response. The request is part of, but does not initiate, an operation; it requests a report pertaining to that active operation. Resource-report, in contrast, is a ***confirmed*** service; there is a request and a response (the server is obliged to respond, although the server is not obliged to include a resource report in the response). The request and response initiate and terminate an operation respectively; the request identifies a particular ***completed*** operation and solicits a report pertaining to that operation (or it may solicit a report pertaining to the entire Z-association).

Parameters of the Resource Report Service

Parameter Name	Client Request	Server Response	Reference
Preferred-Resource-report-format	o		3.2.6.3.1
Op-id	o		3.2.6.3.2
Resource-report-status		m	3.2.6.3.3
Resource-report		o	3.2.6.3.4
Other-information	o	o	3.2.5.3.5
Reference-id	o	ia	3.4

3.2.6.3.1 Preferred-resource-report-format

The client may indicate a resource report format that it prefers.

3.2.6.3.2 Op-id

This parameter may be supplied by the client to identify a completed operation for which the client requests a resource report. This parameter may be used only when version 3 is in force.

- If Op-id is present, it consists of a Reference-id, and refers to the most recently completed operation that used that Reference-id.

Notes:

1. When an operation terminates, if the client anticipates that it will subsequently issue a Resource-report request pertaining to that operation, it is the client's responsibility to ensure that the Reference-id is not re-used before doing so.
2. The client may (but need not) use the same reference-id for the Resource-report operation as that specified in Op-id, and if so, Op-id will nevertheless pertain to a completed operation only. However, it is recommended that the client not specify a value of Op-id equal to any reference-id being used by any active operation other than this Resource-report operation. If the client does so, the server may (but need not) consider the request in error (see failure-6 of Resource-report-status).
3. If the client wants resource information about an ***active*** operation, it should not use the Resource-report service, but instead use the Trigger-resource-control service, as part of that operation. If the operation terminates before the server receives the Trigger-resource-control request, the client will receive a terminating response and may then subsequently issue a Resource-report request pertaining to that (completed) operation.

- If Op-id is not present, the client requests a resource report pertaining to the Z-association.

3.2.6.3.3 Resource-report-status

The server supplies one of following status values:

Status	Meaning
success	A resource report is included (and in the preferred format, if the parameter Preferred-resource-report-format was included in the request)
partial	A resource report is included, but not in the preferred format (applies only if the parameter Preferred-resource-report-format was included in the request)
failure-1	Server unable to supply resource report
failure-2	Operation terminated by server due to resource constraints
failure-3	Access-control failure
failure-4	Unspecified failure
failure-5	There is no known operation with specified id
failure-6	There is an active operation with specified id

Note: Failure-5 and failure-6 apply only when version 3 is in force.

3.2.6.3.4 Resource-report

See 3.2.6.1.1.

3.2.6.3.5 Other-information

This parameter may be used by either the client or server for additional information, not specified by the standard. This parameter may be used only when version 3 is in force.

3.2.6.3.6 Reference-id

See 3.4.

3.2.7 Sort Facility

The Sort facility consists of a two services, Sort and Duplicate-detection.

3.2.7.1 Sort Service

The Sort service allows a client to request that the server sort a result set (or merge multiple result sets and then sort). The client specifies a sequence of sort elements. The result set is to be ordered according to the specified sequence, and subsequent positional requests against the result set will be construed by the server to apply to the result set as so ordered.

Parameters of the Sort Service

Parameter Name	Client Request	Server Response	Reference
Input-result-sets	m		3.2.7.1.1
Sorted-result-set	m		3.2.7.1.2
Sort-sequence	m		3.2.7.1.3
Sort-status		m	3.2.7.1.4
Result-set-status		ia	3.2.7.1.5
Diagnostics		ia	3.2.7.1.6
Result-count		o	3.2.7.1.7
Other-information	o	o	3.2.7.1.8
Reference-id	o	ia	3.4

3.2.7.1.1 Input-result-sets

This parameter is the name of a result set to be sorted, or the names of result sets to be merged and the result sorted.

3.2.7.1.2 Sorted-result-set

This parameter is the name of the sorted result set. It may be the name of an existing result set (including one of the names included in Input-result-set); if so, then if the sort is processed, the existing result set is deleted, and a new result set by that name is created; its content is the sorted results. If Sorted-result-set is not the name of an existing result set and if the sort is processed, a result set by the specified name is created by the server, whose content is the sorted results; the content of the Input-result-sets is left unchanged. In any case, if the sort is not processed, the final content of Sorted-result-set is indicated by the parameter Result-set-status.

3.2.7.1.3 Sort-sequence

The parameter Sort-sequence comprises the elements that are to be used for sorting, together with the direction of the sort (ascending or descending), case sensitivity (if applicable), and server action if an element is missing from a record in the result set to be sorted.

Each sort element includes a sort key (that the server has designated either via the Explain facility, or through some mechanism outside of the standard) that takes one of three possible forms:

(1) field-in-record

(2) abstract access point

(3) private sort key

(1) and (2) correspond respectively to a "retrieval" and "search" view of the record.

The "field-in-record", corresponds to a retrieval element of the record, as defined perhaps by a schema, and specified by an element specification, such as eSpec-2, or by an element set name.

The element specification or name should resolve to a single element; if it resolves to several elements, the sort key is not well defined.

- "abstract access point" corresponds to a search attribute, or several; however if several are supplied, then similarly, they should resolve to a single abstract access point.
- "private sort key" is used when the client and server have a prior agreement about what the supplied key means. For example the client may supply the string 'author', where that string doesn't denote any particular field in the record as defined by the schema, and it isn't an attribute (no attribute set id) but the server knows (because of prior agreement) that that string, when supplied as a private sort key, and when applied to the records in question, refers to a specific field.

3.2.7.1.4 Sort-status

The parameter Sort-status, returned by the server, assumes one of the following values:

Sort Status	Meaning
success	The sort was performed successfully
partial-1	The sort was performed but the server encountered records with missing values in one or more sort elements
failure	The sort was not performed. The server supplies one or more diagnostics in the parameter Diagnostics

3.2.7.1.5 Result-set-status

The server supplies this parameter if and only if the value of Sort-status is 'failure'. It refers to the contents of Sorted-result-set, and its value is one of the following:

Result-set Status	Meaning
empty	The result set is empty
interim	Partial results available, not necessarily valid
unchanged	The content of the result set is unchanged (applies only if Sorted-result-set is one of the input result sets)
none	Result set not created (applies only if Sorted-result-set is not one of the input result sets)

3.2.7.1.6 Diagnostics

The server includes this parameter if the value of Sort-status is 'failure'. It includes one or more diagnostic records.

3.2.7.1.7 Result-Count

The server may use this parameter to report the size of the output result set. The server is never obligated to supply this parameter, there is no default value, and the client should not draw any conclusion from its omission.

3.2.7.1.8 Other-information

This parameter may be used by the client or server for additional information not specified by the standard.

3.2.7.1.8 Reference-id

See 3.4.

3.2.7.2 Duplicate Detection Service

The Duplicate Detection service allows the client to request that the server analyze one or more result sets in terms of potential duplicates and to construct a new result set according to client-specified criteria for detecting, retaining, grouping, and ordering the records including duplicates.

The following notation is used in the "Client Request" and "Server Response" column:

- [0,1] means parameter is optional, not repeatable; i.e. zero or one.
- 0+ means parameter is optional, repeatable; i.e. zero or more.
- 1 means parameter is mandatory, not repeatable; i.e exactly one.
- 1+ means parameter is mandatory, repeatable; i.e. one or more.

Parameters of the Duplicate Detection Service

Parameter Name	Client Request	Server Response	Condition	Reference
Input Result Set Id	1+			3.2.7.2.1
Output Result Set Name	1			3.2.7.2.1
Applicable Portion of Record	[0,1]			3.2.7.2.2
Duplicate-detection Criterion	0+			3.2.7.2.3
Clustering	[0,1]		May be omitted if representative record only is to be retained (if Retention Criterion is 'number of entries' and its value is 1). Otherwise must be supplied.	3.2.7.2.4
Retention Criterion	1+			3.2.7.2.5
Sort Criterion	0+			3.2.7.2.6

Parameter Name	Client Request	Server Response	Condition	Reference
Status		1		3.2.7.2.7
Result count		[0,1]	Must occur if Status is 'success'.	3.2.7.2.8
Diagnostic		0+	Must occur if Status is 'failure'.	3.2.7.2.9
Other-information	[0,1]	[0,1]		3.2.7.2.10
Reference-id	[0,1]	[0,1]	See 3.4	3.2.7.2.11

3.2.7.2.1 Input Result Set Id and Output Result Set Name

The client identifies one or more transient result sets belonging to the current Z-association. The server is to logically merge the sets (removing duplicates and ordering equivalence classes according to the parameters below) into a single result set, specified by the parameter Output Result Set Name.

3.2.7.2.2 Applicable Portion of Record

The client may specify what portion of the record is subject to matching (for example, one or more fields) for purposes of duplicate-detection. If this parameter is omitted, the server decides what portion of the record is subject to matching.

3.2.7.2.3 Duplicate-detection Criterion

For modeling purposes, a temporary, intermediate result set (not the output result set) is assumed to be created, which includes all of the result set items from all of the input result sets (including duplicate result set items). The server applies duplicate-detection criteria supplied in this parameter (or if the client omits this parameter, the server applies whatever duplicate detection criteria it chooses) to partition the intermediate result set into one or more ***equivalence classes*** where two result set items are considered ***equivalent*** if they are duplicate. That is, the partitioning has the following properties:

- Every result set item from one of the input result sets is in exactly one class
- Any two result set items are in the same class if and only of they are duplicates

The server distinguishes a single result-set item within each equivalence class as the ***representative record*** for that class. The selection of representative record might be based on the value of the parameter Sort Criterion.

The client may specify one or more criteria for detecting duplicates. These include the following (the list is subject to extension):

level of match	If this criterion is included, the client specifies a level of match in terms of a percentage. For example, fingerprints might be duplicates based on a 60% match; 100% might mean that records are duplicates only if they are identical.
Case sensitive	
Punctuation sensitive	
Regular expression	If this criterion is included the client supplies a regular expression to govern matching.
result-set duplicates	Two result set items are ***result-set duplicates*** if they point to the same database record.

3.2.7.2.4 Clustering

The client indicates one of the following:

Clusters	The output result set is to contain one item for each equivalence class. For each equivalence class, create a result set item for the representative record only and maintain duplicates as metadata. (Records may subsequently be presented either as (a) representative record with duplicates attached as metadata, using, for example, GRS; or (b) as a cluster record, using an appropriate cluster syntax.)
Individual Entries	Create individual result set items for representative records as well as duplicates that are to be retained (according to Retention Criterion). Order the output result set such that records within an equivalence are grouped together. The parameter Sort Criterion may be supplied, to indicate how the records within a class are to be ordered.

This parameter may be omitted only if 'Number of entries' is supplied as a retention criterion (parameter Retention Criterion) and the value supplied is 1.

3.2.7.2.5 Retention Criterion

The client specifies one or more criteria for how records are to be selected for inclusion in or exclusion from each equivalence class. These include the following (the list is subject to extension):

1. Number of entries	If this criterion is selected, the client supplies a number, N>0, meaning retain (up to) N entries in each equivalence class. N=1 means retain the representative record only. This value may be used in combination with

	(3) and/or (4), but not (2).
2. Percent of entries	If this criterion is selected, the client supplies a percentage, xx, meaning retain xx percent of the entries in each equivalence class. xx=100 means retain all entries. This value may be used in combination with (3) and/or (4), but not (1).
3. Duplicates only	Discard representative record. This value should not be specified unless the value of parameter Clustering is 'Individual Entries'. This value may be used in combination with (1) or (2), and/or (4).
4. Discard result-set duplicates	This value may be used in combination with (1) or (2), and/or (3). If used with (1) or (2) the result-set duplicates should be discarded first (before entries are selected).

3.2.7.2.6 Sort Criterion

The client may provide one or more sort criteria for selecting the representative record as well as for ordering records within an equivalence class.

This parameter will affect the ordering of result set items only within an equivalence class (it does not affect the ordering of equivalence classes). If the value of parameter Clustering is 'Clusters' then this parameter will have no effect whatever on the result set order (though it may be supplied anyway, to govern the selection of representative records as well as the order in which duplicates are presented within a single cluster record).

More than a single sort criterion may be supplied; if so, the order in which they are supplied is from major to minor, and only the first criterion supplied is used to govern selection of a representative record. The sort criteria include the following (the list is subject to extension):

Most Comprehensive	Select the longest (more comprehensive) record as the representative record; order duplicates within an equivalence class by descending comprehensiveness.
Least Comprehensive	Select the shortest (least comprehensive) record as the representative record; order duplicates within an equivalence class by ascending comprehensiveness.
Most Recent	Select the most recent record as the representative record; order duplicates within an equivalence class by ascending age.
Oldest	Select the oldest record as the representative record; order duplicates within an equivalence class by descending age.
Least Cost	Select the least expensive record as the representative record; order duplicates within an equivalence class by ascending cost.
Preferred Database	Select a record from the most preferred database as the

representative record; order duplicates within an equivalence class corresponding to order of preference of databases. When this criterion is supplied the client includes a list of databases in order of preference.

3.2.7.2.7 Status

The server indicates a status of 'success' or 'failure'.

3.2.7.2.8 Result Count

If the value of parameter Status is 'success' then the value of this parameter is the size of the output result set.

3.2.7.2.9 Diagnostic

The server may always include one or more diagnostics in the response. If the value of parameter Status is 'failure', at least one diagnostic must be included.

3.2.7.2.10 Other-Information

This parameter may be used by the client or server for additional information, not specified by the standard.

3.2.7.2.11 Reference-id

See section 3.4.

3.2.8 Browse Facility

The Browse facility consists of a single service, Scan.

3.2.8.1 Scan Service

The Scan service is used to scan an ordered term list (subject terms, names, titles, etc.). The ordering of the term list is server defined. The client specifies a term list to scan and a starting term (implicitly, by specifying an attribute/term combination and a database-id), the size of the scanning steps, and the desired number of entries and position of the starting term in the response.

Note: The Z39.50 term list abstraction is intended as a generalization of the concept of an index corresponding to a search access point. A term list may but need not necessarily be an index, or correspond to a search access point.

Parameters of the Scan Service

Parameter Name	Client Request	Server Response	Reference
Database Names	m		3.2.8.1.1
Term-list-and-start-point	m		3.2.8.1.2
Step-size	o	ia	3.2.81.3
Number-of-entries	m	m	3.2.8.1.4
Position-in-response	o	o	3.2.8.1.5
Scan-status		m	3.2.8.1.6
Entries		o	3.2.8.1.7
Other-information	o	o	3.2.8.1.8
Reference-id	o	ia	3.4

3.2.8.1.1 Database-names

The parameter, Database-names, identifies a set of databases to which the term list (specified by Term-list-and-start-point) pertains.

3.2.8.1.2 Term-list-and-start-point

The client supplies an attribute list and term. The attribute list contains attributes indicating which term list to scan. The term, as qualified by those attributes, indicates where scanning begins; this will be a presumed entry in the term list. If there is no matching entry, the first entry with higher value is to be the starting point.

As an example, to scan a list of personal names: the attribute list might consist of a single attribute whose type is 'use' and whose value is 'personal name'; the term would specify a personal name; the database-id would identify one or more databases to which the list of personal names pertains.

3.2.8.1.3 Step-size

The client may specify the desired number of entries in the term list between two adjacent entries in the response. A value of zero means "do not skip any entries." If the server cannot support the requested step size, it sets Scan-status to 'failure' and includes a non-surrogate diagnostic such as "only step size of zero supported" or "requested step size not supported." If the client omits this parameter, the step size is selected by the server, and the server includes the selected step size in the response.

3.2.8.1.4 Number-of-entries

The client indicates the proposed number of entries to be returned. The server indicates the actual number of entries returned. If the actual number is less than the proposed number, the reason is indicated in Scan-status.

3.2.8.1.5 Position-in-response

The client may optionally indicate the preferred position, within the returned entries, of the specified starting point value. A value of 1 refers to the first of the returned entries. A value of 0 means that the returned entries should begin with the term immediately following the starting point term. A value of Number-of-entries + 1 means that the client requests terms immediately preceding the starting point term.

The server may indicate the actual position of the chosen starting point within the returned entries. Example: If the values of the request parameters Number-of-entries and Position-in-response are 10 and 3 respectively, then the client requests two terms immediately preceding the starting point value, followed by the starting point value, followed by the immediately-following seven terms.

Note: If response parameter Position-in-response is less than the value proposed in the request, the client may conclude that there were fewer terms than expected in the low end of the term list. However, if Position-in-response is the same value in the response as proposed in the request, but Number-of-entries in the response is less than the value proposed in the request, the client may not conclude that there were fewer terms than expected at the high end of the term list, unless Scan-status is Partial-5. The reason that fewer terms than expected are returned is indicated in the Scan-status.

Example Illustrating the Semantics of the Position-in-response Parameter

Consider a term list that includes consecutive terms A, B, C, D, E, and F; the Scan Request specifies C as the starting term, and 2 for Number-of-entries. If the value of Position-in-response is 1, terms C and D are to be supplied. If the value of Position-in-response is 2, terms B and C are to be supplied. If the value of Position-in- response is 3 then terms A and B are to be supplied. If the value of Position-in-response is zero, Terms D and E are to be supplied. If the value of Position-in-response is minus 1, then terms E and F are to be supplied; etc.

3.2.8.1.6 Scan-status

The server indicates the result of the operation. The defined values are:

Value of Scan-status	Meaning
success	The response contains the number of entries (term-list-entries or surrogate diagnostics) requested.
partial-1	Not all of the expected entries can be returned because the operation was terminated by access-control.
partial-2	Not all of the expected entries will fit in the response message.
partial-3	Not all of the expected entries can be returned because the operation was terminated by resource-control, at client request.
partial-4	Not all of the expected entries can be returned because the operation was terminated by resource-control, by server.
partial-5	Not all of the expected entries can be returned because the term list contains fewer entries (from either the low end, high end, or both ends of the term list) than the number of terms requested.

Value of Scan-status	Meaning
failure	None of the expected entries can be returned. One or more non-surrogate diagnostics is returned.

3.2.8.1.7 Entries

The parameter Entries returned by the server:

- Consists of one of the following:
  - N entries, where each entry is a term-list-entry or surrogate diagnostic, where N = Number-of-entries in the request
  - A number of entries that is less than N, and may be zero (reason specified by Scan-status)
- And may also include:
  - One or more non-surrogate diagnostic records (possibly indicating that the operation cannot be processed, and why it cannot)

Each term-list-entry includes a term (occurring in one of the databases specified in the parameter Database-names), and optionally the following:

- A display term (when the actual term is not considered by the server to be suitable for display)
- A list of suggested attributes for use in subsequent Scan requests (useful for scanning multiple indices, e.g. author and title, at the same time)
- A suggested alternative term
- Occurrence-information: this might include a count of records in which the term occurs. It may also list counts for specific attributes, possibly further broken down by database. Alternatively, a term-list-entry might list databases in which the term occurs, and for associated attributes, but no counts.
  Note: A “Count” is a number of records. There is no mechanism provided by this standard to indicate the count of occurrences of the term.
- Other information: additional information concerning the entry

3.2.8.1.8 Other-information

This parameter may be used by the client or server for additional information, not specified by the standard.

3.2.8.1.9 Reference-id

See 3.4.

3.2.9 Extended Services Facility

The Extended Services facility consists of a single service, Extended-services.

3.2.9.1 Extended Services Service

The Extended-Services (ES) service allows a client to create, modify, or delete a task package at the server. The server maintains task packages in a special database, described in section 3.2.9.2. A task package pertains to an ES task.

An extended service is a task type, related to information retrieval, but not defined as a Z39.50 service. Execution of a task by the server is outside the scope of Z39.50. The extended services defined by this standard are listed in section 3.2.9.1.2. Definitions of those services are included in Appendix EXT.

The client sends an ES Request to the server requesting execution of a task. The request includes parameters that the server uses to construct the task package. The server checks the request for validity, for consistency with the user's access privileges, and possibly for other server-dependent limitations. The server sends an ES response indicating that the request was accepted or supplying an indication of the reason the request was rejected.

The ES service is a confirmed service, initiated by the client. The ES operation consists of a request from the client and a response from the server, possibly with intervening Access-control or Resource-control messages. However, although the request may result in the initiation of a task, the task is not considered part of the Z39.50 ES operation. The server response, which completes the ES operation, does not necessarily signal completion of the task. A task may have a lifetime that exceeds a single Z-association. Execution of the ES Operation results in the creation of a task package, represented by a database record in the ES database.

For example, when a server creates a task package of type PersistentResultSet, a (persistent) result set is created, represented by the created task package, in the form of a record in the extended services database. When that package is subsequently retrieved by a client, in either the same or a different Z-association, a copy of that persistent result set is made available to that Z-association, as a Z39.50 result set (i.e. as a transient result set; a result set name, for use during the Z-association, is included within the task package). When a client deletes the task package, the persistent result set is deleted.

A task package contains parameters, some of which are common to all task packages regardless of package type, and others that are specific to the particular extended service. Among the common parameters (indicated in the table below, listed under "task package parameter" in the right column), some are supplied by the client as parameters in the ES request, and are used by the server to form the task package; some of those supplied by the client may be overridden by the server. Others are supplied by the server. The specific parameters are derived from the parameter Task-specific-parameters of the ES request (see Appendix EXT).

Note: The response parameter Task-package below refers to the actual task package, and if it occurs (see 3.2.9.1.13), it includes some or all (depending on the parameter Elements) of the parameters listed under "task package parameter."

Parameters of the Extended Services Service

Parameter Name	Client Request	Server Response	Task-Package Parameter	Reference
Function	m			3.2.9.1.1
Package-type	m			3.2.9.1.2
Package-name	o		o	3.2.9.1.3
User-id	o		o	3.2.9.1.4
Retention-time	o		o	3.2.9.1.5
Permissions	o		o	3.2.9.1.6
Description	o		o	3.2.9.1.7
Server-reference			o	3.2.9.1.8
Creation-date-time			o	3.2.9.1.9
Task-status			o	3.2.9.1.10
Package-diagnostics			o	3.2.9.1.11
Task-specific-parameters	m		(Task-specific)	3.2.9.1.12
Wait-action	m			3.2.9.1.13
Elements	ia			3.2.9.1.14
Operation-status		m		3.2.9.1.15
Operation-diagnostics		ia		3.2.9.1.16
Task-package		ia	o	3.2.9.1.17
Other-information	o	o		3.2.7.1.18
Reference-id	o	ia		3.4

3.2.9.1.1 Function

The client specifies Create, Delete, or Modify. If the function is Create, the server is to create a task package, and assign to it the name specified by the parameter Package-name, if supplied.

If the function is Delete or Modify, the server is to delete or modify the task package specified by the parameter Package-name. A server that supports deletion or modification may nonetheless deny the request, for example because the task is already in progress, or the package is in use.

If the function is Delete, the client requests that if the specified task has not been acted on, it should not be started. If the task is active, the server should either terminate the task or refuse the request.

If the function is Modify, the client requests that parameter values in the request (as well as those within parameter Task-specific-parameters) replace the corresponding values in the task package. If an optional parameter is omitted, the server does not modify that parameter within the task package (thus to return a parameter to its default value, a client must explicitly provide the default value).

3.2.9.1.2 Package-type

The Package-type identifies the extended service requested. The extended services defined by this standard (see Appendix EXT) are:

- Save a result set for later use
- Save a Query for later use
- Define a periodic search schedule
- Order an item
- Update a database
- Create an export specification
- Invoke a previously created export specification

3.2.9.1.3 Package-name

The client may optionally supply a name for the task package to be created. If so, the triple (Package-type, User-id, Package-name) must be unique (i.e. there must be no other task package of that type, for that user with the same name, otherwise the request is in error), and that triple identifies the task package for subsequent reference. Package-name should be included if the client intends to reference the task package.

3.2.9.1.4 User-id

The User-id identifies the user to be associated with the task package. If not supplied, this parameter may default to the Id of the current user. A server may or may not allow a client to supply a user id different from its own.

3.2.9.1.5 Retention-time

The client may optionally specify a retention period (e.g. 2 hours, 3 days, 1 week), which may be overridden by the server. When the retention time has passed, the server may delete the retained task package. A retention time of zero means the task package is not to be retained after the task is completed.

3.2.9.1.6 Permissions

The client may indicate who may access the task package. If the client does not supply this parameter, only the creating user may do so. See 3.2.9.3.

3.2.9.1.7 Description

The client may include a description. It might describe, for example, the result set, for a Persistent Result Set task; or the query, for a Persistent Query task.

3.2.9.1.8 Server-reference

The server may supply a unique identifier for the task package.

3.2.9.1.9 Creation-date-time

The server supplies the date and time that the task package was created.

3.2.9.1.10 Task-status

The server indicates the status of the task. Values are 'pending,' 'active,' 'complete,' and 'aborted'. See also 3.2.9.5.

3.2.9.1.11 Package-diagnostics

The server may include one or more diagnostics in the task package.

3.2.9.1.12 Task-specific-parameters

These are additional parameters, defined by the specific extended service.

3.2.9.1.13 Wait-action

The client indicates whether the server should (or may) include the task package in the ES response. This immediate response mechanism may avoid the need for follow-up Search and Present operations, or in general, for making the task package available through the extended services database (see section 3.2.9.2).

This parameter has four possible values:

Value of Wait-action	Meaning
wait	The server must perform the task before issuing the ES response (unless the operation aborts; see section 3.2.9.4). If the tar-get is not willing to perform the task before issuing the response it must refuse the request by responding with a status of 'failure' and an appropriate diagnostic. If the server accepts the request, it includes the parameter Task-package in the response.
wait-if-possible	The client requests that, if possible, the server perform the task before issuing the ES response and include the task package in the response. If not possible, the server should proceed as though the value were 'do not wait'.

Value of Wait-action	Meaning
do-not-wait	The client does not request that the server attempt to perform the task before issuing the ES response. However, if the server does perform the task before issuing the response, then the response may include the task package.
Do-not-send-task-package	The server may perform the task when it chooses, but is not to include the task package in the response under any circumstance.

3.2.9.1.14 Elements

The client may optionally include this parameter if Wait-action is other than 'do-not-sent-task-package'. It is an element set name for the task package, in the event that it is returned in the response parameter Task-package.

3.2.9.1.15 Operation-status

This is the status of the ES operation. It is one of the following:

Value of Operation-Status	Meaning
done	The request was accepted, the task is complete and results are included in Task-package.
accepted	The request was accepted and the task is queued for processing, or is in process
failure	The request was refused. One or more diagnostics are supplied (in parameter Operation-diagnostics).

See also 3.2.9.5.

3.2.9.1.16 Operation-diagnostics

The server may supply additional diagnostic information if Operation-status is 'failure'.

3.2.9.1.17 Task-package

If Operation-status is 'done,' the server includes the task package. The portion of the actual task package included depends on the parameter Elements.

3.2.9.1.18 Other-Information

This parameter may be used by the client or server for additional information, not specified by the standard.

3.2.9.1.19 Reference-id

See section 3.4.

3.2.9.2 The Extended Services Database

Servers that support the Extended Services facility provide access to a database with the name IR-Extend-1 (referred to as the "extended services database" or "ES database").

Note: Thus if a server claims to support Extended Services, it supports the ES database to the extent, at minimum, that if a client searches IR-Extend-1 the server will not fail the search because there is no such database. In the case where the server does not create task packages (see next note) it may always respond to searches that zero records were identified.

Records in the extended services database are task packages constructed from the Request-parameter-package parameter in ES requests (the server may begin execution of the task at any time after it accepts the request, which may be before the task package has been stored in the database). The server may (but need not) retain a task package until the requested task has completed; it may retain the task package until the client requests that it be deleted. A server may unilaterally delete a task package from the ES Database at any time.

Note: This means, as a practical matter, the server need not actually create a task package for a given task, in particular, when the task is executed immediately. However, it is recommended that a task package exist when the status of the task is pending, active, or aborted.

When the server receives an ES request it may immediately create a task package, with status 'pending,' before completely validating the request. The client may thus search the database anytime after submitting a request (during the same or a subsequent Z-association), for a resulting task package. In particular, if an ES operation is aborted (see 3.2.9.4) the client may be able to determine that the request for that operation was received.

An ES database may be listed in the server Explain database, with a list of extended services the server supports, allowable export destinations, options that a client may supply for an export task, etc.

An extended services database will appear to the client as any other database supported by the server (records may be searched and retrieved by the Z39.50 Search and Retrieval facilities; search processing is defined locally by the server; the server may impose access control or exclude records to which the client is not authorized access). However, certain search terms are predefined in order to allow a semantic level of interoperability. The attribute set used to search the database is defined in Appendix ATR. The task package structures are defined in Appendix EXT.

The ES database may provide the following special element sets (in addition to "F"):

Element Set Name	Meaning
Identification	The creating user's identification, the client-supplied name of the task package, and possible permissions for other users to access the request. Other identifying information such as time of creation may be included.
UniqueName	The creating user's identification and the name of the task package.

Element Set Name	Meaning
Permissions	The contents of the UniqueName element set, and in addition, the granted permissions for the task package. A server might present the full permissions list only to the task package creator, presenting to other users only the permissions applicable to them.
Status	A short summary of the current status of the request, perhaps including cost and other resource usage.
Brief	Identification element set plus the most important elements of the Status element set.

3.2.9.3 Owners and Permissions

The creating user of a task package may apply any extended service function to the package, as well as retrieve the full package (via the Retrieval facility) and invoke the package via other extended services. (Invocation occurs, for example, when a Periodic Query task references a saved Query.)

Using the Modify function of the ES request, a client can change the access permissions of a task package by supplying a new permissions list, which is a sequence of user ids and for each, a sequence of allowed operations, from the following set:

- Delete
- Modify-Contents
- Modify-Permissions
- Present
- Invoke

As an example of the use of the 'invoke' permission, a server might create a task package, on behalf of a client user, of type PersistentQuery; a persistent query is created, represented by the created task package. The server may subsequently be requested to create a PeriodicQuerySchedule task package, on behalf of a different user, which refers to (i.e. "invokes") that persistent query task package. The server would do so only if that user has 'invoke' privilege for that persistent query. As another example, a server may create an ExportSpecification (package) on behalf of one user, and a different user may subsequently 'invoke' that ExportSpecification by creating an InvokeExportSpecification package, if that user has 'invoke' privilege for the ExportSpecification.

Servers may provide group names for use in permission lists, but a group name would be syntactically the same as a user Id. (The server might report the composition of groups, but the mechanism for doing so is not described by this standard.)

3.2.9.4. Aborted Operations

A client may receive a response to an ES request only during the Z-association in which it issues the request (as for any other Z39.50 operation). If an ES operation is aborted (explicitly, or because the Z-association is closed or the connection is lost), the client will not receive a terminating response. This has no effect on the disposition or processing of the task, regardless

of the value of Wait-action that was specified on the request. If an ES operation aborts, Wait-action automatically assumes the value 'do-not-send-task-package'.

If an ES operation is aborted, the client may search the ES database (possibly in a subsequent Z-association) for information that would otherwise have been returned in the response.

3.2.9.5 Description of Status Parameters

Task-Status and Operation-Status (3.2.9.1.10 and 3.2.9.1.17) both apply to Extended Services in general; individual Extended Services may define specific additional statuses (for example the Update Extended Service defines Update-status and Record-status) Operation-status and Task-status distinguish the ES operation from the ES task. The "ES operation" is the ES request followed by the ES response, the result of which is the spawning of an ES task, which may be monitored by Task-status. Thus Operation-status is set only once, after the operation is complete, in contrast to Task-status, a dynamic status that changes as the status of the operation changes. Neither of these two parameters conveys any status information specific to the ES type.

Task-status

Task-status is a general ES parameter, and it exists only in the task package (i.e. it is not an explicit parameter of the ES response). Values are:

- 'pending',
- 'active',
- 'complete', and
- 'aborted'

Its purpose is to allow the client, by repetitively retrieving the package, to monitor the progress of its execution, from initiation of the ES operation until completion of the task. But it is not intended to provide status specific to the type of task, and in particular, it is not intended to convey whether the task was completed successfully, only that it completed.

In the abstract, when the server receives the ES request, if it passes preliminary inspection and the task is queued, Task-status is ‘pending’; if it does not pass preliminary inspection then at the server discretion there may not even be a task package created (see Operation status) but if there is, its status is set to 'aborted'. Once the task starts (which may or may not be before the server sends back the ES response) the status is set to 'active'. If it subsequently aborts, it is set to 'aborted' and if it subsequently completes, it is set to 'complete’. Note that the ‘pending’ state need not occur; the server might start the task immediately upon receipt of the task package.

Operation status

Operation-status, in contrast to Task-status, is always included in the ES response. (Task-status exists only in the task package, and the task package might not be included in the ES response.) Its values are:

- 'done',
- 'accepted', and
- 'failure'.

The operation status of 'done' means the task package is included in the ES response (in that case this status is redundant); 'accepted' corresponds to Task-status of 'pending' or 'active', and 'failure' corresponds to the case where the task package was not even set up because the task did not pass preliminary inspection.

3.2.10 Explain Facility

The Explain facility allows a client to obtain details of the implementation of a server, including databases available for searching, attribute sets and diagnostic sets used by the server, and schema, record syntax and element specification definitions supported for retrieval. Servers that support the Explain facility:

- Provide access (via the Z39.50 Search and Present services) to a database with the name IR-Explain-1 (referred to as the "Explain database");
- Support the explain attribute set, exp-1, defined in Appendix ATR (which defines a set of Use attributes and imports bib-1 non-Use attributes); and
- Support the Explain syntax, which is defined in Appendix REC.
- A record (or result set item representing a record) within the Explain database is referred to as an "Explain record".

3.2.10.1 Searching the Explain Database

The Explain database appears to the client as any other database supported by the server. However, certain search terms, corresponding to information categories, are predefined in order to allow a semantic level of interoperability. Terms are searched case-insensitive.

The exp-1 attribute set is used to search the Explain database. Combinations of Use attributes and terms allow searching upon information category; well-defined combinations of Use attributes may be used to allow additional specification by the client to limit the records to those of immediate interest. Combinations of exp-1 Use attributes to perform a common set of searches are listed in 3.2.10.1.1 and 3.2.10.1.4. Since the Explain database may be searched as any other database using attributes from one or more attribute sets, this list is not exhaustive. However, it is recommended that a server supporting the Explain facility support this list of common searches. As described in 3.2.10.1.2 and 3.2.10.1.3, the HumanStringLanguage, DateAdded, DateChanged, and DateExpires attributes can be used in combination with any of the combinations listed in 3.2.10.1.1 and 3.2.10.1.4.

The exp-1 attribute set consists of a set of Use attributes and imports the non-Use bib-1 attributes. It is recommended that a server supporting the Explain facility support the bib-1 relation attribute 'equal' (see note), position attribute 'any position in field', and structure attribute 'key'.

Note: If the server intends to support searching based on date ranges (e.g. to limit a search to records created before or after a particular date or between two dates), the server should also support one or more of the following relation attributes: 'less than', 'less than or equal', 'greater than', and 'greater or equal'.

Clients should not in general expect that the explain database is searchable using the bib-1 truncation attribute, completeness attribute nor any of the alternative values of the relation,

position and structure attributes defined in bib-1. However, servers are free to provide access to the Explain database using those and other alternative attributes and attribute values.

3.2.10.1.1 Searching for Predefined Information Categories

Records corresponding to a particular explain information category are searched by an operand where the term is the name of that category; for example, all records corresponding to TargetInfo are searched using the term "TargetInfo." For each category one or more ***key elements*** are defined, and may be provided as search terms (using the appropriate attribute). A search with an operand where the Use attribute = 'ExplainCategory' and the term is a category, and with additional operands corresponding to each key for that category where the value of the Use attribute is the key, should result in (at most) a single record.

The primary mechanism for search and retrieval of information from the Explain database is for the client to select the records in a category using the Use attribute 'ExplainCategory' and to extract desired information from those records to formulate a subsequent search. For example the client may search records with ExplainCategory = 'DatabaseInfo,' and retrieve summary information (see 3.2.10.2.2) from those records. Each summary record will include a database name, which serves as a key for a possible subsequent search.

A list and brief description of the Explain information categories (and thus search terms) are given in the table below, as well as the keys for each category. In 3.2.10.3 each category is described in detail.

A client should adhere to the following rules when searching an Explain database by the predefined information categories.

- To search for information about the server, use ExplainCategory='TargetInfo'.
- To search for information about a specific database, use ExplainCategory='DatabaseInfo' in combination with the DatabaseName attribute to specify the key of the desired databaseInfo record.
- To search for information about a specific schema, use ExplainCategory='SchemaInfo' in combination with the SchemaOID attribute to specify the desired schema.
- To search for information about a specific tag set, use ExplainCategory='TagSetInfo' in combination with the TagSetOID attribute to specify the desired tag set.
- To search for information about a specific record syntax, use ExplainCategory='RecordSyntaxInfo' in combination with the RecordSyntaxOID attribute to specify the desired record syntax.
- To search for information about a specific attribute set, use ExplainCategory='AttributeSetInfo' in combination with the AttributeSetOID attribute to specify the desired attribute set.
- To search for information about term lists for a database, use ExplainCategory='TermList-Info' in combination with the DatabaseName attribute to specify the desired database.
- To search for information about a specific extended service, use ExplainCategory = 'ExtendedServicesInfo' in combination with the oid for that extended service.
- To search for the attributes and combination of attributes which may be used in searching a database, use ExplainCategory='AttributeDetails' in combination with the DatabaseName attribute to specify the database for which attribute information is desired.

- To search for information about a specific term list, use ExplainCategory='TermListDetails' in combination with the name for the term list.
- To search for the element set names defined for a record syntax for a particular database, use ExplainCategory='ElementSetDetails' in combination with the RecordSyntaxOID attribute to specify the desired record syntax and the DatabaseName attribute to specify the desired database.
- To search for the definition of a specific element set name, use ExplainCategory = 'ElementSetDetails' in combination with the ElementSetName attribute to specify the desired element set name. There may be multiple records located since the explain database contains one record for each element set name for each record syntax for each database.
- To search for a particular element set name defined for a record syntax, for a particular database, use ExplainCategory='ElementSetDetails' in combination with the ElementSetName attribute to specify the desired element set name, the RecordSyntaxOID attribute to specify the desired record syntax and the DatabaseName attribute to specify the desired database.
- To search for the description of the elements of a retrieval record, for a particular record syntax, in a specific schema, for a particular database, use ExplainCategory='RetrievalRecordDetails' in combination with the RecordSyntaxOID attribute to specify the desired record syntax, the SchemaOID attribute to specify the desired schema, and the DatabaseName attribute to specify the desired database.

Category	**An Explain record in this category describes:**	**Key(s)**
TargetInfo	The server, including search constraints imposed by the server.	server name
DatabaseInfo	A database. A group of databases offering a common set of characteristics may be described as a single, logical, database. In this case, a list of databases subsumed within this logical database is provided.	database name
SchemaInfo	A Schema.	schema oid
TagSetInfo	A tag Set.	tagSet oid
RecordSyntaxInfo	A record syntax.	record syntax oid
AttributeSetInfo	An attribute set, including the attributes supported within the set.	attribute set oid
TermListInfo	Term lists supported for a database.	database name
ExtendedServicesInfo	An extended service.	extended service oid
AttributeDetails	Attributes that can be used to search a database including the other attributes with which it may be combined.	database name
TermListDetails	A term list.	term list name
ElementSetDetails	An element set (for a particular record syntax, for a particular database).	database name, element set name, record syntax oid

Category	An Explain record in this category describes:	Key(s)
RetrievalRecordDetails	The elements of a retrieval record (for a particular record syntax, defined by a particular schema).	databaseName, schema oid, recordSyntax oid
SortDetails	Sort specification for a database.	database name
Processing	Processing instructions for a database, for a particular processing context, name of instructions, and object identifier for the abstract syntax of the externally defined Instructions.	database name, processing-context, name,oid
VariantSetInfo	A variant set definition; classes, types, and values, for a specific variant set definition supported by the server. Support by the server of a particular variant set definition does not imply that the definition is supported for any specific database or element.	variantSet oid
UnitInfo	Unit definitions supported by the server.	unit system name
CategoryList	Explain categories that the server supports.	(no key)

3.2.10.1.2 Searching for Information in a Particular Language

Elements intended to be presented to the user by the client are said to consist of "human readable text." Each record includes a language element indicating the language of the human readable text within the record. The explain database might contain several records with identical information, in different languages. To search for records in a certain language, the HumanStringLanguage attribute may be used (in conjunction with the three-character language code as the term; see Z39.53-1994).

For example, to search for a list of databases that have descriptive records in English, the query might be of the form:

(Category = 'DatabaseInfo') AND (HumanStringLanguage = 'eng')

The HumanStringLanguage attribute is intended primarily for use in Version 2. When version 3 is in force, the use of variants is recommended.

3.2.10.1.3 Searching for Information by Control Dates

To search for new records in an Explain database, use the DateAdded attribute; for updated records use the DateChanged attribute, for records based on their date of expiry use the DateExpires attribute. Any of these three may be used in combination with the searches described above.

3.2.10.1.4 Searching for Information Using Content Values

Some of the Explain records are searchable using attributes, which take values from elements within the pertinent Explain records. These Use attributes can be used to select subsets of records of specific information category. For instance, the Availability Use attribute can be used to select those database information records for databases that are currently available. The use of these attributes by a client should conform to the following rules.

- To locate databases currently available, use the ExplainCategory attribute with term 'DatabaseInfo,' in combination with the Availability attribute with term 'yes'.
- To locate the databases provided by a specific supplier, use the ExplainCategory attribute with term 'DatabaseInfo,' in combination with the Supplier attribute with the supplier's name as term.
- To locate databases provided by a specific producer, use the ExplainCategory attribute with term 'DatabaseInfo' in combination with the Producer attribute with the producer's name as term.
- To locate databases that are not proprietary, use the ExplainCategory attribute with term 'DatabaseInfo,' in combination with the Proprietary attribute with term 'no'.
- To locate databases that have no user fee, use the ExplainCategory attribute with term 'DatabaseInfo,' in combination with the UserFee attribute with term 'no'.

3.2.10.2 Retrieval of Explain Records

A Present request for Explain records should specify the Explain syntax as the Preferred-record-syntax. Each explain information category has its own record layout, and all are described in the Explain syntax definition (see Appendix REC, REC.1).

Explain records include key elements that serve to uniquely identify each record. Each Explain category is defined in term of key elements, non-key "brief" elements (see 3.2.10.2.2), "non-brief" elements, and possibly other categories. Key elements are always part of the brief elements.

3.2.10.2.1 Retrieval and Human Readable Text

The Explain database might provide alternative variations of human readable information (however, for language variations; see note below). For example, a text element might be retrievable in ASCII, HTML, or PDF. To request a particular format, use the variant facilities of Version 3.

Note: For language variation, see 3.2.10.1.2. The Explain database logically includes different records for different languages, and therefore selection based on language occurs during the search.

3.2.10.2.2 Retrieving Summary and Descriptive Information

The Explain facility provides for the retrieval of summary, or "brief" information. For example the client may request summary information about all of the databases supported by a server without retrieving the full databaseInfo records. Within each category's definition, elements are designated as "brief" or "non-brief." Elements designated "brief" are obtained when using the element set name 'B'. Elements designated "non-brief" are obtained (along with brief-elements) when using the element set name 'F'.

The Explain facility also provides for the retrieval of descriptive information, for certain categories, via the element set name 'description' (for details, refer to the ASN.1 definition for the Explain syntax). For example, a Database-info record includes an element that contains a description (in human readable text) of the database; to retrieve only the brief elements and the description element, the element set name 'description' may be used.

Individual categories defined in the Explain syntax may designate other element set names for specific subsets of information within that category.

3.2.10.3 Detailed Descriptions of the Information Categories

This section includes complete descriptions of each information category. In addition to the information enumerated, each record:

- Contains information about the record itself, e.g. date of creation and expiration date of the record
- Includes an element indicating the language of the "human readable text" elements of the record

These are logical descriptions, which do not reflect the possibility that there might be language variants of a record or syntax variants of an element.

Many of the Explain elements are optional, but are not so indicated in the description below. For specific information, refer to the ASN.1 definition.

3.2.10.3.1 TargetInfo

TargetInfo is information about the server. There is one such Explain record in the Explain database.

Brief elements:

- A name for the server (only one), in human readable text
- Recent news of interest to people using this server, in human readable text
- An icon used to represent this server (in machine presentable form)
- Whether named results sets are supported (result set names other that "default")
- Whether multiple databases can be searched in one search request
- The maximum number of concurrent result sets supported, for a given Z-association. Thus for example, if the value is 2, say a client creates (via a search) result set A, then B; a subsequent attempt to create C will exceed the maximum (note however that the server action when the maximum is exceeded is not specified; for instance, the server might unilaterally delete A in order to create C, or it might return an error), however, if the client first deletes B, then it could create result set C without exceeding the maximum.
- The maximum size (in records) of a result set
- The maximum number of terms allowed in one search request. This is subject to server interpretation. It could mean maximum number of operands in the search, or operands of the form "attributesPlusTerm". Or, suppose for example result set A is the result of the search "cat" AND "dog"; and result set B is the result of the search (Result set A) AND "moon". The latter search has only one "Term", it has two "Operand"s, and all together the search involves three terms.

- A timeout interval after which the server will trigger an event if no activity has occurred.
- A "welcome" message from the server to be displayed by the client.
- Contact information for the organization supporting this server
- A description of the server, in human readable text
- A set of nicknames or alternate names by which the server is known
- Restrictions pertaining to this server, in human readable text
- A payment address (e.g. business office) for the organization supporting this server
- Hours of operation
- A list of supported database combinations
- Internet address and Port number
- Languages supported for message strings
- The following elements, where each object listed is supported for one or more databases. (To determine which are supported for a particular database, retrieve the record for that database.)
    - Which query-types are supported, and details for each supported type
    - Diagnostic sets supported
    - Attribute sets supported
    - Schemas supported
    - Record syntaxes supported
    - Resource challenges supported
    - Access challenges supported
    - Cost information
    - Variant sets supported
    - Element set names supported
    - Unit systems supported

3.2.10.3.2 Database-Info

Database-Info is a detailed description of a database and database-related restrictions and parameters. There is one such Explain record for each database supported.

Brief elements:

- Full database name (only one)
- Whether this is an Explain database (possibly for a different server)
- A list of short (or alternate) names for the database
- An icon used to represent this database (in machine presentable form)
- Whether there is charge to access this database
- Whether this database is currently available for access
- A human-readable name or title for the database (as opposed to the database name, which is typically a short string not meant to be human-readable, and not variable by language.)

Non-brief elements:

- A list of keywords for the database
- A description of the database, in human readable text
- Associated databases: those that the server allows (and possibly encourages) to be searched in combination with this database
- Sub-databases that make up this conceptual single database
- Any disclaimers concerning this database, in human readable text
- News about this database, in human readable text
- A record count for the database (and whether the count is accurate or an estimate)
- A description of the default order in which records are presented, in human readable text
- An estimate of the average record size (in bytes)
- A maximum record size (in bytes)
- Hours of operation that this database is available
- Best time to access this database, in human readable text
- Time of last update of this database
- Update cycle/interval for this database
- Coverage dates of this database, in human readable text
- Whether this database contains proprietary information
- A description of copyright issues relating to this database, in human readable text
- A notice concerning copyright that the server expects the client to display to the user if possible, in human readable text
- Description and contact information for the database producer, database supplier, and for how to submit material for inclusion in this database, in human readable text
- Which query-types are supported for this database, and details for each supported type
- Diagnostic sets supported for this database
- Attribute sets supported for this database
- Schemas defined for this database
- Record syntaxes supported by this database
- Resource reports supported for this database
- Text describing access control for this database, in human readable text
- Costing information related to this database, in both machine-readable format, and in human readable text, for connect, present, and search
- Variant sets supported for this database
- Element set names supported for this database, with names and descriptions given in human readable text
- Unit systems supported for this database

3.2.10.3.3 Schema-Info

Schema-Info is descriptive information about a database schema. There is one Explain record for each schema supported by the server.

Note: This is not specific to a database.

Brief elements:

- The object identifier of the schema definition
- The name of this schema

Non-brief elements:

- A description of this schema, in human readable text
- TagSets used by this schema, and for each, a designated tagType
- The abstract record structure defined by this schema

3.2.10.3.4 Tag-Set-Info Descriptive information about a given tagSet

TagSetInfo is included as an Explain category to allow a client to retrieve (and in turn allows the end-user to discover) information about a tagSet supported on the server. The server can convey, for a supported tagSet, the identifier of the tagSet (an Object Identifier), and for each supported element, its name, tag, datatype, and description.

This Explain category might support collaboration between the end-user and client, where the client retrieves the TagSetInfo information and conveys names and descriptions to the end-user, who then decides which elements are of interest and requests the client to retrieve those specific elements. Thus the semantics of an individual element are conveyed transparently to the end-user, the client never needs to understand their meaning, only their datatype and tag; and the end-user never knows the tag or datatype.

There is one such Explain record for each supported tagSet.

Brief elements:

- The object identifier for the tagSet
- The name of this tagSet

Non-brief elements:

- A description of this tagSet, in human readable text
- For each element defined in the tagSet
  - The name of the element
  - Nicknames for the element
  - The tag assigned to the element
  - A description of the element
  - Its datatype

3.2.10.3.5 Record-Syntax-Info

Record-Syntax-Info is descriptive information about a record syntax. There is one Explain record for each abstract record syntax supported by the server.

Note: This is not specific to a database.

Brief elements:

- The object identifier of the abstract record syntax
- A name by which this syntax is known

Non-brief elements:

- Transfer syntaxes supported for this abstract syntax (object identifiers)
- A description of this abstract record syntax, in human readable text
- An ASN.1 module describing the syntax

The record structure defined by this syntax.

3.2.10.3.6 Attribute-Set-Info

Attribute-Set-Info is descriptive information about an attribute set. There is one record for each supported attribute set.

Brief elements:

- The attribute set Id (object identifier) for this attribute set
- Its name

Non-brief elements:

- For each attribute type, its name, description, and integer value of the type, and a list of attributes.
  - Its name
  - Description
  - Its value
  - Names of equivalent attributes. Equivalences are derived from the attribute set definition (not from the servers behavior)
- Description of the attribute set

3.2.10.3.7 TermList-Info

TermList-Info is descriptive information about term-lists. There is one Explain record for each database.

Brief elements:

- Full database name (one only).
- Summary information about each term-list associated with this database (for each term-list described, there is a TermList-details record):
  - Name of the term-list. Must be unique for the database. This is the name to be used to search for the TermList-details record for this term list.
  - Its title. For users to see; need not be unique

- An indication of how expensive it is to search, using the associated attributes. The server indicates one of the following:
    - The attribute (combination) associated with this list will do fast searches. **Note:** To obtain the attribute combination, retrieve the associated TermList-details record.
    - The attribute (combination) will work as expected. So there is probably an index for the attribute (combination) or some similar mechanism.
    - Can use the attribute (combination), but it might not provide satisfactory results. Probably there is no index, or post- processing of records is required.
    - Cannot search with this attribute (combination) alone
- Whether the term-list may be scanned
- A list of names of alternative, broader term-lists
- A list of names of alternative, narrower term-lists

(No non-brief elements.)

3.2.10.3.8 Extended-Services-Info

Extended-Services-Info is descriptive information about an extended service. There is one Explain record for each extended service supported.

Brief elements:

- The object identifier of the extended service
- A name by which this extended service is known
- Boolean flags, indicating:
    - Whether it is a private extended service
    - Whether restrictions apply
    - Whether a fee applies
    - Whether the service is available
    - Whether retention is supported
- What level of wait-action is supported

Non-brief elements:

- A description, in human readable text
- Explain elements specific to this extended service (defined within the specific extended service definition)
- An ASN.1 module for the Explain definition

3.2.10.3.9 Attribute-Details

Attribute-Details contain information for each attribute. There is one Explain record for each supported database.

Brief elements:

- Name of the database to which this attribute information applies

Non-brief elements:

- For each attribute set supported for the database, the object identifier of the attribute set, and for each attribute within the set:
  - The attribute type
  - A default value which applies if the attribute is omitted, and a description of default behavior in human readable form
  - For each value of the attribute:
    - The attribute value
    - A description of that value in human readable text
    - Sub-attributes (for Use attributes): a list of alternative values that allow access to the same aspect of the record, but in greater detail
    - Super-attributes (for Use attributes): a list of alternative values that allow access to the same aspect of the record at a coarser level
    - Whether the value is only "partially supported": i.e. the value is accepted but may not provide expected results
- A list of all attributes combinations supported for the database

3.2.10.3.10 Term-list-Details

Term-list-Details is descriptive information for a term-list. There is one record for each term-list listed by TermList-info records.

Brief elements:

- Name of the term-list

Non-brief elements:

- A description
- Attribute combination corresponding to this list. If list may be scanned, this is the attribute combination to be used by scan.
- Maximum step-size supported
- Collating sequence (e.g. ASCII) in human-readable text
- Order (ascending or descending)
- Estimated number of terms
- A list of sample terms (not guaranteed to be valid; optimally would represent a uniformly distributed sampling of the list)

3.2.10.3.11 Element-Set-Details

Element-Set-Details is descriptive information about an element set. There is one Explain record for each element set for each record syntax for each database.

Brief elements:

- The database to which this record pertains
- The element set name for the element set described by this record
- The record syntax to which this record pertains
- The schema for which this element set is defined

Non-brief elements:

- A description, in human readable text, of the element set
- For each element in the element set, the information provided for each element by the Retrieval-Record-Details category

3.2.10.3.12 Retrieval-Record-Details

Retrieval-Record-Details is descriptive information about the elements of a retrieval record. Note that the elements are relative to a database schema. There is one such Explain record for each database for each schema for each record syntax.

Brief elements:

- The database, schema, and record syntax to which this Explain record pertains

Non-brief elements (for each element described by the syntax):

- The name of the element
- The tag of the element, if any
- A list of schema elements that comprise this element within the record syntax
- The maximum size of the element
- The minimum size of the element
- The average size of the element
- The size of the element, if fixed length
- Whether or not the element is repeatable
- Whether or not the element is required
- A description of the element, in human readable text
- A description of its contents, in human readable text
- Charging/billing issues related to this element, in human readable text
- Restrictions (e.g. copyright, proprietary) pertaining to use and access to this element, in human readable text
- Alternate names for this element
- Generic names for this element text (e.g. a "geographicSubject" element might also be under the generic name "subject")
- Attribute combinations corresponding to this element

3.2.10.3.13 Sort-Details

Sort-Details is a description of the sorting capabilities supported by the server. There is one record for each database.

Brief elements:

- Database to which this sort description pertains

Non-brief elements:

- For each sort key:
  - A description
  - If the key is a record element, a specification of the element
  - If the key is an attribute combination, a specification of that combination
  - The type of key: character, numeric, structured
  - Whether the key is case-sensitive

3.2.10.3.14 Processing-Info

Instructions, representing how the server believes the data should be processed by the client for presentation to the user. Instructions are defined externally. For a given database and processing context (access, search, retrieval, record-presentation, and record-handling) for which the server offers processing information, there may be more than one set of instructions; these are distinguished by name. Each set of instructions may be available in more than one abstract syntax; these are distinguished by object identifier. Thus an Explain record of this type is distinguished by database, processing context, name, and object identifier.

Brief elements:

- Full name of the database to which this record pertains
- The context for which this processing information is pertinent
- A name for this processing information
- An object identifier, for the abstract syntax of the externally defined instructions

Non-brief elements:

- A description of the instructions, in human readable form
- The machine processable instructions, externally defined (whose abstract syntax is identified by the object identifier referenced above)

3.2.10.3.15 Variant-set-info

Variant-set-info is descriptive information about a variant set definition supported by the server; classes, types, and values supported for a particular variant set. Support of a particular variant set definition does not imply that the definition is supported for any specific database or element.

Brief elements:

- The object identifier of the variant set definition
- Its name

Non-brief elements:

- A list of supported classes, including name and description; and for each, a list of supported types, including name and description; and for each, a list of supported values.

3.2.10.3.16 Unit-info

Unit-info is descriptive information about a unit system definition supported by the server.

Brief elements:

- The name of the unit system

Non-brief elements:

- A description
- A list of unit types, including name and description, and for each, a list of units, including name and description

3.2.10.3.17 Category-list

Category-list is a list of the Explain categories supported by the server. There is one such record for the Explain database. It consists of the information below, for each supported category.

Brief elements:

- The search term used in conjunction with Use attribute of ExplainCategory to search for records of this category

Note: The following need occur only if the server is supporting a category not defined in this standard.

- The original search term. (This is for information categories where the server is supporting a revision of the original definition of a category.)
- A description
- An ASN.1 definition of the record for this category

3.2.11 Termination Facility

The Termination Facility consists of the single service, Close.

3.2.11.1 Close Service

The Close service allows either a client or server to abruptly terminate all active operations and to initiate termination of the Z-association.

The Close service may be used only when version 3 is in force. If so, following initialization, at any time until a Close request is either issued or received, either the client or server:

- May issue a Close request, consider all active operations to be abruptly terminated, await a Close response (discarding any intervening messages), and consider the Z-association closed; and
- Should be prepared to receive a Close request, consider all active operations to be abruptly terminated, issue a Close response, and consider the Z-association closed.

Parameters of the Close Service

Parameter Name	Request	Response	Note	Reference
Close-reason	m	m		3.2.11.1.1
Diagnostic-Information	o	o	server only	3.2.11.1.2
Resource-report-format	o		client only	3.2.11.1.3
Resource-report	o	o	server only	3.2.11.1.3
Other-information	o	o		3.2.11.1.4
Reference-id	ia	ia		3.4

3.2.11.1.1 Close-reason

This parameter indicates the reason why the client or server is closing the Z-association. Its values are:

- finished
- shutdown
- system problem
- cost limits
- resources
- security violation
- protocol error
- lack of activity
- unspecified
- response to Close request

Note: Both the Close request and Close response map to the same protocol message (Close APDU). If both systems issue a Close request at the same time, each will receive the peer message as a Close response (even though the message was not sent as such). This potential ambiguity will not effect the correct operation of the protocol. However, for the case where the message is indeed sent as a Close response, the last of the above listed statuses, "response to Close request" is provided and may optionally be used.

3.2.11.1.2 Diagnostic-information

The server may include an optional text message, providing additional diagnostic information.

3.2.11.1.3 Resource-report-format and Resource-report

When the client issues a Close request: the client may include the parameter resource-report-format to request that the server include a resource report (see 3.2.6.1.1) in the response. The server's decision to include a resource report in the response (and the format) is unilateral: it may

include or omit a report regardless of whether the client included the parameter resource-report-format.

When the server issues a Close request: the server may unilaterally include a resource report.

3.2.11.1.4 Other-information

This parameter may be used by the client or server for additional information, not specified by the standard.

3.2.11.1.5 Reference-id

The parameter Reference-id may be included or omitted on a Close request or response from the client.

The server should omit Reference-id on a Close request. On a Close response, if the server is responding to a Close request that included Reference-id, the server may either include Reference-id using the identical value, or it may omit the parameter. If the server is responding to a Close request that did not include a Reference-id, the server should omit the parameter.

3.3 Message/Record Size and Segmentation

A "segment" is a message that is sent (or is in preparation for transmission) by the server as part of an aggregate Present response, i.e. a Segment request or Present response.

Throughout Section 3.3, the term "record" is used as follows:

- Unless otherwise qualified, it indicates a "response record," i.e., retrieval record or surrogate diagnostic.
- Except within Section 3.3.3, it refers to a "surrogate diagnostic record" if the record size exceeds preferred-message-size.
- "Record N" means "the response record corresponding to the database record identified by result set entry N."
- A record is considered to be a string of bytes (for the purpose of describing segmentation procedures).
- "Record size" refers to the size of a record, in bytes.

Except within Section 3.3.3, a set of records is said to "fit into a segment" if the sum of their sizes, not including protocol control information, does not exceed Preferred-message-size. For the Present operation, the server might be unable to fit the requested records in a single segment, because of record or message size limitations. In that case, the server may perform segmentation of the Present response (if segmentation is in effect) by sending multiple segments (Segment requests followed by Present response).

Two levels of segmentation, level 1 and level 2, are subject to negotiation. If neither level is in effect, the server response to a Present request consists of a simple Present response (a single segment), which contains an integral number of records. If level 1 segmentation is in effect, the server response to a Present request may consist of multiple segments (Segment requests followed by a Present response), and each segment must contain an integral number of records,

i.e. records may not span segments. If level 2 segmentation is in effect, the server response to a Present request may consist of multiple segments, and records may span segments.

3.3.1 Procedures When No Segmentation is in Effect

The procedures in this section (3.3.1) apply when no segmentation is in effect. (They apply not only to a Present operation when no segmentation is in effect, but they also apply in general to a Search operation, whether or not segmentation is in effect; a Search response is not subject to segmentation.)

The server responds to a Present request with a simple Present response (or to a Search request with a Search response), which contains an integral number of records. If the server is not able to return all of the records requested, because of message size limitations, the server should fit as many records as possible.

Assume that the server is attempting to return records M through N. If records M through N fit in the response, then the server returns those records. Otherwise, the server returns records M through P, where P is chosen such that records M through P fit in the response, but records M through P+1 do not.

Illustration

Assume that the server is attempting to return records 1 through 10; records 1 through 6 fit in the response, but retrieval records 1 through 7 will not fit.

The size of retrieval record 7, itself:

(a) Does not exceed Preferred-message-size, or

(b) Exceeds Preferred-message-size, but does not exceed Exceptional-record-size, or

(c) Exceeds Exceptional-record-size.

In case (a), the server returns records 1 through 6. In case (b), except as noted below (see "Exception"), the server substitutes a diagnostic record for retrieval record 7, indicating that the record exceeds Preferred-message-size. In case (c) the server substitutes a diagnostic record for retrieval record 7, indicating that the record exceeds Exceptional-record-size. If Exceptional-record-size equals Preferred-message-size then there is no distinction between the meaning of the two diagnostics.)

In case (b) or (c):

- If the diagnostic record will not fit along with records 1 through 6, the server returns records 1 through 6. (Preferred-message-size must always be large enough to contain any diagnostic record; thus a subsequent present request beginning with record 7 will retrieve the diagnostic.)
- Otherwise, the server inserts the diagnostic record and proceeds to attempt to fit records 8 through 10.

Exception

If a Present request specifies a single record (i.e. Number-of-records-requested equals 1) then if the size of that retrieval record exceeds Preferred-message-size, but does not exceed

Exceptional-record-size, the server will return that single retrieval record. Note that this exception applies only to a Present operation and not to a Search operation.

Thus in case (b), the client may subsequently retrieve retrieval record 7, by issuing a Present request in which that record is the only record requested.

Note that the purpose of this distinction between Preferred-message-size and Exceptional-record-size is to allow the transfer of normal length records to proceed in a routine fashion with convenient buffer sizes, while also providing for the transfer of an occasional exceptionally large retrieval record without requiring the client to continually allocate and hold local buffer space for worst-case records. Note also that this intended purpose is defeated if the client routinely requests a single record.

3.3.2 Level 1 Segmentation

When level 1 segmentation is in effect, the server may segment the aggregate Present response into multiple segments (zero or more Segment requests followed by a Present response), each consisting of integral records (i.e. records may not span segments). The procedures described in this section (3.3.2) apply if level 1 segmentation is in effect.

Beginning with the first record requested and continuing with adjacent higher number records, the server forms segments to contain the requested records. Each segment is sent as a Segment request, except the last, which is sent as a Present response.

The number of segments must not exceed the value of the (optional) Present request parameter Max-segment-count, if supplied.

If Max-segment-count is supplied, and its value is 1, then the procedures of 3.3.1 apply. Also, the same exception as cited in 3.3.1 applies if a Present request has requested a single record.

Assume that the client requests result set records M through N.

Case A: M<N (i.e. more than one record requested).

1. Set P=M
2. If records P through N fit in a segment:
   - Fit records P through N in the segment
   - Go to step 3

   Otherwise,
   - Fit records P through Q, where Q (which is less than N) is such that records P through Q fit in a segment, but records P through Q+1 do not
   - If Max-segment-count is reached, go to step 3
   - Send the segment as a Segment request
   - Set P=Q+1
   - Repeat step 2
3. Send the segment as a Present response.

Case B: M=N (i.e. a single record requested).

The server sends a simple Present response (a single segment). The size of the segment may exceed Preferred-message-size. The segment contains the single requested retrieval record, or a surrogate diagnostic record if the size of the record exceeds Exceptional-record-size.

Illustration

Assume the client has requested records 1 through 10.

1. If all ten records fit in a segment, the aggregate Present response consists of a Present response including the requested records. Present-status is 'success' (all expected response records available).
2. Suppose records 1 through 4 fit in a segment, but records 1 through 5 do not; records 5 through 9 fit in a segment but records 5 through 10 do not. (Assume the Present request has specified a value of 3 or greater for the parameter Max-segment-count.) Then the aggregate Present response consists of:

   - A segment request including records 1 through 4,
   - A segment request including records 5 through 9, and
   - A Present response including record 10.

   Present-status is 'success' (all expected response records available).

   Note that the server is expected to pack as many records into a segment as will fit; thus for example, the first segment would not consist of records 1 through 3, because records 1 through 4 will fit.
3. Assume the conditions in (2) are true, except that the Present request has specified a value of 2 for the parameter Max-segment-count. Then the aggregate Present response consists of:

   - A Segment request including records 1 through 4, and
   - A Present response including records 5 through 9.

   Present-status is 'partial-2' (not all expected response records available, because they will not all fit within the preferred message size).

3.3.3 Level 2 Segmentation

When level 2 segmentation is in effect, the server may segment the aggregate Present response into multiple segments (as is the case for level 1 segmentation) and in addition, records may span segments. The procedures described in this section (3.3.3) apply if level 2 segmentation is in effect.

If a retrieval record will not fit in a segment (along with records already packed into the segment) it may be segmented into multiple contiguous fragments (see 3.3.3.1) to be packed into consecutive segments according to the procedures detailed in 3.3.3.2 and 3.3.3.3.

3.3.3.1 Fragments

A fragment is a proper sub string of a record (as noted above, within section 3.3.3 a record is treated as a string of bytes). A particular instance of segmentation of a record results in a sequence of two or more fragments whose concatenation (not including protocol control information) is identical to the record. However, there may be different instances of segmentation of a particular record, and the client cannot necessarily predict how a record will be segmented into fragments by the server in a particular instance.

For the purpose of procedure description (3.3.3.3) a starting fragment is defined to be a fragment that starts at the beginning of a record. An intermediate fragment is a fragment that neither starts at the beginning nor ends at the end of a record. A final fragment is a fragment that ends at the end of a record. An integral record (not segmented) is not a fragment.

The sum of the sizes of the records and record fragments in a segment, not including protocol control information, must not exceed Max-segment-size (see 3.3.3.2).

3.3.3.2 Segment Size, Record Size, and Segment Count

If level 2 segmentation is in effect, the Present request may optionally include these three parameters:

Max-segment-size	The largest allowable segment. If included, overrides Preferred-message-size (for this Present operation only). If not included, Max-segment-size assumes the value Preferred-message-size.
Max-record-size	The largest allowable retrieval record within the aggregate Present response. If included, it must equal or exceed Max-segment-size. (If level 2 segmentation is in effect, the parameter Exceptional-record-size that was negotiated during initialization does not apply, whether or not Max-record-size is included, unless the value of Max-segment-count is 1.)
Max-segment-count	The maximum number of segments the server may include in the aggregate Present response. If its value is 1, no segmentation is applied for the operation, the procedures of section 3.3.1 apply, and Max-record-size should not be included.

If the latter two parameters are both included, Max-record-size must not exceed the product Max-segment-size times Max-segment-count.

If Max-record-size but not Max-segment-count is included, the client should be prepared to receive as many segments as necessary to retrieve the requested records.

If Max-segment-count is included (and its value is greater than 1), but Max-record-size is not, the product Max-segment-size times Max-segment-count is the maximum record size for the operation.

If the latter two parameters are both omitted the client should be prepared to receive arbitrarily large records and an arbitrary number of segments.

3.3.3.3 Segmentation Procedures

The following procedures apply for level 2 segmentation. The server fits as many integral records as possible into the first segment. If all of the requested records will fit, the segment is sent as a simple Present response. Otherwise, in the space remaining within that segment the server fits a starting fragment of the following record (if possible), and the segment is sent as a Segment request. The server then fits the remainder of that record into the next segment (if possible; and if not possible, sends Segment requests as necessary with intermediate fragments, and fits the final fragment, if any, into the beginning of the next segment) and fills as many integral records as possible within the space remaining within that segment. If the last of the requested records is placed in the segment (or Max-segment-count is reached) the segment is sent as a Present response. Otherwise the server continues to fill segments in this manner until the last of the requested records is placed in a segment or Max-segment-count is reached, and sends each segment as a Segment request except the last, which it sends as a Present response. These procedures are stated more formally as follows:

Assume that the client requests records M through N. (Note that "Record" means "surrogate diagnostic record" if the size of the record exceeds Max-record-size, or if the server is unable to segment the record so that each fragment fits within a segment.)

1. Set R=M (begin preparation of first segment)
2. If record R fits in the current segment:
   - Fit integral records R through P, where P is the largest number (not exceeding N) so that records R through P fit
   - If P equals N, or if Max-segment-count is reached, go to step 8
   - R=P+1
3. **Note:** Having reached this step, record R will not fit in the current segment.

   If the Present request has included Max-segment-count and the server is unable to determine whether record R will fit in the remainder of the aggregate response:
   - Insert a surrogate diagnostic record, which in effect suggests that the client might again attempt to retrieve the record, but without specifying a Max-segment-count
   - Go to step 7
4. If record R will not fit in the remainder of the aggregate response, go to step 8
5. If record R will fit in the remainder of the aggregate response, but no starting fragment will fit in the current segment:

   Note: This condition precludes the possibility that the segment is empty; see note preceding step 1.
   - Transmit a Segment request (begin preparation of the next segment)
   - Go to step 2
6. **Note:** Having reached this step, Record R will fit in the remaining segments; it will not fit within the current segment, but a starting fragment will fit in the current segment.
   - Fit the largest possible starting fragment of record R and transmit a Segment request
   - Fill as many complete segments as necessary (which may be zero) with intermediate fragments of record R and send Segment requests

- Begin preparation of the next segment, first inserting the final fragment of record R

7. Set R = R+1
   - If R is less than or equal to N, go to step 2
8. Send a Present response.

Illustration

Assume the client has requested records 1 through 12. All records are 500 bytes, except record 5, which is 10,000 bytes. Max-segment-size is 3200.

1. Suppose record 5 consists of 10 elements, each 1000 bytes. The server is able to segment record 5, but only at element boundaries; the server will not let the elements span fragments.
   Note: this means that the server may segment the record so that a fragment consists of bytes M*1000+1 through (M+N)*1000, M= 0,1, ...9; N = 1, 2, ..., 10-M; e.g. bytes 1-1000, 1-2000, 1-3000, 1000-2000, 1000-3000, 1000-4000, etc.
   Suppose further that the server cannot segment any other records. The aggregate Present response is as follows:

segment 1	Segment request consisting of records 1 through 4, and the first 1000 bytes of record 5 as a starting fragment. **Note:** the size of the segment is 3000 bytes which is less than the Max-segment-size of 3200; but the server cannot fit another fragment in the segment because that would cause the segment size to exceed the 3200-byte maximum (the minimum fragment size is 1000 bytes).
segment 2	Segment request consisting of bytes 1001 through 4000 of record 5 as an intermediate fragment
segment 3	Segment request consisting of bytes 4001 through 7000 of record 5 as an intermediate fragment
segment 4	Segment request consisting of bytes 7001 through 10,000 of record 5 as a final fragment
segment 5	Segment request consisting of records 6 through 11
segment 6	Present response consisting of record 12

2. Suppose further that the server can segment the smaller records into 100 byte fragments (or multiples).

Segments 1 through 3 are as in case 1

segment 4	Segment request consisting of bytes 7001 through 10,000 of record 5 as a final fragment, and bytes 1 through 200 of record 6 as a starting fragment
segment 5	Segment request consisting of bytes 201 through 500 of record 6 as a final fragment, records 7 through 11, and the first 400 bytes of record 12 as a starting fragment
segment 6	Present response consisting of bytes 401 through 500 of record 12 as a final fragment

3. Suppose the server can segment any of the records at arbitrary byte boundaries.

segment 1	Segment request consisting of records 1 through 4 and the first 1200 bytes of record 5 as a starting fragment
segment 2	Segment request consisting of bytes 1201 through 4200 of record 5 as an intermediate fragment
segment 3	Segment request consisting of bytes 4201 through 7400 of record 5 as an intermediate fragment
segment 4	Segment request consisting of bytes 7401 through 10,000 of record 5 as a final fragment, record 6, and the first 100 bytes of record 7 as a starting fragment
segment 5	Present response consisting of bytes 101 through 500 of record 7 as a final fragment, and records 8 through 12

3.3.3.4 Segmentation and Access-or Resource-Control

A segmentation sequence may be interrupted by Access-control or Resource-control. A server might send one or more segments and follow (prior to sending the Present Response) with either an Access-Control request (for example because the next set of segments requires authorization) or a Resource-Control request (for example indicating that the next set of segments will cost more).

3.4 Operations and Reference-id

A request from the client of a particular operation type initiates an operation, which is terminated by the respective response from the server. The following operation types are defined: Init, Search, Present, Delete, Resource-report, Sort, Scan, Extended-services, and Duplicate-detection. (Thus each client request type corresponds to an operation type with the exception of the following request types: Trigger-resource-control and Close.) An operation consists of the initiating request and the terminating response, along with any intervening Access-control and Resource-control requests and responses, Trigger-resource-control requests, and Segment requests. An operation is assigned a Reference-id by the client, the client includes the Reference-id within the initiating request, and the same Reference-id must be included within each message of the operation. Fox example, if the client sends a Search Request and includes a Reference-id with value “123,” then the Search Response must also include that same reference id, “123.” If 'serial operations' is in effect, the Reference-id parameter may be omitted in the initiating request; in that case the reference-id is considered null for that operation, and all other messages of that operation must also omit the Reference-id parameter.

Any message sent from client to server or vice versa (i.e. any request or response defined by this service definition) is part of an operation (identified by its Reference-id), with the following exceptions:

- A Close request or response is not part of any operation.
  Note: A Close request or response may include a reference-id, according to the procedures specified in 3.2.11.1.5.
- If 'concurrent operations' is in effect any Resource-control or Access-control request or response which does not include a Reference-id is not part of an operation.

This standard does not assume any relationship between a given operation and any subsequent operation even if the latter operation uses the same reference-id. This standard does not specify the contents of the Reference-id parameter, nor its meaning, except to the extent that it is used to refer to an operation. Reference-ids are always assigned by the client and have meaning only within the client system. Since no semantics are attributed to the Reference-id, it has no implied data type and can only be described as transparent binary data. (Its ASN.1 type is therefore OCTET STRING.)

Even though the client and server should supply reference-ids as described above, in the case where an incorrect reference-id is supplied (and so long as 'serial operations' is in effect), the receiver of the incorrect reference-id may simply ignore it, or may declare a protocol error (i.e. issue a Close with close reason 'protocol error'), and this standard takes no position on which choice is better. An example of an incorrect reference-id would be when the client sends a Search Request with a reference-id and the server responds with a reference-id of a different value, or no reference-id at all. Another example: the client sends a Search Request, the server sends an Access-control Request (with the correct reference-id) and the client sends an Access-control Response with a different reference-id.

3.5 Concurrent Operations

If 'concurrent operations' is in effect, the Reference-id parameter is mandatory in an initiating request (however, see note), and the client may initiate multiple concurrent operations, each identified by a different reference-id.

Note: The Reference-id parameter is always optional in an Init request; 'concurrent operations' does not take effect until negotiation is complete, and is thus not in effect during an Init operation.

Once an operation is initiated, until that operation is terminated, another operation may not be initiated with the same reference-id. This standard does not specify the order in which concurrent operations are processed at the server; the server may process concurrent operations in any manner it chooses.

Example:

The client may issue a Search request using Reference-id "100," and then issue a second Search request using Reference-id "101" before receiving the Search response from the first Search request. There would then be two concurrent operations. Receipt by the client of the response corresponding to the second Search request (identified by Reference-id "101") would terminate the second operation, and that might occur before termination of the first operation (identified by Reference-id "100"). The client might then issue a Present request (against the result set created by the second operation), initiating another operation. In that case, the client must supply a Reference-id other than "100" (because there is an active operation with that Reference-id). The new Reference-id could (but need not) be "101"; if it is, the server may not assume any implied relationship between this new operation and the previous operation which used Reference-id "101."

No operation may be initiated while an Init operation is in progress. No operation may be initiated within a Z-association after a Close request has been sent or received.

All result sets are, in principle, available to any operation. It is possible that two or more concurrent operations will attempt to reference the same result set. This standard does not specify what happens in that circumstance. The client should not initiate concurrent Search operations with the same value of Result-set-id.

Other than the restriction cited above (that when the client uses a Reference-id to initiate an operation, until that operation is terminated it may not use that Reference-id to initiate another operation) there are no restrictions on the re-use or management of Reference-ids by the client. The client might re-cycle Reference-ids randomly among users, or it may manage local threads by assigning different Reference-ids to end-users. The server is not required to know how the client manages Reference-ids, or in particular, that the client is using Reference-ids to distinguish different users. There is no requirement for the server to have any knowledge of multiple end users at the client, the server interacts only with the (single) client.

3.6 Composition Specification

For each database supported the server defines one or more schemas (see 3.1.5), and designates one as the default schema. For each schema, the server designates one or more element specification identifiers.

An element specification identifier is the object identifier of an element specification format (a structure used to express an element specification) or an element set name. The latter is a primitive name. An element specification is an instance of an element specification format, or an element set name.

For the default schema, at least one of the element specification identifiers must be an element set name, and the server designates one as the default element set name for the database.

Note: The server designates this information either via the Explain facility, or through some mechanism outside of the standard.

For each record to be returned in a Search or aggregate Present response, the server applies an abstract record structure (defined by a schema for the database to which that record belongs) to form an abstract database record, to which the server applies an element specification to form another instance of the abstract database record (the latter might be a null transformation), to which the server applies a record syntax, to form a retrieval record.

If the client includes the parameter Comp-spec (in a Present request) the procedures of 3.6.1 apply. For a Search operation, or a Present operation when the parameter Comp-spec is omitted, the default schema is assumed for each record, and the procedures of 3.6.2 apply.

3.6.1 Comp-spec Specified

The Present request parameter Comp-spec includes a set of one or more pairs of a database name and associated composition specification. Each composition specification may include a schema identifier (or if not, the default schema for the database is assumed) and an element specification. For each record to be returned in the aggregate Present response:

- If the database to which the record belongs is specified (as a component of one of the pairs) then the server forms an abstract database record by applying the corresponding composition specification (i.e. by first applying the abstract record structure, defined by the

schema, to the database record to form an abstract database record, and then applying the element specification; where the schema and element specification are from the composition specification), if it is able to do so.

- Otherwise, the server forms an abstract database record by applying the abstract record structure defined by the default schema, and default element set name, for the database to which the record belongs.

The parameter Comp-spec may alternatively consist of a single composition specification with no database specified. In that case, for each record to be returned, if the server is able to form an abstract database record according to that composition specification, it does so. If not, an abstract database record is composed according to the default schema and default element set name for the database to which the record belongs.

The server applies a record syntax (which may be included in the composition specification or within the parameter Preferred-record-syntax) to the resulting abstract database record, to form a retrieval record.

3.6.2 Comp-spec Omitted

When requesting the retrieval of a set of records from a result set, if the parameter Comp-spec is omitted, the procedures of this section apply.

Notes:

1. This is always the case on a Search request, because the parameter Comp-spec is not included in the definition of the Search request.
2. This is always the case when version 2 is in force, because the parameter Comp-spec is not defined in version 2.

The Search request parameters Small-set-element-set-names and Medium-set-element-set-names, and the Present request parameter Element-set-names, take the form of a set of one or more pairs of a database name and associated element set name. For each record to be returned in the Search or aggregate Present response, the server first applies the abstract record structure defined by the default schema for the database to which the record belongs, to form an abstract database record, and then applies an element set name, as follows:

- If the database to which the record belongs is specified (as a component of one of the pairs), and if the corresponding element set name is valid for the default schema for the database, then the server applies that element set name.
- If not, the server applies the default element set name for the database.

Each of these parameters may alternatively consist of a single element set name with no database specified. In that case, for each record to be returned, if the element set name is valid for the default schema for the database to which the record belongs, the server applies that element set name; if not, the server applies the default element set name for the database.

A server must always recognize the character string "F" as an element set name to mean "full"; when it is applied to an abstract database record, it results in the same abstract database record (i.e. a null transformation).

A server must always recognize the character string "B" as an element set name to mean "brief" record. This standard does not define the meaning of "brief." Unless the client knows the server's definition of "brief" for a given schema, it should not assume that any particular elements are included.

Element set names are case-insensitive. (See the discussion of case-sensitivity of database names and result set names in 3.2.2.1.3. Element set names, as database names, are often passed around outside of the protocol, for example, in profiles.)

The client may specify a "preferred-record-syntax," which the server applies (to the abstract database record formed by the application of the element set name) to form a retrieval record. If the client does not specify a preferred-record-syntax, the server may select one (see 3.2.2.1.5).

3.6.3 Record Syntax

For each record to be returned in a Search or aggregate Present response, the element set name, or the schema and element specification from the composition specification, results in an abstract database record, as described above. To that abstract database record, the server applies a record syntax, indicated as described above. The term "record syntax" has the following meaning:

- When specified by the client (either as the value of Preferred-record-syntax or within a composition specification), it takes the form of an OID and refers to an abstract syntax (paired, or to be paired by the server, with a transfer syntax) that the client requests the server use for retrieval records.
- When specified by the server, it takes the form of an OID or p-context accompanying a retrieval record in a Search or Present response, and it refers to an abstract syntax paired with a transfer syntax.

When Server cannot Supply a Record According to Requested Syntax

When Preferred-record-syntax is supplied (and Comp-spec is not supplied, or if it is supplied, no record syntax ids are included within) and a particular record is not available in the requested syntax , the server should return a surrogate diagnostic such as 238: "Record not available in requested syntax", or a variation such as 1070: "user not authorized to receive record(s) in requested syntax" (used as a surrogate). The server may fail the request in the case where ***none*** of the requested records is available in the requested syntax (227: "No data available in requested record syntax "), or when the syntax is not supported (239: "Record syntax not supported"), or user is not authorized (1070, used as a non-surrogate). In any case, whenever preferredRecordSyntax is supplied, the server should not supply any records in any other syntax.

When Requested Syntax is not Supplied

When Preferred-record-syntax is not supplied (and Comp-spec is not supplied, or if it is supplied, no record syntax ids are included within) the server may interpret the omission to mean that the client wishes the server to select an appropriate syntax (which may be different for different record). However the server is free to treat this case as it sees fit; it may:

- Fail the request (with diagnostic 1071: "preferredRecordSyntax not supplied" used as a non-surrogate, diagnostic; or 1069: "No syntaxes available for this request");
- Select a syntax for some records and decline to select a syntax for others (with diagnostic 1071, used as a surrogate diagnostic); or

- Select a syntax for each record.

3.7 Type-1 and Type-101 Queries

This section specifies procedures when Query-type is 1 (or 101; see Note 2 below). Type-1 is the "Reverse Polish Notation" (RPN) query. It has the following structure:

```
RPN-Query   ::= Argument | Argument + Argument + Operator

Argument    ::=   Operand | RPN-Query

operand     ::=   AttributeList + Term | ResultSetId | Restriction

Restriction ::=   ResultSetId + AttributeList

operator    ::=   AND | OR | AND-NOT | Prox
```

The notation above is used as follows:

- ::= means "is defined as"
- | means "or"
- + means "followed by", and + has precedence over | (i.e. + is evaluated before |).

Notes:

1. For type-1, the Prox operator and the Restriction operand are defined for version 3 only. When version 2 is in effect, it is a protocol error to include either the Prox operator or Restriction operand in a type-1 query.
2. The type-101 query is defined as identical to the type-1 query, with the exception that the Prox operator and Restriction operand are defined not only for version 3, but for version 2 as well. Thus the definition of the type-101 query is independent of version.

A Z39.50-conforming server must support the type-1 query, but support of the type-1 query does not imply support of any of the defined operators or operands.

The server designates what query types it supports, and which operators and operands.

Note: The server designates this information either through the Explain facility or through some mechanism outside of the standard.

If the server claims support for the Prox operator, the server should also designate whether it supports the extended result set model for proximity (the extended result set model for searching as described in 3.1.6 and its specialization for proximity as described in 3.7.2.2). If the server claims support for the Restriction operand, then it must also support the extended result set model for restriction (the extended result set model for searching and its specialization for restriction as described in 3.7.3).

Note: Only in certain circumstances (detailed below) does support of the Prox operator require support of the extended result set model for proximity. However, support of the Restriction operand always requires support of the extended result set model for Restriction.

3.7.1 Representation and Evaluation of the Type-1 and Type-101 Queries

At the client, the query is represented by a tree. Each subtree represents an operand, either a simple operand or a complex operand. Each leaf node represents a simple operand: Result-set-id, AttributeList+ Term, or Restriction. Each non-leaf node represents a complex operand: a subtree whose root is an operator, and which contains two subtrees, a left operand and a right operand.

The client traverses the tree according to a left post-order traversal, to produce a sequence of (simple) operands and operators, which is transmitted to the server.

At the server, evaluation of the sequence of operands and operators is illustrated by the use of a stack. Whenever an operand is encountered, it is put on the stack. Whenever an operator is encountered, the last two objects that have been put on the stack are pulled off and the operator is applied as follows.

Each operand represents a set of database records. Each is one of the following:

(a) AttributeList+term -- In which case it represents the set of database records obtained by evaluating the specified attribute-set and term against the collection of databases specified in the Search request.

(b) ResultSetId -- In which case it represents the set of database records represented by the transient result set identified by ResultSetId.

(c) Restriction operand (ResultSetId+AttributeList): In which case it represents the set of database records represented by the result set identified by ResultSetId, restricted by the specified attribute set (see 3.7.3).
Note: If the Restriction operand occurs the server must support the extended result set model for restriction; otherwise the query is in error.

(d) An intermediate result set (resulting from a previous evaluation placed on the stack) -- In which case it represents the records identified by that result set.

Let S1 and S2 be the sets represented by the left and right operand respectively. Let S be defined as follows:

- If the operator is AND, S is the intersection of S1 and S2
- If the operator is OR, S is the union of S1 and S2
- If the operator is AND-NOT, S is the set of elements in S1 which are not in S2
- If the operator is Prox:
  - If both operands are of form (a) S is the subset of records in the set (S1 AND S2) for which A ProxTest B is true (see 3.7.2.1) where A and B are the two operands.
  - Otherwise:
    - The server must support the extended result set model for proximity; or else the query is in error.
    - Let R1 and R2 be result sets representing the sets S1 and S2 (i.e. each is either: the result set specified by the corresponding operand, if it was of form (b), or the hypothetical result set representing the set of records represented by that operand, otherwise.
  - In either case, both R1 and R2 are assumed to conform to the extended result set model for proximity.)

- Each entry in R1 and R2 contain positional information, in the form of position vectors. For each record represented by both R1 and R2, consider every ordered pair consisting of a position vector associated with the record as represented in R1 and a position vector associated with the record as represented in R2. For each pair that qualifies according to the ProxTest:
    - The record is qualified into the set S; and
    - A position vector is created for that record as represented in the resultant set, composed from that ordered pair.

An intermediate result set is created, which represents the records in the set S, and is put on the stack. When evaluation of the query is complete (i.e. all query-terms have been processed) there will be one object remaining on the stack (otherwise the query is in error), representing a set of database records, which is the result of the query.

Example

Suppose databaseName = A; resultSetName = 1; query = " 'dog' ". Subsequently, a follow-on query specifies

databaseName = B; resultSetName = 2; query = " 'dog' OR 'resultSet 1' ". That is, the second query asks for records from database B containing the word "dog" or records from result set 1 (result set 1 contains records from database A). The latter search, though it pertains only to database B (and there are no records from result set 1 that belong to database B) should nonetheless include records from A. (See 3.7.1 (a) and (b). Note that when an operand is of the form AttributesPlusTerm, it is evaluated against the specified databases, but when an operand is of the form resultSetName, it is not.)

3.7.2 Proximity

3.7.2.1 The Proximity Test

The proximity test, ProxTest, includes a Distance, Relation, Unit, and two boolean flags: Ordered and Exclusion.

- Distance: Difference between the ordinal positional values of the two operands. (E.g., if unit is 'paragraph,' distance of zero means "same paragraph".) Distance is never negative.
- Relation: LessThan, LessThanOrEqual, Equal, GreaterThanOrEqual, GreaterThan, or NotEqual.
- Unit: Character, Word, Sentence, Paragraph, Section, Chapter, Document, Element, Subelement, ElementType, Byte, or a privately defined unit.
- Ordered flag: If set, the test is for "right" proximity only (the left ordinal must not exceed the right ordinal and Distance is compared with the difference between the right and left ordinals); otherwise, the test is for "right" or "left" proximity. (Distance is compared with the absolute value of the difference between the left and right ordinals.)
- Exclusion flag: If set, "not" is to be applied to the operation (for example if the test with Exclusion flag 'off' is "'cat' within 5 words of 'hat'," then the same test with Exclusion flag 'on' is "'cat' not within 5 words of 'hat'").

Example:

Suppose A and B respectively specify "personal name = 'McGraw,J.' " and "personal name = 'Stengel, C.' ," and:

- Distance is 0,
- Relation is 'equal,'
- Proximity-unit is 'paragraph',
- Ordered flag is 'false',
- Exclusion flag is 'false'.

Then the result is the set of records in which both of the personal names occur within the same paragraph. Using the same example, if the Exclusion flag is set to 'true,' the result is the set of records in which the two personal names never both occur within the same paragraph.

If the Ordered flag is set to 'true' (and Exclusion flag to 'false') then the result is the set of records in which the personal name 'McGraw, J.' occurs within the same paragraph as, but before, the personal name 'Stengel, C.'.

If distance is instead 1 ('ordered' and 'exclusion' flag 'false') the result is the set of records in which the two personal names occur in adjacent paragraphs. If, in addition, Relation-type is 'less-than-or-equal' the result is the set of records in which the two names occur within the same or adjacent paragraphs.

3.7.2.2 Extended Result Set Model for Proximity

In the extended result set model for proximity, the server maintains positional information, in the form of one or more position vectors, associated with each record represented by the result set, which may be used in a proximity operation as a surrogate for the search that created the result set.

Example:

Let R1 and R2 be result sets produced by type-1 query searches on the terms 'cat' and 'hat'. In the extended result set model for proximity, the server maintains sufficient information associated with each entry in R1 and with each entry in R2 so that the proximity operation "R1 near R2" would be a result set equivalent to the result set produced by the proximity operation "cat near hat" ("near" is used here informally to refer to a proximity test).

The manner in which the server maintains this information is not prescribed by the standard. Appendix ERS (non-normative) provides examples.

An implementation may support proximity without supporting the extended result set model for proximity. For example, it might support "cat near hat" and not support "R1 near R2" (where R1 and R2 are result sets representing 'cat' and 'hat' respectively).

3.7.3 Restriction and the Extended Result Set Mode l

The Restriction operand specifies a result-set-id and a set of attributes, and it represents the set of database records identified by the specified result set, restricted by the specified attributes.

Example:

Let R be the result set produced by a search on the term 'cat,' representing three records:

1. Where 'cat' occurs in the title,
2. Where 'cat' occurs in the title and as an author, and
3. Where 'cat' occurs in the title, as an author, and subject.

Then "R restricted to 'author"' might produce the result set consisting of the entries 2 and 3 of R.

In the extended result set model for restriction, the server maintains information associated with each record represented by the result set, that may be used in the evaluation of a restriction operand as a surrogate for the search that created the result set. The manner in which the server maintains this information is not prescribed by the standard. Appendix ERS (non-normative) provides examples.

If an implementation supports result set restriction, then it implicitly supports the extended result set model for restriction (the model is implicit in the semantics). However, supporting the extended model for restriction is an abstract concept for which there is no conformance requirement. It simply means that the server maintains some information (necessary to carry out the operation) and the manner in which the information is maintained is not prescribed by this standard.

4. Protocol Specification

This section (4) specifies the formats, procedures, and conformance requirements for the Z39.50 protocol, governing the transfer of information between a Z39.50 client/server pair. Sections 4.1 and 4.2 respectively describe the formats and rules for exchange of Z39.50 application protocol data units (APDUs). An APDU is a unit of information, transferred between client and server, whose format is specified by the Z39.50 protocol, consisting of application-protocol-information and possibly application-user-data. Sections 4.3 and 4.4 respectively describe rules for extensibility and conformance requirements.

4.1 Abstract Syntax and ASN.1 Specification of Z39.50 APDUs

This section describes the abstract syntax of the Z39.50 APDUs (Application Protocol Data Units). An APDU is a unit of information transferred between a client and server. The abstract syntax is is described using the ASN.1 notation defined in ISO 8824.

The comments included within the ASN.1 specification are part of the standard.

See Appendix 18.

4.2 Protocol Errors

Syntactical errors in received APDUs are considered to be protocol errors (with the following exceptions: Unknown data elements, and unknown options within the Options data element, will be ignored on received Init APDUs). Incorrectly formatted APDUs or APDUs with invalid data are also considered to be protocol errors, as are incorrectly sequenced messages. Additional conditions that may be treated as protocol errors are described in 4.4.2.2.

This standard does not specify the actions to be taken upon detection of protocol errors. A system detecting a protocol error may:

- issue a Close request, with close-reason ‘protocol error’ (when version 3 is in force);
- Terminate the connection; or
- Ignore the error.

4.3 Encapsulation

Z39.50 APDUs may be included, or ***encapsulated,*** within other APDUs. The rules for encapsulation are as follows.

1. Encapsulation is negotiated during initialization. Option bit 15 is designated. If the client sets option bit 15 in the Init request, then:
   - The client proposes that "encapsulation be in effect" for the Z-association (or limited

to the Init operation if version 2 is proposed; see rule 5 below).

- The Init request may also (but need not) include one or more APDUs as described in 2.
- If the server also sets option bit 15, then encapsulation is in effect for the Z-association (or limited to the Init operation, if version 2 is negotiated). In that case, if the Init request had included encapsulated APDUs, the server's behavior (as concerns the Init response) is as described below.
- If the server does not set option bit 15, encapsulation is not in effect. (If the server does not even recognize option bit 15, it will ignore it and not set the bit in the response, as per the general rules for negotiation.) In this case if the server includes information that appears to be encapsulated APDUs in the Init response, the client should not assume that the information is encapsulated APDUs, and this specification does not prescribe client behavior in this situation.

2. If encapsulation is in effect, then within an operation-initiating APDU the client may include, within otherInfo (or, if the APDU is Init, within the userInfo parameter using UserInfo-1; see Appendix USR) to simulate OtherInfo; (see Use of Init Parameters for User Information in Appendix USR), one or more APDUs, including any operation-initiating APDU (excluding Init), optionally followed by Close. These are referred to as "encapsulated APDUs".
   The first APDU in this chain of nested APDUs (i.e. the one that is not encapsulated in any other) is called the “base APDU”.
   The first encapsulated PDU is included in the otherInfo parameter of the base PDU (or in the userInfo parameter, if the base APDU is Init). The next encapsulated APDU is included in the otherInfo parameter of the first encapsulated APDU, and so on. Each encapsulated APDU is included in otherInfo as CHOICE externallyDefinedInfo, where the oid for the external is 1.2.840.10003.2.1 (the oid for Z39.50 APDUs).
3. There may be at most a single occurrence of an encapsulated APDU within the base APDU, and at most a single occurrence of an encapsulated APDU within an encapsulated APDU (thus arbitrary nesting is permitted, but multiple threads are not). When this is not the case, the prescribed behavior is not defined by the protocol.
4. The use of the otherInfo and/or userInfo parameter is specified in order to support encapsulation within version 2 or 3. In a future version of the protocol, there might be explicit encapsulation parameters in certain APDUs.
5. For version 2, encapsulation may be supported only within the Init operation. Thus, when version 2 is negotiated, encapsulation may also be negotiated, but for use only within Init.
6. When the client includes encapsulated APDUs the intent is that the server execute the APDUs serially in the order that they are nested (from shallowest to most deeply nested), and include APDU responses similarly nested in the APDU response to the base APDU, and formatted in the manner described in (2), that is, with each encapsulated APDU included in otherInfo as CHOICE externallyDefinedInfo, where the oid for the external is 1.2.840.10003.2.1.
7. If encapsulation is in effect, the server may not simply ignore encapsulated APDUs. The server may choose not to execute encapsulated APDUs, but if so, must include in the response to the base APDU an appropriate diagnostic (see 10), for example: "this specific sequence of APDUs is not supported".
8. Based on pre-screening analysis, the server may decide to execute neither the base APDU nor any encapsulated APDUs, for example in the case where in the server's

opinion the client intent was that if the server did not expect to be able to execute the full sequence of APDUs, then it should not attempt to execute any of them. Similarly, in this case the server should include in the response to the base APDU an appropriate diagnostic (see 10).

9. If encapsulation is in effect, the server may choose to execute some but not all of a sequence of nested APDUs. In that case it should not execute any encapsulated APDU following one that it chose not to execute (and similarly should include an appropriate diagnostic in the base-APDU response, see 10). Thus if there are N encapsulated APDUs, the server will always execute either none, or the first M APDUs (M less than or equal to N). The server should also not execute any APDUs following a APDU that it attempted to execute but execution failed.
10. The server may use the General Diagnostic Container format (see Appendix DIAG) to supply diagnostics in these cases described in 7, 8, and 9.
11. Access-control and resource-control apply to the operation initiated by the base APDU, from a modeling perspective. However, as a practical matter, Access-control and Resource-control formats may be developed to allow a server to indicate to what specific encapsulated APDU the Access-control or Resource-control request pertains. Similarly, for trigger-resource-control: the client might send a base APDU with several APDUs nested, and later send a trigger-resource-control request. From a modeling perspective it applies to the base APDU. But the request could actually be asking (based on the report-id requested) "how far into the sequence are you?" and subsequently the client might issue a second trigger request specific to the embedded request, which (again, from an operation model perspective) would still apply to the base APDU.
12. Encapsulation is not intended to support segmentation.

4.4 Conformance

A system claiming to implement the procedures in this standard shall comply with the conformance requirements in 4.4.1. These requirements are elaborated in 4.4.2.

4.4.1 General Conformance Requirements

The system shall:

(a) Act in the role of client or server
(b) Support the Init, Search, and Present services. See 4.4.2.2.1
(c) Support the syntax in 4.1
(d) Support the Type-1 Query. See 4.4.2.2.2
(e) Support (at minimum) version 2 of the protocol
(f) Follow the procedures specified in sections 3 and 4
(g) Assign values to APDU data elements according to the procedures of sections 3 and 4.1

4.4.2 Specific Conformance Requirements

4.4.2.1 provides a table of Z39.50 features for which 4.4.2.2 specifies conformance requirements. In particular, conformance requirements are described as they pertain to version 2 and version 3 respectively.

4.4.2.1 Z39.50 Features

The following table of Z39.50 features indicates the applicable protocol version (2 or 3), a reference to a description of the feature, and a reference to the section within 4.4.2.2 that describes conformance requirements for the feature. The "item" column is used by the sections within 4.4.2.2 to refer back to the table.

Item	Feature	Version	Reference	Conformance
1	**Init Service**	V2 and V3	3.2.1.1	4.4.2.2.1
2	**Search Service**	V2 and V3	3.2.2.1	4.4.2.2.1
3	Query type-1	V2 and V3	3.7	4.4.2.2.2
4	Multiple attribute sets	V3	Note 1	4.4.2.2.3
5	Multiple data types for search term	V3	Note 2	4.4.2.2.3
6	Complex attribute values	V3	Note 3	4.4.2.2.3
7	Result set restriction	V3	3.7	4.4.2.2.3
8	Proximity	V3	3.7.2	4.4.2.2.4
9	Query type-101	V2 and V3	3.7	4.4.2.2.4
10	Query types 0, 2, 100	V2 and V3	3.2.2.1.1	4.4.2.2.4
11	Query type 102	V3	3.2.2.1.1	4.4.2.2.5
12	Additional-search-information parameter in Search request and response	V3	3.2.2.1.12	4.4.2.2.6
13	Named result sets	V2 and V3	3.2.2.1.3	4.4.2.2.23
14	**Present Service**	V2 and V3	3.2.3.1	4.4.2.2.1
15	Additional-ranges and Comp-spec parameters on Present request	V3	3.2.3.1.2, 3.2.3.1.6	4.4.2.2.7
16	Max- segment-count, -segment-size, -record-size parameters on Present request	V3	3.2.3.1.7	4.4.2.2.8
17	Diagnostic format -- default form	V2 and V3	Note 4	4.4.2.2.9
18	Diagnostic format -- external form	V3	Note 4	4.4.2.2.9
19	addinfo type VisibleString	V2, V3	Note 5	4.4.2.2.10
20	addinfo type InternationalString	V3	Note 5	4.4.2.2.10
21	Multiple non-surrogates in Search or Present response	V3	Note 6	4.4.2.2.11
21A	String identifier for schema	V3 (2001 only)	Note 10	4.4.2.2.30

Item	Feature	Version	Reference	Conformance
22	**Segment Service**	V3	3.2.3.2	4.4.2.2.12
23	Level-1 segmentation	V3	3.3.2	4.4.2.2.12
24	Level-2 segmentation	V3	3.3.3	4.4.2.2.12
25	**Delete Service**	V2 and V3	3.2.4.1	4.4.2.2.13
26	failure-10 value of Delete-list-status on Delete response	V3	3.2.4.1.4	4.4.2.2.15
27	**Access-control Service**	V2 and V3	3.2.5.1	4.4.2.2.14
28	Security-challenge-response and diagnostic in Access-control response	V3	Note 7	4.4.2.2.16
29	**Resource-control Service**	V2 and V3	3.2.6.1	4.4.2.2.14
30	**Trigger-resource-control Service**	V2 and V3	3.2.6.2	4.4.2.2.13
31	**Resource-report Service**	V2 and V3	3.2.6.3	4.4.2.2.13
32	Op-id parameter of Resource-report-request	V3	3.2.6.3.2	4.4.2.2.17
33	failure-5 and failure-6 values of Resource-report-status in Resource-report response	V3	3.2.6.3.3	4.4.2.2.18
34	**Sort Service**	V2 and V3	3.2.7.1	4.4.2.2.13
34.1	Result-count parameter of Sort Response	V2 and V3	3.2.7.1.7	4.4.2.2.26
35	**Scan Service**	V2 and V3	3.2.8.1	4.4.2.2.13
36	**Extended-Services Service**	V2 and V3	3.2.9.1	4.4.2.2.13
37	**Close Service**	V3	3.2.11.1	4.4.2.2.19
38	Explain facility	V2 and V3	3.2.10	4.4.2.2.20
39	Other-information (in a request or response other than Scan, Sort, or Extended Services)	V3	Note 8	4.4.2.2.6
40	Other-information in Scan, Sort, and ES	V2 and V3	Note 8	4.4.2.2.21
41	Concurrent Operations	V3	3.5	4.4.2.2.22
42	InternationalString full use of	V3	Note 9	4.4.2.2.24

Item	Feature	Version	Reference	Conformance
	GeneralString repertoire			
43	Reference Id	V2 and V3	3.4	4.4.2.2.25
44	Duplicate Detection Service	V2 and V3	3.2.12	4.4.2.2.13
45	Negotiation Model	V2 and V3	Appendix NEGO	4.4.2.2.27
46	Query type 104	V2 and V3	3.2.2.1.1	4.4.2.2.28
47	Encapsulation	V2 and V3	4.2.4	4.4.2.2.29

Notes:

1. In version 2 a type-1 query includes a single, global attribute set id, which identifies an attribute set definition that pertains to all of the attributes within the query. In version 3 a type-1 query also includes a global attribute set id, but in addition, each attribute within the query may also be qualified with an attribute set id (which, If included, overrides the global attribute set id).
2. In version 2 a search term must be of ASN.1 type OCTET STRING. In version 3 it may be any of the following: OCTET STRING, INTEGER, InternationalString, OBJECT IDENTIFIER, GeneralizedTime, EXTERNAL, IntUnit, or NULL.
3. In version 2, in a type-1 query, an attribute value must be numeric (i.e. ASN.1 type INTEGER). In version 3, an attribute value may be numeric or 'complex'. The complex form may include multiple values, each either numeric or character string, and a semantic action indicator (corresponding to some semantic action defined within the attribute set definition).
4. See introductory text of Appendix DIAG.
5. In version 2, when using default diagnostic format, the addInfo parameter must be ASN.1 type VisibleString. In version 3 it may be type InternationalString.
6. In version 2, a Search or Present response may include at most a single non-surrogate diagnostic record. In version 3 a Search or Present response may include multiple non-surrogate diagnostic records. (Responses other than Search or Present that include diagnostics may include multiple non-surrogate diagnostics regardless of version.)
7. In version 2, in the Access control response, securityChallengeResponse must occur, and no diagnostic may occur. In version 3, securityChallengeResponse may be omitted, if the parameter 'diagnostic' is present.
8. In version 2, the parameter otherInformation may be used only in Scan, Sort, and Extended Services requests and responses. In version 3 it may be used in any request or response.
9. See definition of InternationalString in ASN.1 for APDUs.
10. The Z39.50 Present request may include a schema identifier allowing the client to request that records be supplied according to a specific schema. In Z39.50-1995 the identifier must be an ISO object identifer (OID), and it is assumed that the schema is a GRS-1 compatible schema (i.e. that the record syntax will be GRS-1). Z39.50-2003 allows the identifier to be a string (or an OID).

4.4.2.2 Detailed Requirements

4.4.2.2.1 Init, Search, and Present Services

(See items 1, 2 and 14 above.)

A system must support the Init, Search, and Present services.

This means that a client must be capable of sending Init, Search, and Present requests and receiving the respective responses. A server must respond properly to Init, Search, and Present requests with respective responses.

A client may indicate (via option bits) during initialization that it does not intend to utilize the Present service during the Z-association; this does not constitute non-conformance. If, however, a client indicates that it does intend to utilize the Present service, and the server refuses, this does constitute non-conformance on the part of the server.

This requirement is independent of version.

4.4.2.2.2 Type-1 Query

(See item 3 above.)

A client must be capable of formulating a type-1 query within a Search request, and a server should expect to receive a type-1 query.

A client or server may support other query types. If the client fails to send a type-1 query during a Z-association, this does not constitute non-conformance on the part of the client. If, however, the client does send a type-1 query and the server responds with a diagnostic indicating "query type not supported" this does constitute non-conformance on the part of the server.

This requirement does not mean that any specific feature of the type-1 query must be supported. A server that receives a type-1 query that conforms to the type-1 query syntax but which includes a feature that it does not support must not treat this condition as a protocol error (but instead should return an appropriate diagnostic, however, that diagnostic must not indicate "query type not supported").

This requirement is independent of version.

4.4.2.2.3 Multiple Attribute Sets, Multiple Data Types for Search Term, Complex Attribute Values, Result Set Restriction, and Proximity

(See items 4, 5, 6, 7, and 8 above.)

For version 2, the client may not use any of these features in a type-1 query. If server receives a type-1 query with any of these features, it may treat this condition as a protocol error.

For version 3, the client may but is not required to use any of these features in a type-1 query. The server should expect type-1 queries to include any or all of these features, but is not required to support any of these features. If the server receives a type-1 query which includes any of these

feature that it does not support, it must not treat this condition as a protocol error (but rather should return an appropriate diagnostic).

4.4.2.2.4 Query Types 0, 2, 100, and 101

(See items 9 and 10 above.)

A client is not required to support queries of any of these types. A server should expect to receive, but need not support queries of these types. If a server receives a query of one of these types that it does not support it must not treat this condition as a protocol error but instead should return a diagnostic indicating that the query type is not supported.

This requirement is independent of version.

4.4.2.2.5 Query Type-102

(See item 11 above.)

For version 2, a client may not use the type-102 query. If a server receives a type-102 query it may treat this condition as a protocol error.

For version 3, a client may, but need not support the type-102 query. A server should expect to receive, but need not support, type-102 queries; if it receives a type-102 query it must not treat this condition as a protocol error.

Note: Z39.50 lists type-102 as a valid query type (for version 3) but does not include a definition.

4.4.2.2.6 Additional-search-information Parameter in Search Request or Response; Other-information Parameter in any Request or Response other than Scan, Sort, Extended Services, or Duplicate Detection

(See items 12 and 39 above.)

For version 2, a system may not use these parameters; if a system receives one of these parameters it may treat this condition as a protocol error.

For version 3, a system is never required to use any of these parameters. However, a system should expect to receive these parameters, but is not required to interpret or process the information contained within the any of these parameters.

4.4.2.2.7 Additional-ranges and Comp-spec Parameters on Present Request

(See item 15 above.)

For version 2, the client may not use these parameters. If the server receives one of these parameters it may treat this condition as a protocol error.

For version 3, the client is not required to, but may use either of these parameters. The server should expect to receive, but need not support either of these parameters. If the server receives

but does not support one of these parameters, it should not treat this condition as a protocol error (but instead should return an appropriate status value and/or diagnostic).

4.4.2.2.8 Max-segment-count, Max-segment-size, and Max-record-size Parameters on Present Request

(See item 16 above.)

For version 2, as well as for version 3 when segmentation is not in effect, the client may not use these parameters; if the server receives any of these parameters it may treat this condition as a protocol error.

For version 3:

- If level-1 segmentation is in effect:
  - The client may but is not required to support Max-segment-count. The server should expect to receive, but need not support Max-segment-count. If the server receives but does not support Max-segment-count, it must not treat this condition as a protocol error (but instead should return an appropriate status value and/or diagnostic).
  - The client may not use Max-segment-size or Max-record-size. If server receives either it may treat this condition as a protocol error.
- If level-2 segmentation is in effect:
  - The client may but is not required to support any of these three parameters. The server should expect to receive, but need not support any of these parameters. If the server receives but does not support a parameter, it must not treat this condition as a protocol error (but instead should return an appropriate status value and/or diagnostic).

4.4.2.2.9 Diagnostic Format

(See items 17 and 18 above.)

For version 2, the server may send diagnostics in a Search or Present response using the default form only. If the client receives a diagnostic which does not conform to the default form, it may treat this condition as a protocol error.

Note: This rule applies to Search and Present responses only. Responses other than Search or Present that include diagnostics are not affected.

For version 3, the server may send diagnostics using the default or external form. The client should expect to receive diagnostics in either form.

4.4.2.2.10 Addinfo of Default Diagnostic Format

(See items 19 and 20 above.)

For version 2, when the server sends a diagnostic in a Search or Present response using the default form, the addinfo parameter must be of ASN.1 type VisibleString. If the client receives a diagnostic that violates this rule, it may treat this condition as a protocol error.

For version 3 the addinfo parameter may be of either type VisibleString or InternationalString.

4.4.2.2.11 Multiple Non-surrogates in Search or Present Response

(See item 21 above.)

For version 2, the server must not include multiple non-surrogate diagnostics in a Search or Present response; if it does so, the client may treat this condition as a protocol error.

Note: This rule applies to Search and Present responses only. There are responses other than Search or Present that include diagnostics, and these are not affected.

For version 3, the server may (but is not required to) include multiple non-surrogate diagnostics in a Search or Present response and if it does, the client must not treat this condition as a protocol error.

4.4.2.2.12 Segmentation

(See items 22, 23, and 24 above.)

For version 2, as well as for version 3 when segmentation is not in effect, the server may not send a Segment request, and if it does, the client may treat this condition as a protocol error.

For version 3, level-1 or level-2 segmentation may be negotiated, however neither the server nor the client is required to support segmentation.

4.4.2.2.13 Delete Service, Trigger-resource-control Service, Resource-report Service, Sort Service, Scan Service, Extended-Services Service, and Duplicate Detection Service

(See items 25, 30, 31, 34, 35, 36, and 44 above.)

A system is not required to support any of these services. They are independently negotiable. If the server receives a request of one of these types and the respective service is not in effect, it may treat this condition as a protocol error.

This requirement is independent of version.

4.4.2.2.14 Access-control and Resource-control Services

(See items 27 and 29 above.)

A system is not required to support either of these services. They are independently negotiable. If the client receives an Access-control or Resource-control request and the respective service is not in effect (or if the request occurs while the client is awaiting an Init response and the client has not proposed the respective option in the Init request), it may treat this condition as a protocol error.

This requirement is independent of version.

4.4.2.2.15 'failure-10' value of Delete-list-status on Delete Response

(See item 26 above.)

For version 2, the server may not return this value; if it does the client may treat this condition as a protocol error.

For version 3, the server may return this value.

4.4.2.2.16 Security-challenge-response and Diagnostic in Access-control Response

(See item 28 above.)

For version 2, the client must include in the Access-control response the parameter Security-challenge-response, and may not include a diagnostic. If the server receives an Access-control response that violates this rule it may treat this condition as a protocol error.

For version 3, the client may include a diagnostic, and if so, the parameter securityChallengeResponse may be omitted.

4.4.2.2.17 Op-id Parameter of Resource-report Request

(See item 32 above.)

For version 2, the client may not use this parameter; if the server receives this parameter it may treat this condition as a protocol error.

For version 3, the client may, but is not required to include this parameter. The server should expect to receive, but need not support the parameter. If the server receives but does not support this parameter, it should not treat this condition as a protocol error (but instead should return an appropriate status).

4.4.2.2.18 failure-5 and failure-6 Resource-report-status in Resource-report Response

(See item 33 above.)

For version 2, the server may not return either value for this status; if it does the client may treat this condition as a protocol error.

For version 3, the server may return either value.

4.4.2.2.19 Close Service

(See item 37 above.)

For version 2, the Close service may not be used. If a system receives a Close request, it may treat this condition as a protocol error.

For version 3, a system must expect to receive a Close request, and must be capable of responding with a Close response. A system is not required to send a Close request.

4.4.2.2.20 Explain Facility

(See item 38 above.)

There are no conformance requirements pertaining to the Explain facility, either for version 2 or version 3. A system may choose to support or not support Explain.

Note that implementation of Explain requires, at minimum, support for searching the Explain database and for the Explain record syntax. This standard does not require support for searching any particular database or support for any particular record syntax.

4.4.2.2.21 Other-information Parameter in Scan, Sort, Extended Services, and Duplicate Detection Request

(See item 40 above.)

The parameter Other-information may occur in a Scan, Sort, Extended Services or Duplicate Detection request or response. A system should expect to receive this parameter, but is not required to interpret or process the information contained within the parameter.

This requirement is independent of version.

4.4.2.2.22 Concurrent Operations

(See item 41 above.)

For version 2, as well as for version 3 when concurrent operations is not in effect, if a client attempts to initiate concurrent operations (i.e. attempts to initiate an operation when an operation is already active), the server may treat this as a protocol error.

For version 3, a system may choose to support or not to support concurrent operations.

4.4.2.2.23 Named Result Sets

(See item 13 above.)

A system may choose to support or not support named result sets.

If the server receives a Search request where the value of the parameter Result-set-id is other than 'default' and the server does not support named result sets:

- If version 2 is in effect, the server should not treat this condition as a protocol error but should instead return an appropriate diagnostic.
- If version 3 is in effect, the server may treat this condition as a protocol error or may instead return an appropriate diagnostic.

4.4.2.2.24 InternationalString Definition

(See item 42 above.)

For version 2, a value of a parameter of ASN.1 type InternationalString must conform to the VisibleString definition. A system which receives a value that violates this rule may treat this condition as a protocol error.

For version 3, a value of a parameter of ASN.1 type InternationalString must conform to the GeneralString definition. A system which receives a value that does not conform to the VisibleString definition (but does conform to the GeneralString definition) must not treat this condition as a protocol error.

4.4.2.2.25 Reference-id

(See item 43 above.)

For both version 2 and version 3, a client may choose to support or not support the Reference-id parameter; a server must support the Reference-id parameter. Note, however, for version 3, client support of concurrent operations (see 4.4.2.2.23) implies support for the reference-id parameter.

4.4.2.2.26 Result-count Parameter of Sort Response

(See item 34.1 above.)

A client may support the Sort service and still not recognize this parameter (because it is newly defined in Z39.50-2003), as long as option bit 16 is not negotiated. Thus if option bit 16 is not negotiated and the client receives this parameter it may consider this to be a protocol error. If option bit 16 is negotiated, the client must recognize this parameter. The server is never required to support this parameter (thus even if option bit 16 is negotiated the server is never obligated to send this parameter).

This requirement is independent of version.

4.4.2.2.27 Negotiation Model

(See item 44 above.)

Neither the client nor server is required to support the negotiation model. However, if the client and server both set the "negotiation model" option bit both signify adherence to the model and may assume that negotiation is carried out in accordance with the model.

This requirement is independent of version.

4.4.2.2.28 Query Type 104

(See item 46 above.)

Neither the client nor server is required to support the type-104 query. If the type-104 option bit is negotiated, the client may send type-104 queries, and the server must recognize type-104 queries but is not required to support any specific external query definition.

This requirement is independent of version.

4.4.2.2.29 Encapsulation

(See item 47 above.)

Neither the client nor server is required to support encapsulation, it is a negotiated feature. Rule for negotiation of encapsulation are supplied in 4.3.

4.4.2.2.30 String Indentifier for Schema

(See item 21A above.)

Neither the client nor server is required to support this, it is a negotiated feature. If option bit 21 is negotiated, the client may send string identifiers for schemas, and the server must recognize them as valid identifiers but is not required to support any specific schemas.

4.4.3 Z39.50 Version 3 Baseline Requirements

This section details the minimum requirements, beyond version 2, for a Z39.50 implementation to claim conformance to version 3, that is, to indicate in an Init request or response that it supports version 3.

The Version 3 Baseline Requirements include core and conditional requirements. Core requirements apply to all version 3 implementations. Conditional requirements pertain to features that are optional in version 2. These are requirements that are applicable in version 3 only if a particular, optional feature is implemented, and only if that feature was part of version 2. Only the Delete, Access-control, and Resource-report Services are affected.

Requirements are listed separately for client and server.

In the rules below, the terms ***accept***, ***recognize*** and ***support*** have the following meaning:

- **Accept**: Rules stating that a system must accept a particular object mean that the system must not declare a protocol error due to receipt of that object; the system is not required to support the function (see “support” below) associated with that object
- **Recognize**: Rules stating that a system must recognize a particular object mean that the system must not declare a protocol error due to receipt of that object, and must either support the function associated with the object (see below) or must respond appropriately (e.g. with a diagnostic) that it does not support the function.
- **Support**: Rules stating that a system must support a particular function mean that the system must implement the function and exhibit behavior prescribed in the standard pertaining to that function.

4.4.3.1 Core Requirements

4.4.3.1.1 V3 Core Requirements for Client

1. The client must accept the parameter additionalSearchInfo in a Search response
2. The client must accept both the VisibleString and InternationalString form of the addInfo parameter within a diagnostic record
3. The client must accept diagnostics in both the default and external forms
4. The client must accept multiple non-surrogate diagnostic records in a Search or Present response
5. The client must accept the parameter otherInfo in any APDU
6. The client must support receipt of a Close request
7. In the absence of character-set negotiation, the client must accept all values conforming to the GeneralString definition for parameters of ASN.1 type InternationalString

4.4.3.1.2 V3 Core Requirements for Server

1. The server must recognize search terms in a type-1 query of type OCTET STRING, INTEGER, InternationalString, OBJECT IDENTIFIER, GeneralizedTime, EXTERNAL, IntUnit, or NULL.
2. The server must recognize within a type-1 query (in addition to the global attribute set id) an attribute set id qualifying any individual attribute within the query
3. The server must recognize the operand 'resultAttr' within a type-1 query
4. The server must recognize the operator 'prox' within a type-1 query
5. The server must recognize (in addition to numeric attribute values) attribute values of form 'complex'(as defined in the ASN.1 for APDUs) within a type-1 query
6. The server must recognize a type-102 query
7. The server must accept the parameter additionalSearchInformation in a Search request
8. The server must recognize the parameter AdditionalRanges on a Present request
9. The server must recognize the 'complex' (in addition to the 'simple') form of the parameter recordComposition on a Present request
10. The server must accept the parameter otherInfo in any APDU
11. The server must support receipt of a Close request
12. In the absence of character-set negotiation, the server must accept all values conforming to the GeneralString definition for parameters of ASN.1 type InternationalString

4.4.3.2 Conditional Requirements

4.4.3.2.1 V3 Conditional Requirements for Client

If the client implements the Delete service as well as concurrent operations: the client must accept a value of 'failure-10' for the Delete-list-status on Delete response.

If the client implements the Resource-report service, the client must accept a value of failure-5 or failure-6 for Resource-report-status in Resource-report response.

4.4.3.2.2 V3 Conditional Requirements for Server

If the server implements access control, the server must accept an Access-control response where the parameter Security-challenge-response is omitted and which includes a dagnostic.

If the server implements resource control, the server must recognize the parameter OpId of the Resource-report request.

Appendix 1 OID: Z39.50 Object Identifiers

Normative

OID.1 Object Identifier Assigned to This Standard

The following ISO object identifier has been assigned to this standard:

{iso (1) member-body (2) US (840) ANSI-standard-Z39.50 (10003)}

Note: This OID was originally assigned to Z39.50-1992; it applies also to Z39.50-1995 and Z39.50-2003.

OID.2 Object Classes

The following values, corresponding to object classes, are registered at the level immediately subordinate to ANSI-standard-Z39.50:

1 = (no longer assigned)

2 = abstract syntax definition for APDUs

3 = attribute set definitions

4 = diagnostic definitions

5 = record syntax definitions

6 =(no longer assigned)

7 = resource report format definitions

8 = access control format definitions

9 = extended services definitions

10 = user information format definitions

11 = element specification format definitions

12 = variant set definitions

13 = database schema definitions

14 = tag set definitions

15 = negotiation definitions

16 = query definitions

The following ASN.1 module establishes shorthand notation for the Z39.50 object identifier, and for the object classes. The notation is used in appendices that follow.

See ASN1.2

OID.3 Object Identifiers for Z39.50 APDUs

This standard assigns the following object identifier for the ASN.1 definition of APDUs in 4.1.

```
Z39-50-APDU          {Z39-50-APDU 1}
```

Note: the same OID for APDUs is used for Z39.50-1992, Z39.50-1995, and Z39.50-2003, to enable interworking between the versions.

OID.4 Object Identifiers Used by This Standard

Z39.50 object identifiers are either public or locally defined. Public Z39.50 object identifiers are those listed in this standard or officially registered by the Z39.50 maintenance agency (see OID.5). Locally defined Z39.50 object identifiers are registered by a registered Z39.50 Implementor (see OID.6 and OID.7).

OID.5 Object Identifiers Assigned by the Z39.50 Maintenance Agency

Additional object identifiers may be assigned by the Z39.50 Maintenance Agency (see note), of the form:

```
{Z39-50 n m}
```

where {z39-50 n} is an object class defined in OID.2, or is an additional object class defined by the maintenance agency.

Note: At the time of approval of this standard, the Z39.50 Maintenance Agency is the Library of Congress.

OID.6 Locally Registered Objects

Locally registered objects are of the form:

```
{Z39-50 n 1000 p m}
```

where {z39-50 n} is as described in OID.5, and 'p' is the OID index of a registered Z39.50 Implementor (contact the Z39.50 Maintenance Agency for procedures for registration of an implementor). A locally registered object may be published or private. Local, published objects are those whose definitions are coordinated with and published by the Z39.50 Maintenance Agency. Local, private objects are those whose definitions are not published by the Z39.50 Maintenance Agency.

OID.7 Experimental Objects

Experimental objects are of the form:

{Z39-50 n 2000 p m}

where {z39-50 n} is as described in OID.5, and 'p' is the OID index of a registered Z39.50 Implementer.

Appendix 2 ATR: Attribute Sets

(Normative)

Each attribute set defines a set of types and for each type a set of values. An attribute list (see AttributeList in the ASN.1 for APDUs, 4.1), constructed from an attribute set definition, is a list of attribute pairs. An attribute pair (AttributeElement in the ASN.1 for APDUs) consists of an attribute type and a value list (attributeValue within AttributeElement), where each value in the list is defined for that type.

When version 2 is in force, each value list is a single value and is an integer. When version 3 is in force, attributeValue (within AttributeElement) may select 'complex', allowing the value list to include multiple values (each may be integer or string) and also to specify a 'semanticAction', indicating how the server is to treat the multiple attributes.

When an attribute list contains any attribute pair where attributeValue selects 'complex', there must not be any attribute type within the attribute list for which there is more than a single attribute pair.

ATR.1 Attribute Set exp-1

This section defines the attribute-set exp-1, for use with an Explain database. The attribute set exp-1 defines a single attribute type, 'Use'. In addition, this attribute set definition ***imports*** non-Use bib-1 attributes, i.e. those of type Relation, Position, Structure, Truncation, and Completeness. The types and values defined within the bib-1 attribute set for these attributes may be used within the exp-1 attribute set, using the object identifier for this attribute set. It is recommended that a server supporting the Explain facility support the Relation attribute 'equal', Position attribute 'any' position in field', and Structure attribute 'key'.

Note: If the server supports searching based on date ranges (e.g. to limit a search to records created before or after a particular date or between two dates), the server should also support one or more of the following relation attributes: 'less than', 'less than or equal', 'greater than', and 'greater or equal'.

Table 1: Exp-1 Use Attributes

Use	Value	Use	Value	Use	Value
ExplainCategory	1	HumanStringLanguage	2	DatabaseName	3
ServerName	4	AttributeSetOID	5	RecordSyntaxOID	6
TagSetOID	7	ExtendedServiceOID	8	DateAdded	9
DateChanged	10	DateExpires	11	ElementSetName	12
Processing Context	13	ProcessingName	14	TermListName	15
SchemaOID	16	Producer	17	Supplier	18
Availability	19	Proprietary	20	UserFee	21
VariantSetOID	22	UnitSystem	23	Keyword	25
ExplainDatabase	26	ProcessingOID	27		

Notes:

(1) The search terms for Use attribute ExplainCategory are listed in table 2.

(2) The search term when the Use attribute is HumanStringLanguage is the three-character language code from ANSI/NISO Z39.53-1994 -- Codes for the Representation of Languages for Information Interchange.

(3) The search terms when the Use attribute is ProcessingContext are listed in table 3.

(4) Where the search term is an object identifier (where the name of the Use attribute ends with "OID"): for version 2, it is recommended that the term be a character string representing a sequence of integers (each represented by a character string) separated by periods. For version 3, it is recommended that the term be represented as ASN.1 type OBJECT IDENTIFIER.

(5) Use attribute Keyword is used when searching for DatabaseInfo records (i.e. in combination with an operand where Use is ExplainCategory and term is DatabaseInfo). Its use is to search in the keyword element, for terms that match one of the query terms.

(6) Use attribute ExplainDatabase is used when searching for DatabaseInfo records (i.e. in combination with an operand where Use is ExplainCategory and term is DatabaseInfo). The term should be NULL, for version 3, or otherwise ignored by the server. The Relation attribute either should be omitted or should be AlwaysMatches.

Use attributes DateAdded, DateChanged, DateExpires" correspond to elements in the CommonInfo for every Explain record, respectively, dateAdded, dateChanged, and expiryDate. These elements pertain to the Explain record itself: when it was first added to the Explain database, when it was last updated, and when it should be thrown away from any caches.

Table 2: Search terms associated with use attribute ExplainCategory

TargetInfo	TermListInfo	SortDetails
DatabaseInfo	extendedServicesInfo	Processing
SchemaInfo	AttributeDetails	CategoryList
TagSetInfo	TermListDetails	VariantSetInfo
RecordSyntaxInfo	ElementSetDetails	UnitInfo
AttributeSetInfo	RetrievalRecordDetails	

Table-3: Search terms associated with use attribute ProcessingContext

Access	Retrieval	RecordHandling
Search	RecordPresentation	

ATR.2 Attribute Set ext-1

This section defines the attribute-set ext-1, for use with an Extended Services database. Two types are defined:

Attribute Type	Value
Use	1
Permissions	2

Additional attributes (types and/or values) may be defined within a specific Extended Service definition. The attribute set id to be used to identify those attributes is the ObjectIdentifier that identifies the specific Extended Service.

Table 4: Ext-1 Use Attributes

Use	Value	Use	Value	Use	Value
UserId	1	PackageName	2	CreationDatetime	3
TaskStatus	4	PackageType	5	RetentionTime	6
ServerReference	7				

Table 5: Ext-1 Permission Attributes

Use	Value	Use	Value	Use	Value
Delete	1	Modify	2	ModifyPermissions	3
Present	4	Invoke	5	Any	6

Note: The Permission attribute is for use only when the value of the Use attribute is UserId, in which case the purpose is to search for task packages for which the specified user has the specified permission.

Appendix 3 DIAG: Z39.50 Diagnostics

Normative

When version 2 is in force, a Z39.50 diagnostic record conforms to the following format:

```
DefaultDiagFormat ::= SEQUENCE{
      diagnosticSetId   OBJECT IDENTIFIER,
      condition         INTEGER,
      addinfo           VisibleString}
```

The diagnostic record includes an integer corresponding to a condition or error, and an (object) identifier of the diagnostic set definition that lists the condition corresponding to that integer.

When version 3 is in force, a diagnostic record may assume the form above, or alternatively, may be defined as EXTERNAL, identified by an OBJECT IDENTIFIER (which identifies the diagnostic ***format***, rather than the diagnostic ***set***).

DIAG.1 General Diagnostic Set

Diagnostic set bib-1 was defined in earlier versions of this standard and is renamed in this standard as the General Diagnostic Set, with the same object identifier:

```
general-diagnostics     {Z39-50-diagnostic 1}
```

The table below is for use when DiagnosticSetId (within DefaultDiagFormat) equals the object identifier for the General Diagnostic set, in which case, Condition takes values from the "Code" column below.

AddInfo is ASN.1 type VisibleString. However, for several of the diagnostics below, AddInfo is used to express the value of a parameter that has an ASN.1 type other than VisibleString. Where Addinfo is used to express a numeric value, it should be a character string representation of that value. Where Addinfo is used to express an object identifier, it should take the form of a sequence of integers (each represented by a character string) separated by periods.

The General Diagnostic Set includes the diagnostics listed below, which includes all of the general diagnostics registered at the time of approval of this standard. For a complete list, see http://lcweb.loc.gov/z3950/agency/defns/diag.html

Code	Meaning	Addinfo
1	permanent system error	(unspecified)
2	temporary system error	(unspecified)
3	unsupported search	(unspecified)

Code	Meaning	Addinfo
4	Terms only exclusion (stop) words	(unspecified)
5	Too many argument words	(unspecified)
6	Too many boolean operators	(unspecified)
7	Too many truncated words	(unspecified)
8	Too many incomplete subfields	(unspecified)
9	Truncated words too short	(unspecified)
10	Invalid format for record number (search term)	(unspecified)
11	Too many characters in search statement	(unspecified)
12	Too many records retrieved	(unspecified)
13	Present request out-of-range	(unspecified)
14	System error in presenting records	(unspecified)
15	Record not authorized to be sent intersystem	(unspecified)
16	Record exceeds Preferred-message-size	(unspecified)
17	Record exceeds Exceptional-record-size	(unspecified)
18	Result set not supported as a search term	(unspecified)
19	Only single result set as search term supported	(unspecified)
20	Only ANDing of a single result set as search term	(unspecified)
21	Result set exists and replace indicator off	(unspecified)
22	Result set naming not supported	(unspecified)
23	Specified combination of databases not supported	(unspecified)
24	Element set names not supported	(unspecified)
25	Specified element set name not valid for specified database	(unspecified)
26	Only generic form of element set name supported	(unspecified)
27	Result set no longer exists - unilaterally deleted by server	(unspecified)
28	Result set is in use	(unspecified)
29	One of the specified databases is locked	(unspecified)
30	Specified result set does not exist	(unspecified)
31	Resources exhausted - no results available	(unspecified)
32	Resources exhausted - unpredictable partial results available	(unspecified)
33	Resources exhausted - valid subset of results available	(unspecified)
100	(unspecified) error	(unspecified)
101	Access-control failure	(unspecified)
102	Challenge required, could not be issued - operation terminated	(unspecified)
103	Challenge required, could not be issued - record not included	(unspecified)

Code	Meaning	Addinfo
104	Challenge failed - record not included	(unspecified)
105	Terminated at client request	(unspecified)
106	No abstract syntaxes agreed to for this record	(unspecified)
107	Query type not supported	(unspecified)
108	Malformed query	(unspecified)
109	Database unavailable	database name
110	Operator unsupported	operator
111	Too many databases specified	maximum
112	Too many result sets created	maximum
113	Unsupported attribute type	type
114	Unsupported Use attribute	value
115	Unsupported term value for Use attribute	term
116	Use attribute required but not supplied	(unspecified)
117	Unsupported Relation attribute	value
118	Unsupported Structure attribute	value
119	Unsupported Position attribute	value
120	Unsupported Truncation attribute	value
121	Unsupported Attribute Set	oid
122	Unsupported Completeness attribute	value
123	Unsupported attribute combination	(unspecified)
124	Unsupported coded value for term	value
125	Malformed search term	(unspecified)
126	Illegal term value for attribute	term
127	Unparsable format for un-normalized value	value
128	Illegal result set name	name
129	Proximity search of sets not supported	(unspecified)
130	Illegal result set in proximity search	result set name
131	Unsupported proximity relation	value
132	Unsupported proximity unit code	value
201	Proximity not supported with this attribute combination attribute	list
202	Unsupported distance for proximity	distance
203	Ordered flag not supported for proximity	(unspecified)
205	Only zero step size supported for Scan	(unspecified)
206	Specified step size not supported for Scan step	size
207	Cannot sort according to sequence	sequence

Code	Meaning	Addinfo
208	No result set name supplied on Sort	(unspecified)
209	Generic sort not supported (database-specific sort only supported)	(unspecified)
210	Database specific sort not supported	(unspecified)
211	Too many sort keys	number
212	Duplicate sort keys	key
213	Unsupported missing data action	value
214	Illegal sort relation	relation
215	Illegal case value	value
216	Illegal missing data action	value
217	Segmentation: Cannot guarantee records will fit in specified segments	(unspecified)
218	ES: Package name already in use	name
219	ES: no such package, on modify/delete	name
220	ES: quota exceeded	(unspecified)
221	ES: extended service type not supported	type
222	ES: permission denied on ES - id not authorized	(unspecified)
223	ES: permission denied on ES - cannot modify or delete	(unspecified)
224	ES: immediate execution failed	(unspecified)
225	ES: immediate execution not supported for this service	(unspecified)
226	ES: immediate execution not supported for these parameters	(unspecified)
227	No data available in requested record syntax	(unspecified)
228	Scan: malformed scan	(unspecified)
229	Term type not supported	type
230	Sort: too many input results	max
231	Sort: incompatible record formats	(unspecified)
232	Scan: term list not supported	alternative term list
233	Scan: unsupported value of position-in-response	value
234	Too many index terms processed	number of terms
235	Database does not exist	database name
236	Access to specified database denied	database name
237	Sort: illegal sort	(unspecified)
238	Record not available in requested syntax	alternative suggested syntax(es)
239	Record syntax not supported	syntax
240	Scan: Resources exhausted looking for satisfying terms	(unspecified)

Code	Meaning	Addinfo
241	Scan: Beginning or end of term list	(unspecified)
242	Segmentation: max-segment-size too small to segment record	smallest acceptable size
243	Present: additional-ranges parameter not supported	(unspecified)
244	Present: comp-spec parameter not supported	(unspecified)
245	Type-1 query: restriction ('resultAttr') operand not supported	(unspecified)
246	Type-1 query: 'complex' attributeValue not supported	(unspecified)
247	Type-1 query: 'attributeSet' as part of AttributeElement not supported	(unspecified)
1001	Malformed APDU.	
1002	ES: EXTERNAL form of Item Order request not supported.	
1003	ES: Result set item form of Item Order request not supported.	
1004	ES: Extended services not supported unless access control is in effect.	
1005	Response records in Search response not supported.	
1006	Response records in Search response not possible for specified database (or database combination). *See note 1.*	
1007	No Explain server. *See note 2.*	pointers to servers that have a surrogate Explain database for this server.
1008	ES: missing mandatory parameter for specified function	parameter
1009	ES: Item Order, unsupported OID in itemRequest.	OID
1010	Init/AC: Bad Userid	
1011	Init/AC: Bad Userid and/or Password	
1012	Init/AC: No searches remaining (pre-purchased searches exhausted)	
1013	Init/AC: Incorrect interface type (specified id valid only when used with a particular access method or client)	
1014	Init/AC: Authentication System error	
1015	Init/AC: Maximum number of simultaneous sessions for Userid	
1016	Init/AC: Blocked network address	
1017	Init/AC: No databases available for specified userId	
1018	Init/AC: System temporarily out of resources	
1019	Init/AC: System not available due to maintenance	when it's expected back up
1020	Init/AC: System temporarily unavailable	when it's expected back up
1021	Init/AC: Account has expired	
1022	Init/AC: Password has expired so a new one must be	

Code	Meaning	Addinfo
	supplied	
1023	Init/AC: Password has been changed by an administrator so a new one must be supplied	
1024	Unsupported Attribute. *See note 3*.	an unstructured string indicating the object identifier of the attribute set id, the numeric value of the attribute type, and the numeric value of the attribute.
1025	Service not supported for this database	
1026	Record cannot be opened because it is locked	
1027	SQL error	
1028	Record deleted	
1029	Scan: too many terms requested.	Addinfo: max terms supported
1030 - 1039	*currently unnassigned*	
1040	ES: Invalid function	function
1041	ES: Error in retention time	(unspecified)
1042	ES: Permissions data not understood	permissions
1043	ES: Invalid OID for task specific parameters	oid
1044	ES: Invalid action	action
1045	ES: Unknown schema	schema
1046	ES: Too many records in package	maximum number allowed
1047	ES: Invalid wait action	wait action
1048	ES: Cannot create task package -- exceeds maximum permissable size *(see note 4)*	maximum task package size
1049	ES: Cannot return task package -- exceeds maximum permissable size for ES response *(see note 5)*	maximum task package size for ES response
1050	ES: Extended services request too large *(see note 6)*	maximum size of extended services request
1051	Scan: Attribute set id required -- not supplied	
1052	ES: Cannot process task package record -- exceeds maximum permissible record size for ES *(see note 7)*	maximum record size for ES
1053	ES: Cannot return task package record -- exceeds maximum permissible record size for ES response *(see note 8)*	maximum record size for ES response
1054	Init: Required negotiation record not included	oid(s) of required negotiation record(s)
1055	Init: negotiation option required	
1056	Attribute not supported for database	attribute (oid, type, and

Code	Meaning	Addinfo
		value), and database name
1057	ES: Unsupported value of task package parameter *(See Note 9)*	parameter and value
1058	Duplicate Detection: Cannot dedup on requested record portion	
1059	Duplicate Detection: Requested detection criterion not supported	detection criterion
1060	Duplicate Detection: Requested level of match not supported	
1061	Duplicate Detection: Requested regular expression not supported	
1062	Duplicate Detection: Cannot do clustering	
1063	Duplicate Detection: Retention criterion not supported	retention criterion
1064	Duplicate Detection: Requested number (or percentage) of entries for retention too large	
1065	Duplicate Detection: Requested sort criterion not supported	sort criterion
1066	CompSpec: Unknown schema, or schema not supported.	
1067	Encapsulation: Encapsulated sequence of APDUs not supported	specific unsupported sequence
1068	Encapsulation: Base operation (and encapsulated APDUs) not executed based on pre-screening analysis.	
1069	No syntaxes available for this request. *See note 10.*	
1070	user not authorized to receive record(s) in requested syntax	
1071	preferredRecordSyntax not supplied	
1072	Query term includes characters that do not translate into the target character set.	Characters that do not translate

Notes:

1. Diagnostic 1006 is intended for the case of an intermediary providing access to multiple servers, some of which may support piggybacking and some which do not. This diagnostic is for the intermediary to use in case the particular end server doesn't support piggybacking (as opposed to diagnostic 1005, which, in the case of an intermediary, would imply that the intermediary does not support piggybacking).
2. Diagnostic 1007 is intended for use as Search diagnostic, when the client attempts to search the Explain database, and although the server doesn't support Explain, it is smart enough to recognize that this is what the client is attempting, and is able to recommend a surrogate server.
3. Diagnostic 1024 was included because existing attribute-related diagnostics are specific to the bib-1 attribute set. For example a query might contain the operand "Parent-collection = 'federal theater Project'" where 'parent-collection' is a Use attribute from the digital collection attribute set. If the server does not support that attribute, it may return this diagnostic and attach the string (in the addinfo field) "attribute set:

1.2.840.10003.3.7; type: 1; value: 4".

4. Diagnostic 1048 applies when the client sends an ES request (update) containing one or more records, and the resultant task package is too large for the server. (The client must then find a way to reduce the size of the task package or use some other means of sending the update request.)
5. Diagnostic 1049 applies when the client sends an ES request (update) with waitAction = 'wait', the task package is created, but it is too large for the server to return in the response. The client can then use Search and Present on the task package database to retrieve the task package, perhaps specifying an element set that will reduce record sizes, or using segmentation. (When using Search and Present on the task package the diagnostics 16, "Record exceeds preferred message size", and 17, "Record exceeds preferred message size" apply.)
6. Diagnostic 1050 applies when the client sends an ES request (Update) containing one or more records, and the entire message is too large for the server. The client must then find a way to reduce the message size or use some other means of sending the update request.
7. Diagnostic 1052 applies when the client sends an ES request (Update) containing one or more records; the message is within message size limits and the task package is within task package limits, but one of the records is too large. Diagnostic 1052 would be substituted as a surrogate diagnostic within the returned task package. The offending record would have no effect on the processing of other records that may have been included in the request, and in fact these other records may be returned in the ES response in the case of waitAction = 'wait'. The client should recreate the record within the size limit and submit another ES request with that record.
8. Diagnostic 1053 applies when the client sends an ES request (Update) with waitAction = 'wait', containing one or more records; the message is within message size limits and the task package is within task package limits, but one of the records is too large to fit in task package for return in the ES response. Diagnostic 1053 would be substituted as a surrogate diagnostic within the returned task package in the ES response. The record may in fact have been updated but it could not be included in the returned task package. The client can then use Search and Present on either the database itself or on the task package database to retrieve the record, if necessary specifying an element set that will reduce the record size, or using segmentation. (When using Search and Present on the task package the diagnostics 16, "Record exceeds preferred message size", and 17, "Record exceeds preferred message size" apply.)
9. Diagnostic 1057 applies for example when a client sends a PeriodicQuerySchedule with a period of "fortnight", but the server only supports period in seconds and cannot convert to fortnight; or the client send ExportInvocation where the value of 'records' is' ranges', but the server only support a value of 'all'.
10. Diagnostic 1069 is used when Present status is 'failure'. This is a non-surrogate diagnostic applying to the Present operation (or Retrieval phase of search operation) at large rather than to a single record.

DIAG.2 General Diagnostic Container

The General Diagnostic Container is assigned the following object identifier:

```
generalDiagnosticContainer    {Z39-50-diagnostic 4}
```

This format provides a service-independent and status-independent mechanism for a server to provide operation-level diagnostics. It provides a well-known object identifier that a client will recognize to mean "diagnostics inside", even though the client may not recognize any of the object identifiers (and consequently any of the diagnostics) contained within.

For example, suppose one or more APDUs are encapsulated within a Search (see 4.3), the Search executes successfully, but the server choose not to execute the encapsulated APDUs. The server is expected to include in the response to the Search APDU an appropriate diagnostic, for example: "specified sequence of APDUs is not supported". The diagnostic mechanism defined for Search does not readily accommodate a diagnostic of this nature, particularly where the Search status is 'success'.

This format is intended for use within otherInfo (or by simulation of otherInfo using the userInformationField; see USR.2: Use of Init Parameters for User Information). It may also be used to support diagnostic information in an Init response; see DIAG.3 “Returning diagnostics in an InitResponse”. It is not intended to supercede diagnostic mechanisms already defined for individual services.

See ASN1.3

DIAG.3 Returning Diagnostics in an InitResponse

A server may supply one or more diagnostics in an InitResponse APDU, within UserInfo-1 (see USR.3), within userInformationField.

When the server wishes to return one or more diagnostics, it may do so using the General Diagnostic Container (see DIAG.2) which may be included within UserInfo-1, which is the EXTERNAL to be referenced by userInformationField.

See related specification “Use of Init Parameters for User Information.” USR.2.

For examples of diagnostics meaningful in an Init response see General Diagnostic Set (DIAG.1) diagnostics 1010 through 1013.

Appendix 4 REC: Record Syntaxes

Normative

This appendix lists Z39.50 record syntaxes, but only those that are generic (SUTRS and GRS-1) or directly related to the protocol (Explain and Extended Services Task Package). In Z39.50-1995 a more complete list of record syntaxes was included. These can be found at http://lcweb.loc.gov/z3950/agency/defns/oids.html#5

REC.1 Explain Record Syntax

See ASN1.4

REC.2 Simple Unstructured Text Record Syntax, SUTRS

The Simple Unstructured Text Record Syntax (SUTRS) is intended to be used as a record syntax in a Search or Present response, to present textual data so that the client may display it with little or no analysis and manipulation. A SUTRS record is unstructured; the text of a SUTRS record might represent individual elements, but the elements are not explicitly identified by the syntax. The convention prescribed by the SUTRS definition is to use a delimiter within the text to indicate the end of a line of text. The prescribed line terminator is ASCII LF (X'0A'). Thus a SUTRS record consists simply of a string of textual data.

This definition recommends that the maximum line length be 72 characters unless an alternative maximum is requested, for example via a variantRequest. This is not an absolute maximum, but it is recommended that servers make a best effort to limit lines to this length.

See ASN1.5

Note: A SUTRS record valid when version 3 is in force might not be valid for version 2. When SUTRS is used in version 2, even though it carries the GeneralString tag, it may only include characters from the VisibleString repertoire.

REC.3 Generic Record Syntax 1

See ASN1.6

REC3.1 Embedding MARC in a GRS-1 Record

This section describes how to embed a MARC record within a GRS-1 record. This pertains to the case where GRS-1 is the record syntax; it does not address nor preclude the case where the record syntax itself is one of the MARC formats, e.g. MARC21.

When a MARC record is to be embedded inside a GRS-1 record, the MARC record should be encoded as EXTERNAL, via the 'ext' CHOICE for ElementData. The associated Object Identifier (for the EXTERNAL) will be the Object Identifier assigned to the particular MARC format (e.g. 1.2.840.10003.5.10 for MARC21).

The reason for this specification is that there is potentially more than one way to embed MARC within GRS, for example, the MARC record could be encoded as OCTET STRING, where an applied variant is supplied to identify the MARC format.

According to this specification, the EXTERNAL form, rather than OCTET STRING, should be used, regardless of whether or not an applied variant is supplied (it may, but need not, be supplied).

REC.4 Record Syntax For Extended Services Task Package

See ASN1.7

Appendix 5 RSC: Resource Report Formats

Normative

This appendix provides the definition of the resource report formats resource-2, whose object identifier is:

```
resource-2  {Z39-50-resourceReport 2}
```

In earlier versions of this standard the definition of resource-1 was also provided. Its object identifier is:

```
resource-1  {Z39-50-resourceReport 1}
```

resource-1, defined in 1992, provides 16 categories of resources with no provision for extensibility. Resource-2, defined in 1995, inherits the original 16 categories, with provision for extensibility. Thus resource-2 is a compatible superset of resource-1. The resource-1 definition is therefore not provided. However, It is recommended that a client be prepared to recognize the resource-1object identifier.

Resource Report Format Resource-2

See ASN1.8

Appendix 6 ACC: Access Control Formats

Normative

This appendix provides definitions for the following access control formats:

```
prompt-1    {Z39-50-accessControl  1}

des-1       {Z39-50-accessControl  2}

krb-1       {Z39-50-accessControl  3}
```

Access control formats are defined for use within the parameters securityChallenge and securityChallengeResponse of the AccessControlRequest and AccessControlResponse APDUs, and idAuthentication of the InitializeRequest APDU.

See ASN1.9

Appendix 7 EXT: Extended Services Defined by this Standard

Normative

This standard defines and registers the Extended Services listed below, and assigns the following object identifiers:

PersistentResultSet	**{Z39-50-extendedServices 1}**
PersistentQuery	**{Z39-50-extendedServices 2}**
PeriodicQuery Schedule	**{Z39-50-extendedServices 3}**
ItemOrder	**{Z39-50-extendedServices 4}**
DatabaseUpdate	**{Z39-50-extendedServices 5}**
ExportSpecification	**{Z39-50-extendedServices 6}**
ExportInvocation	**{Z39-50-extendedServices 7}**

EXT.1 provides service descriptions, and EXT.2 provides ASN.1 definitions.

EXT.1 Service Definitions

An Extended Service is carried out by an Extended Service (ES) task, which is invoked by an ES operation. The ES Service is described in 3.2.9.1.

Execution of the ES Operation results in the creation of a task package, represented by a database record in the ES database. A task package contains parameters, some of which are common to all task packages regardless of package type, and others that are specific to the task type. Among the common parameters, some are supplied by the client as parameters in the ES request, and others are supplied by the server.

Table-1 : Parameters Common to all Extended Services

Common Task Package Parameter	Client supplied	Server supplied	Reference
packageType	m		3.2.9.1.2
packageName	o		3.2.9.1.3
userId	o		3.2.9.1.4
retentionTime	o	o	3.2.9.1.5
permissionsList	o	o	3.2.9.1.6
description	o		3.2.9.1.7
serverReference		o	3.2.9.1.8
creationDateTime		o	3.2.9.1.9
taskStatus		m	3.2.9.1.10
packageDiagnostics		o	3.2.9.1.11

The specific parameters are derived from the ES request parameter Task-specific-parameters. Table 1 provides a summary of common parameters. Their descriptions are included in 3.2.9.1. For parameters listed as both "client supplied" and "server supplied," when both client and server supply a value, the server supplied value overrides the client supplied value.

EXT.1.1 Persistent Result Set Extended Service

The Persistent Result Set Extended Service allows a client to request that the server create a persistent result from a transient result set belonging to the current Z-association. The Persistent Result Set task has no effect on the transient result set; it remains available for use by the Z-association. The persistent result set is saved for later use, during the current or a different Z-association. It may subsequently be deleted, by deletion of the task package.

Note: The client may thus cause deletion of the persistent result set, by deleting the task package, if the client user has "delete" permission for that package.

A Present (using the ResultSetName element specification), against the Persistent Result Set Parameter Package returns a Parameter Package that contains a server-supplied transient result set name, which may be used during the same Z-association wherever a result set name may be used (e.g. within a query, or in Present, Sort, or Delete request).

This definition does not specify how persistent result sets are implemented (only how they are viewed by the client). When a transient result set is "saved", presumably it will be restored subsequently into another transient result set (either in the same session or a different session). So suppose for example transient result set A is saved (i.e. a Persistent Result Set Task Package representing that result set is created) and subsequently restored into transient result set B (i.e. the task package is “Present”ed and the server supplies the result set name B, meaning, implicitly, that the client may use B to reference the restored result set). Suppose (for illustration) that it is restored within the same session during which it was earlier saved; in that case, the result sets A and B should be identical (if a record has changed according to result set A, then it has also changed according to result set B), if the server has implemented result sets according to the abstract model, i.e. via pointers. But there is no such requirement that the server do so. The server might instead "save" the result set by actually copying the records, in which case the two-result set may not be identical. The standard does not specify how the server is to actually save and/or store the database records, or how similar to the original records the restored records must be.

Note that management aspects of persistent result sets are not included in this definition. When a result set is saved, a task package is created in the ES database that represents the result set. However, the result set itself (i.e. the content) is not part of the ES database, that is, result set records from the saved result set are not directly retrievable. They may be retrieved only as described above, that is, by restoring the saved result set to a transient result set and then presenting from that transient result set. So a persistent result set cannot be directly modified. A client can save a result set and the client (or a different client) may subsequently restore it, modify it, and save it again. But the management aspects of this are not within the scope of this definition, except to the extent that the Extended Services facility does provide "permissions" capability for use by an administrator. A persistent result set may be directly deleted (a client can simply delete the task package, using the delete function on an extended services request, which in effect deletes the persistent result set).

The parameters of the Persistent Result Set Extended Service are those shown in Table 1 as well as those in Table 2.

Table 2: Specific Parameters for Persistent Result Set

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
clientSuppliedResultSet	ia		
replaceOrAppend	ia		
serverSuppliedResultSet		ia	ia
numberOfRecords		o	o

clientSuppliedResultSet — The client supplies the name of a transient result set belonging to the Z-association. If function is 'create', the server is to create a persistent result set from this transient result set. If function is 'modify' the server is to either replace an existing persistent result set (corresponding to the specified package name) with this result set, or append this result set to an existing persistent result set. This parameter is mandatory when the value of the request parameter function is 'create' or 'modify', and is not included when function is 'delete'.

replaceOrAppend — This parameter occurs when function is 'modify' (and is valid only when the client user has "modify-contents" permission). Its value is 'replace' or 'append' meaning that the specified result set is, respectively, to replace, or to be appended to, the existing persistent result set.

serverSuppliedResultSet — When the client retrieves the task package, the server supplies the name of a transient result set, which then belongs to the Z-association. The result set is a copy of the persistent result set represented by the package. The server includes this parameter only when the task package is retrieved (i.e. not on an ES response) and does not include the parameter if the element set name on the Present request indicates that the parameter is not to be included.

numberOfRecords — The server indicates the total number of records in the persistent result set.

EXT.1.2 Persistent Query Extended Service

The Persistent Query Extended Service allows a client to request that the server save a Z39.50 Query for later reference, during the same or a subsequent Z-association.

The parameters of the Persistent Query Extended Service are those shown in Table 1 as well as those in Table 3.

Table 3: Specific Parameters for Persistent Query

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
querySpec	m		
actualQuery		m	m
databaseNames	o		o
additionalSearch Information	o		o

querySpec and ActualQuery — The client supplies either the query to be saved or the name of another persistent query to be copied into this package. The server supplies the actualQuery: if the client has supplied a query, the server uses that query; if the client supplies a task package name, the server copies the corresponding query.

databaseNames — The client optionally supplies a list of databases.

additionalSearchInformation — See 3.2.2.1.12.

EXT.1.3 Periodic Query Schedule Extended Service

The Periodic Query Schedule Extended Service allows a client to request that the server establish a Periodic Query Schedule. The client can also request that the schedule be "activated," either as part of the initial request to create the schedule, or as part of a subsequent request to modify the schedule. The parameters of the Periodic Query Schedule Extended Service are those shown in Table 1 as well as those in Table 4.

Table 4: Specific Parameters for Periodic Query Schedule

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
activeFlag	m		m
querySpec	m		
actualQuery		m	m
databaseNames	ia	m	m
additionalSearchInfo	o		o
period	m	o	m
expiration	o	o	o
resultSetPackageName	o	ia	ia
resultSetDisposition	ia		ia
alertDestination	o		o
exportParameters	o		o

lastQueryTime		m	m
lastResultNumber		m	m
numberSinceModify		o	o

activeFlag	On a Create request, if this flag is set, the Periodic Query Schedule is to be activated immediately upon receipt and validation of its parameters; otherwise the schedule is to be Created but not activated. On a Modify request (which may contain as little as just the ActiveFlag), the client may activate or deactivate the schedule. In the parameter package, this parameter indicates whether the schedule is active.
querySpec and ActualQuery	The client supplies either a query or the name of a Persistent Query Package. (If the client supplies a query, or if the specified query package does not include a list of databases, then the databaseNames parameter is required.) The server supplies the actualQuery: if the client has supplied a query, the server uses that query; if the client supplies a task package name, the server copies the corresponding query.
databaseNames	The client may supply a list of databases; the list is required if the client supplied a query rather than a query package name for querySpec, or if the specified query package does not include a list of databases.
additionalSearchInfo	The client may use this parameter to supply additional search information, not specified by this definition.
period	The time period between invocations of the query. The server may override the period specified by the client. Period may be a number of days, a frequency (e.g. daily, business daily, weekly, monthly), or 'continuous', meaning the search is to be run continuously (or at the server's discretion).
expiration	The client may optionally supply a time/date for the server to discontinue execution of this Periodic Query. If the client does not supply a value, the client is proposing "no expiration." The server may override the client supplied value. If the client supplies a value and the server does not support expiration, the server should reject the ES request.
resultSetPackageName	The client may optionally supply the name of an existing Persistent Result Set package. If the client omits this parameter, the server is to create a persistent result set, unless the parameter exportParameters is included.
resultSetDisposition	This parameter takes on the value 'createNew', 'replace',

	or 'append', indicating respectively whether the server is to create a new result set each time the query is invoked, replace the contents of the existing result set, or append any new results to the end of the result set. The value 'createNew' should be used only if the client and server have an agreement about naming conventions for the resulting package. If the value of the parameter Period is 'continuous' it is recommended that the value of this parameter be 'append'. The value 'append' allows the server to continually extend the result set by appending new records.
alertDestination	The client may optionally supply a destination address for Alerts triggered by receipt of new Periodic Query results (e.g. fax number, email address, pager number).
exportParameters	The client may optionally supply the name, or actual contents, of an Export Parameter Package to be used with this Periodic Query. It is included only if the client wants newly posted results to be exported; if so, new results may also be posted to ResultSetName if also specified.
lastQueryTime	The server indicates the last time this Periodic Query was invoked.
lastResultNumber	The server indicates the number of new records obtained last time query was invoked.
numberSinceModify	The server indicates the total number of records obtained via invocation of the Query since the last time this Periodic Query Package was modified.

EXT 1.4 Item Order Extended Service

The Item Order Extended Service allows a client to submit an item order request to the server. The parameters of the Item Order Extended Service are those shown in Table-1 as well as those in Table5.

Table-5: Specific Parameters for Item Order

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
requestedItem	m		
item Request		ia	ia
supplemental Description	o		o
contactInformation	o		o
additionalBillingInfo	o		o
statusOrErrorReport		m	m
auxiliaryStatus		o	o

requestedItem — The client identifies the requested item, either by:

(a) A request whose format is defined externally, and which may be an Interlibrary Loan Request APDU of ISO 10161; or

(b) A result set item (name of a transient result set belonging to the current Z-association and an ordinal number of an entry within that result); or

(c) Both.

itemRequest — If requestedItem is (a) (e.g. an interlibrary loan request), the server copies it into the task package (although the server might first modify the request). If requestedItem is (b), the server may construct a corresponding item request; if it does not, then the requested item will not be identified within the task package.

supplementalDescription — The client may supply additional descriptive information pertaining to the requested item, as a supplement to requestedItem.

contactInformation — The client may optionally supply a name, phone number, and electronic mail address of a contact-person.

additionalBillingInfo — The client may optionally indicate payment method, credit card information, customer reference, and customer purchase order number.

statusOrErrorReport — The server supplies a status or error report. The definition of the report is external to this standard, and may be based on the StatusOrErrorReport APDU of the ILL protocol.

auxiliaryStatus — The server may provide an auxiliary status as a supplement to the status information which might be provided by the statusOrErrorReport.

EXT 1.5 Database Update Extended Service

The database Update Extended Service allows a client to request that the server update a database: insert new records, replace or delete existing records, or update elements within records.

Note: This service definition does not address concurrency; if multiple users try to update the same record, it may be that only the first request served by the server will update the intended data, and the remaining requests may update a record whose content has changed.

The parameters of the databaseUpdate Extended Service are those shown in Table-1 as well as those in Table-6.

Table-6: Specific Parameters for DatabaseUpdate

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
action	m		m
databaseName	m		m
schema	o		o
suppliedRecords	m		
recordIds	o		
supplementalIds	o		
correlationInfo	o		o
elementSetName	o		o
updateStatus		ia	ia
globalDiagnostics		ia	ia
taskPackageRecords		ia	ia
recordStatuses		ia	ia

action — The client indicates recordInsert, recordReplace, recordDelete, or elementUpdate.

databaseName — The client indicates the database to which the action pertains.

schema — The client indicates the database schema that applies for this update.

Note: The action, databaseName, and schema are specified once, and apply to all of the included records. It is not possible to specify different values for different records in the same task package. For separate Actions (etc), use separate task packages.

suppliedRecords	The client supplies one or more records. (Along with each the client may also supply a recordId, supplemental identification, and correlation information; see following three parameters.) For recordInsert or recordReplace, the client supplies whole records. For recordReplace or recordDelete, each supplied record (or corresponding supplemental identification or recordId) must include sufficient information for the server to identify the database record. For recordDelete, sufficient identifying information should be supplied for each record, but the whole record need not necessarily be supplied. For elementUpdate, the elements within a supplied record are to replace the corresponding elements within the database record, and the remainder of the database record is unaffected. Records must be supplied in a manner that allows the corresponding elements in the database record to be identified (e.g. via tags defined by the schema). For any element within a supplied record, if there is no corresponding element within the database record, if there is more than a single occurrence of the corresponding element, or if the element is not sufficiently identified, the update will not be performed for that record. (For elementUpdate, supplementalId may be used for identification of the record, but not for identification of elements.)
recordIds	Corresponding to each supplied record the client may optionally supply a record Id.
supplementalIds	Corresponding to each supplied record the client may supply supplemental identification to allow the server to identify the database record, or to identify the correct version of the database record. This may be a timestamp, a version number, or may take some other form, for example, a previous version of the record.
CorrelationInfo	Corresponding to each supplied record, the client may include one or both of the following: 1. A correlationNote – information pertaining to the update of the record, for example, why it was updated, who updated it, the nature of the update, etc.; 2. A correlationIdentifier -- An identifier for the record CorrelationInfo provides a means for the client or user to insert information into an Update ES task package, corresponding to a particular record included within the task package. This information allows a client or user when subsequently retrieving the package (possibly a

different client or user than that which originally submitted the ES Update request), to discover this information for a given record.

CorrelationInfo is intended to be opaque to the server, who should not process it or change it.

In case 1 above it would take the form of the note, a human-readable (i.e. non-processable) string. In this case, the user who originally inserted the information may have anticipated that a different user might subsequently retrieve the task package.

In case 2 above, it would take the form of an identifier. It is not intended necessarily to be a unique or unambiguous identifier of the record; it is intended to uniquely and unambiguously identify the record only within the task package. (Thus if the same record occurs in two different task packages it may have different correlation ids; conversely, a correlation id used to identify a record within one task package may be re-used to identify a different record in a different task package.)

Thus a client may assign a unique id for each record in an Update ES request and maintain a table for each Update task package that correlates each id assigned within the task package to the record to which it is assigned, so that when the task package is retrieved, for each instance of TaskPackageRecordStructure (each such instance corresponds to one record that was in the client Update ES request) the client will be able to determine which record that instance pertains to (correlationInfo is included within TaskPackageRecordStructure).

The reason for the correlation identifier is that the actual record might not be included within taskPackage RecordStructure, or if it is, the record itself might not have an unambiguous identifier. Thus its scope is much narrower than an all-purpose identifier (it is therefore defined as INTEGER, because integer representation is sufficient for its purpose).

ElementSetName

The client indicates an element set name indicating which elements of the updated records are to be included in the task package. If omitted, updated records are not to be included in the task package.

updateStatus

This parameter occurs in the task package only when taskStatus is 'complete' or 'aborted'. It is one of the following:

Update Status	Meaning
Success	Update performed successfully.
Partial	Update failed for one or more records.
Failure	Server rejected execution of the task (one or more non-surrogate diagnostics should be supplied in parameter globalDiagnostics).

See also EXT 1.5.1.

globalDiagnostics — One or more non-surrogate diagnostics, supplied if updateStatus is Failure.

taskPackageRecords — When taskStatus is 'complete': the task package includes a structure for each supplied record. The structure may include part or all of the updated record (depending on 'elementSetName') or a surrogate diagnostic (when recordStatus, below, is 'failure'), as well as correlationInfo and record status (see next parameter).

When taskStatus is 'pending' or 'active': the task package includes the above for each record for which update action is complete. For those records for which action is not complete, the structure includes the correlationInfo and status.

recordStatuses — Corresponding to each task package record, the task package includes a record status:

Record Status	Meaning
success	The record was updated successfully.
queued	The record is queued for update, or the update is in process (this status may be used in lieu of inProcess, when the server does not wish to distinguish between these two statuses).
inProcess	The update for this record is in process.
failure	The update for this record failed. A surrogate diagnostic should be supplied in lieu of the record (within the structure corresponding to the record, within the parameter taskPackageRecords).

See also EXT 1.5.1.

EXT 1.5.1 Summary of Status Parameters for Update ES

As noted in 3.2.9.5, OperationStatus and taskStatus both apply to Extended Services in general. updateStatus and recordStatus are specific to Update. These distinguish the update task at large from update action that applies to each individual record.

Update status

UpdateStatus is not set until the task is complete, or rejected. Its values are:

- 'success'
- 'partial'
- 'failure'

recordStatus

In addition, there is a "record status" for each record. Values are:

- 'success'
- 'failure'
- 'queued'
- 'inProcess'

For each record, when the task package is initially set up, this status is set to 'queued'; subsequently it is set to either 'success' or 'failure' when update action is complete for the record. In the interim, the status may change from 'queued' to 'inProcess' (the server may skip either of these statuses, ‘queued’ and ‘inProcess’). So at any time after the task begins, any record may have status of 'queued', 'inProcess', 'success' or 'failure'. The status may change from ‘queued’ to ‘inProcess’ to ‘success’ or ‘failure’, but once the status becomes 'success' or 'failure', it should not subsequently change. One usage of this status is to enable a client to monitor the progress of the task, on a record-by-record basis.

updateStatus is not set until recordStatus is set for each individual record; it will be 'success' if recordStatus is 'success' for every record, and will be 'partial' if recordStatus is 'success' for some but not all records. So 'partial' (for updateStatus), doesn't mean "partially done" it means "task done, but only some of the records were successfully updated".

When taskStatus is 'pending' all the record statuses are 'queued'. When taskStatus is 'active' some of the record statuses may be other than 'queued'. When taskStatus is 'complete' or 'aborted' none of the record statuses should be 'queued'.

EXT 1.6 Export Specification Extended Service

The Export Specification Extended Service allows a client to request that the server establish an export specification. Once established, the export specification may be subsequently invoked (repeatedly) by an Export Invocation Extended Services task; in fact, multiple invocations may be running simultaneously.

An Export Specification includes a delivery destination as well as other information that controls the delivery of a unit of information (one or more result set records). The destination might be a printer or some other device. The delivery mechanism could include fax, electronic mail, file transfer, or a server-supported print device. The parameters of the Export Specification Extended Service are those shown in Table-1 as well as those in Table-7.

Table-7: Specific Parameters for Export Specification

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter

composition	m		m
exportDestination	m		m

composition — This parameter consists of a record syntax, element specification, variants, etc. of the records to be Exported.

exportDestination — The client indicates an address or other destination instruction (e.g. e-mail address, printer address, fax number).

EXT 1.7 Export Invocation Extended Service

The Export Invocation Extended Service allows a client to invoke an export specification. The client may supply an export specification, or the name of an export specification that has been established by an Export Specification task as described in EXT 1.6. The parameters of the Export Invocation Extended Service are those shown in Table-1 as well as those in Table-8.

Table-8: Specific Parameters for Export Invocation

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
exportSpecification	m		
resultSetId	m		
resultSetRecords	m		
numberOfCopies	m		
estimatedQuantity		o	o
quantitySoFar		o	o
estimatedCost		o	o
costSoFar		o	o

exportSpecification — The client supplies the packageName, or actual contents, of an export specification.

resultSetId — The client supplies the name of a transient result set, from which records are selected for export.

resultSetRecords — The client indicates which records are to be exported. This parameter may specify that all records in the result set are to be exported, or it may specify a set of ranges of result set records, in which case the last range may indicate that all records beginning with a specific record are to be exported.

numberOfCopies — The client indicates the number of copies requested.

estimatedQuantity and quantitySoFar — The server optionally indicates the number of pages,

	message packets, etc., estimated in the information to be exported, and the actual amount exported so far.
estimatedCost and costSoFar	The server optionally supplies an estimate of the cost to export this information, and the cost accrued so far.

EXT.2 ASN.1 Definitions of Extended Services Parameter Package

Each definition below corresponds to an individual extended service. Each structure occurs within an ES request or as a task package. Correspondingly, each is defined as a CHOICE of 'esRequest' and 'taskPackage'. If the structure occurs within an ES request, it occurs as the parameter taskSpecificParameters. The structure may occur as a task package either within an ES response (the parameter taskPackage), or in a record retrieved from an ES database, within the parameter taskSpecificParameters within the structure defined by the record syntax ESTaskPackage; see REC.4.

'esRequest' consists of all service parameters supplied by the client in the ES request; these are divided into those that are and those that are not to be retained in the task package; 'toKeep' and 'notToKeep'. 'taskPackage' consists of all specific task parameters; which are divided into those supplied by the client and those supplied by the server, i.e. 'clientPart' and 'serverPart'. Note that 'toKeep' (from 'esRequest') is always the same sub-structure as 'clientPart' (from taskPackage), so that structure is shared, in ClientPartToKeep.

Each definition may define one or more of ClientPartToKeep, ClientPartNotToKeep, and ServerPart. In EXT.1, in the parameter table in the service definition for a specific ES, for each parameter:

- If the parameter is marked "client supplied," but is not marked in the right column (i.e. it does not occur in the task parameter package) then that parameter is represented in ClientPartNotToKeep.
- If the parameter is marked "client supplied," and also marked in the right column, then that parameter is represented in ClientPartToKeep.
- If the parameter is marked "server supplied" (in which case it will always also be marked in the right column), and not also marked "client supplied" then that parameter is represented in ServerPart.
- If the parameter is marked "client supplied," and also marked "server supplied" (in which case it will be marked in the right column), then it is a para-meter for which the client may suggest a value and the server may override that value. In this case the client suggested value is represented in ClientPartNotToKeep and the server value (which may be the same) is represented in ServerPart.

See ASN1.10

Appendix 8 USR: User Information Formats

Normative

UserInformation formats are defined for the following: userInformationField in the Init and InitResponse APDUs, additionalSearchInfo in the Search and SearchResponse APDUs, and otherInfo in all APDUs. UserInformation formats may include negotiation records, defined for the parameters userInformationField and otherInfo in the Init and InitResponse APDUs.

USR.1 SearchResult-1

The definition for the userInformation format SearchResult-1 is provided below; it is defined for use within a SearchResponse APDU. The following object identifier is assigned:

```
SearchResult-1   {Z39-50-userInfoFormat 1}
```

SearchResult-1 is for use primarily within the AdditionalSearchInformation parameter in the Search Response. The format allows the server to provide information per query component (the whole query or a sub-query, possibly restricted to a subset of the specified databases). The server may also create and provide access to a result set for each query component.

This format may also be used as a Resource Report format, within the ResourceReport parameter of the resource-control request, to allow the server to report on the progress of the search. However, when used in this manner, the server should not create a result set for a query component unless processing for that component is complete.

See ASN1.11

USR.2 Use of Init Parameters for User Information

The Init Request and Init Response both include the two parameters User- information-field and Other-information for provision of miscellaneous, externally-defined information. User-information-field was defined in Z39.50-1992 but Other-information was not in the 1992 version (it was first defined in Z39.50-1995). It is included in every Z39.50 APDU, but its use is valid only when version 3 is in force. The use of Other-information during initialization (i.e. within the Init request or response, but particularly in the request) is not recommended, because of the uncertainty, during initialization, of what protocol version, 2 or 3, is in force.

Other-information has a richly defined structure (in contrast to User- information-field, defined simply as EXTERNAL) developed for Z39.50-1995 to allow potentially complex combinations of information to be exchanged, when version 3 is in force. There is, however, an EXTERNAL definition, UserInfo-1 (see USR.3) that User-information-field may assume, identical to Other-information. Therefore the use of Other-information within the Init request and response may be avoided without loss of functionality.

Externally defined information carried within an InitRequest or InitResponse APDU should thus be carried within userInformationField, supplying the Object Identifier of UserInfo-1.

The structure may carry an arbitrary number of sequences, any of which may include a category and may be of any of the listed types: characterInfo, binaryInfo, externallyDefinedInfo, or oid. The category is optional; diagnostics and negotiation records need not include a category.

A diagnostic should be represented as externallyDefinedInfo; see DIAG.3 “Returning diagnostics in an InitResponse”. A negotiation record should be represented either as externallyDefinedInfo or oid, and is identified as a negotiation record by the object identifier (that is, the object identifier should identify a negotiation record definition), either within the EXTERNAL (for externallyDefinedInfo) or the object identifier itself (for oid).

USR.3 General User Information Format, UserInfo-1

The definition for the userInformation format UserInfoFormat-userInfo-1 is provided below; it is defined for use within the userInformationField parameter of the InitializeRequest or InitializeResponse APDUs, in situations where the otherInfo parameter cannot be used (specifically, when version 3 is not in force). The userInformationField parameter is defined simply as EXTERNAL, while the otherInfo parameter has a richer definition. The purpose of this definition is to register an object defined as identical to that richer definition that the userInformationField parameter may assume.

The following object identifier is assigned:

```
UserInfoFormat-userInfo-1 {Z39-50-userInfoFormat  3}
```

See ASN1.12

Appendix 9 ESP: Element Specification Formats

Normative

This appendix provides the definitions of the element specification formats eSpec-2 and sSpec-q, whose object identifier are:

eSpec-2 {Z39-50-elementSpec 2}

eSpec-q {Z39-50-elementSpec 3}

ESP.1 Definition of Element Specification Format eSpec-2

For description of element specifications and detailed semantics, see Appendix RET. This definition is based on an earlier element specification definition, eSpec-1 {Z39-50-elementSpec 1}, whose definition was provided in Z39.50-1995. Espec-2 is compatible with eSpec-1, but includes additional functionality, which is detailed in comments within the definition. It is recommended that implementors of this standard implement eSpec-2 (rather than eSpec-1). For interoperability, the following is also recommended:

- Servers who implement eSpec-2 (and not eSpec-1) should recognize the object identifier for eSpec-1. Thus when a client sends an element specification tagged as eSpec-1, treat it as though it were an eSpec-2 specification. (A conforming eSpec-1 specification will always conform to the eSpec-2 definition.)
- If a client is unable to interoperate with a server, because the server does not support eSpec-2, then so long as the element specification does not employ the schemaId, the client may send the specification using the eSpec-1 object identifier.

See ASN1.14 for ASN.1 Definition of eSpec-2.

ESP.2 Definition of Element Specification Format eSpec-q

Element specification definition eSpec-q consists essentially of a 'valueRestrictor' and an optional 'elementSelector'.

valueRestrictor

The valueRestrictor in the ASN.1 definition below takes the form of a type-1 query, whose purpose is to limit the scope of the retrieved information.

Example

Suppose eSpec-q is to be applied to holdings records (based on the holding schema, 1.2.840.10003.13.7). The valueRestrictor may be used to restrict the retrieved information

to holdings for a specific institution: In this case the valueRestrictor would take the form of a type-1 query where:

- The Access Point is institutionOrSiteId (corresponding to institutionOrSiteId within SiteLocation within HoldingsStatement in the Holdings schema).
- The term is a specific institution code, for example 'MdMC-T'.

elementSelector

The optional elementSelector in the ASN.1 definition below takes the form of an element specification, for example, eSpec-2 (which may degenerate to an element set name). It may be used in the normal manner, to select the actual desired elements to be retrieved (subject to restriction by valueRestrictor).

Example

Again suppose eSpec-q is to be applied to holdings. elementSelector may take the form of eSpec-2 to request that retrieved record be composed of siteLocation, dateOfReport, numberOfCopies, and UnionCatLendingInfo.

In the two examples above, the combined use of the valueRestrictor and elementSelector would result in the retrieval of siteLocation, dateOfReport, numberOfCopies, and UnionCatLendingInfo, for all holdings for which the institution code is 'MdMC-T', for all result set records indicated in the Present request.

If the elementSelector is omitted, the server chooses the element set.

See ASN1.15 for ASN.1 Definition of eSpec-q.

.

Appendix 10 VAR: Variant Sets

Normative

This appendix provides the definition for variant set variant-1, with object identifier:

`variant-1  {Z39-50-variantSet 1}`

This definition describes the classes, types, and values, for the variant set Variant-1, that may occur in a variant specification. A variant specification is a sequence of triples; each triple is a variant specifier (as referenced by the identifier variantSpecifier in GRS-1 and eSpec-2). The first component of the triple is a "Class" (integer), the second is a "Type" (integer) defined within that class, and the third is a "Value" defined for that type (its datatype depends on the type).

The following classes, types, and values are defined for Variant-1 (***For detailed semantics of variant-1, see Appendix RET)***.

Class 1: Variant Id

May be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Meaning	Datatype
1	Variant Id	OCTET STRING

Class 2: Body Part Type

May be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Meaning	Datatype/value
1	IanaType/subType	InternationalString: "<ianaType>/<subType> " e.g. "text/xml"
2	Z39.50Type [/subType]	InternationalString: e.g. "'sgml/'dtdName" (for example "sgml/TEI") or "sgml". Subtype is optional. See http://lcweb.loc.gov/z3950/agency/defns/body-z.html
3	otherType[/subType]	InternationalString: bilaterally agreed upon. Subtype is optional.

Type	Meaning	Datatype/value
4	identified by ISO Object Identifier	OBJECT IDENTIFIER;: When An ISO Object Identifier has been defined to identify a body part type, it may be used in lieu of a mime type (i.e where 'type' is 1,2, or 3, for 'iana', 'Z39.50', or 'other'). For example, to identify marc21, the ISO identifier 1.2.840.10003.5.10 may be used. That is an example where there is no alternative mime type defined. As another example, the OID 1.2.840.10003.5.109.1 identifies pdf (an example where there is an alternative mime type identification).

Class 3 Formatting/Presentation

May be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Meaning	Datatype/value
1	Characters per line	Integer
2	line length	IntUnit
3	lines per page	INTEGER
4	dots per inch	INTEGER
5	paperType-Size	InternationalString; e.g. A-1, B, C
6	deliverImages	BOOLEAN
7	portraitOrientation	BOOLEAN ('true' means "portrait")
8	textJustification	InternationalString; 'left', 'right', 'both', or 'center'
9	fontStyle	InternationalString
10	fontSize	InternationalString
11	fontMetric	InternationalString
12	lineSpacing	INTEGER
13	numberOfColumns	INTEGER
14	verticalMargins	IntUnit
15	horizontalMargins	IntUnit
16	pageOrderingForward	BOOLEAN
17	beginDocsOnNewPage	BOOLEAN ('false' means "concatenate documents")
18	termHighlighting	BOOLEAN

19	footnoteLocation	InternationalString: 'inline', endOfPage', 'endEachDoc', 'endLastDoc'
20	paginationType	InternationalString

Class 4: Language/Character Set

May be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Meaning	Datatype/value
1	language	InternationalString (from ANSI/NISOZ39.53-1994)
2	registered character set	INTEGER: registration number from ISO International Register of Character Sets
3	character set id	OBJECT IDENTIFIER
4	encoding id	OBJECT IDENTIFIER
5	private string	InternationalString

Class 5: Piece

Type	Meaning	Datatype/value
1 (variant Request only)	What fragment wanted	INTEGER startnextpreviouscurrentlast
2 (applied Variant only)	What fragment returned	INTEGER startmiddlelastend for nowwhole
Remaining types may be used within variant Request or appliedVariant		
3	start	IntUnit
4	end	IntUnit
5	how much	IntUnit
6	step	INTEGER or IntUnit
7	serverToken	OCTET STRING

Class 6: Meta-data Requested

May be used within a variantRequest only.

Type	Meaning	Datatype/value
1	cost	Unit or NULL
2	size	Unit or NULL
3	hits, variant-specific	NULL
4	hits, non-variant-specific	NULL
5	variant list	NULL

Type	Meaning	Datatype/value
6	is variant supported?	NULL
7	document descriptor	NULL
8	surrogate information	NULL
998	all meta-data	NULL
999	other meta-data	OBJECT IDENTIFIER

Class 7: Meta-data Returned

May be used within a supportedVariant or appliedVariant.

Type	Meaning	Datatype/value
1	cost	IntUnit
2	size	IntUNitUnit
3	integrity	integer
4	separability	integer
5	variant supported	boolean
6	variant description	InternationalString; used by server to provide text description of a variant

Class 8: Highlighting

May be used within a supportedVariant or appliedVariant.

Type	Meaning	Datatype/value
1	prefix	octet string
2	postfix	octet string
3 (variantRequest only)	server default	NULL

Class 9: Miscellaneous

Type	Meaning	Datatype/value
1 (variant Request only)	no data	NULL
2 (variant Request only)	unit	Unit (client requests element according to specific unit)
3	version	InternationalString

Type	Meaning	Datatype/value
4	variant description (Used when variant specifications such as mime type, size, etc. are not sufficiently descriptive when there are various representations and attributes of an object, such as *highly compressed*, *high resolution*, *reference image*, and *original*; these characterizations are not for client use but rather, descriptive information for the user. See also class 7, type 6).	NULL; used by client to request text description of a variant
5	content is a pointer (See also class 2, type 4.)	NULL; • On an applied variant, indicates that the content of the element is a pointer (e.g. URL) to the actual data, not the actual data itself. • On a variant request, indicates that a pointer to the actual data is requested, not the actual data itself. • On a supported variant, indicates that a pointer to the actual data is available. (This may occur in conjunction with another supported variant for the same element that does not use this variant spec, allowing the client to select either the actual data or a pointer.)

Appendix 11 TAG: TagSet Definitions and Schemas

Normative

A database schema represents a common understanding shared by the client and server, of the information contained in the records of the database represented by that schema, to allow retrieval of portions of that information.

The primary component of a database schema is an abstract record structure, which lists schema elements in terms of their tagPaths. A tagPath is a representation of the hierarchical path of an element, expressed as a sequence of nodes, each represented by a tag. Each tag in a tagPath consists of a tagType and tagValue. The tagType is an integer; the tagValue may be an integer or character string. The tagType qualifies the tagValue; it might identify a tagSet, which might be registered (or alternatively, it might be defined locally within the schema).

Also included in a schema is a definition of how the various tagTypes are used within the tagPaths for the schema elements. The definition might simply be a mapping of tagTypes to tagSets.

For all schemas, tagTypes 1 through 3 are assumed to have the following meaning:

tagType	Used to Qualify:
1	An element defined in tagSet-M (see TAG.1)
2	An element defined in tagSet-G (see TAG.2)
3	A tag locally defined by the server (intended primarily for string tags, but numeric tags are not precluded)

For a detailed description of the use of schemas, tagSets, etc. see Appendix RET.

This appendix provides definitions for the tag sets tagSet-M and tagSet-G. TagSet-M includes elements intended for use as meta-data associated with a database record (or portion of a database record). TagSet-G includes generic elements.

The object identifier for these definitions are:

```
tagSet-M  {Z39-50-tagSet 1}

tagSet-G  {Z39-50-tagSet 2}
```

For detailed semantics of the elements defined in these tagSets, see Appendix RET.

Note: With the exception of tag sets M and G, there is no relationship between the tagType value (see above) and the last component of the tagSet OID. TagSet-M is 1.2.840.10003.14.1 and the tagType value is always 1, and tagSet-G is 1.2.840.10003.14.2 and the tagType value is always 2. However this relationship does not carry further. TagType value 3 is defined as 'locally defined' (while the OID 1.2.840.10003.14.3 corresponds to a registered tagSet).

Schemas definitions provide mappings of (scalar integer) tagType to tagSet. Tag types 1, 2, and 3 are well-known; beginning with 4, the tagSet referenced depends upon the schema that is in effect. Thus the tagType is shorthand for an OID, where the binding of tagType to OID is defined by the schema, and different schemas may define different bindings.

For example, in a GRS-1 retrieval record where tagType 4 occurs, if the Collections schema is in effect, it refers to the collections tagSet. If the GILS schema is in effect, tagType 4 refers to the GILS tagSet. Another schema might use both Collections and GILS elements and may assign tagTypes 4 and 5 to Collections and GILS respectively.

TAG.1 Definition of tagSet-M

Tag	Element Name	Datatype	Meaning
1	schemaIdentifier	OBJECT IDENTIFIER	Identifies the schema in use. This element is available for cases where the client does not specify a schema in the request, or where the server uses a schema different than that requested by the client.
2	elementsOrdered	BOOLEAN	If 'true', then sibling elements (i.e. with the same parent) are presented as follows: tagTypes are ascending; for elements with the same tagType, integer tag values are ascending, and precede elements with string tags (which are not necessarily ordered).
3	elementOrdering	INTEGER	How sibling elements with the same tag are ordered: 1 = "Normal" consumption order (pages, frames) 2 = Chronological, e.g., news articles 3 = Semantic size, e.g. increasingly comprehensive abstracts 4 = Generality, e.g. thesaurus words, increasing generality, concentric object snapshots, zoom-out order 5 = Elements explicitly undistinguished by order 6 = undefined; may (or not) be ordered by private agreement 7 = Singleton; never more than one occurrence
4	defaultTagType	INTEGER	The tagType that applies for any element for which tagType is not included.
5	defaultVariant SetId	OBJECT IDENTIFIER	The Variant set identifier that applies when the server returns a variant specification for an element, but does not include a variant set identifier.
6	defaultVariant Spec	VariantSpec	If this element is present, then the specified variant applies to all subsequent elements, when applicable, which do not include a variant specification.
7	processing Instructions	International String	Recommendation by the server on how to display this record to the user
8	recordUsage	INTEGER	1 = Redistributable 2 = Restricted, and the tagSet-M element 'restriction' (defined below) contains the restriction 3 = Restricted, and the restriction, contains a license pointer

Tag	Element Name	Datatype	Meaning
9	restriction	International String	This element, if present, should immediately follow recordUsage, and is a statement (if recordUsage is 1 or 2), or a pointer to the license (if recordUsage is 3).
10	rank	INTEGER	The rank of this record within the result set. If N records are in the result set, each record should have a unique rank from 1 to N.
11	userMessage	International String	A message, pertaining to this record, that the server asks the client to display to the user
12	uri	International String	Uniform resource identifier. This is a URI for the record.
13	record	structured	This element may be used for nested records, when the database record itself includes database records (possibly from a different database). Note that tagSet-M elements that occur subordinate to this element apply only to that nested record.
14	local control number	same as tagSet-G element 'identifier'	An identifier of the record, unique within the database. May be used to indicate a record's unique id for use within a Z39.50r url (RFC 2056), that is, for subsequent search/retrieval, using the identifier as a search term with bib-1 Use attribute docid (1032) and structure attribute URx (104).
15	creation date	same as tagSet-G element dateTime	Date that the record was created
16	dateOfLast Modification	same as tagSet-G element dateTime	Most recent date that this record was modified
17	dateOfLast Review	same as tagSet-G element dateTime	Most recent date that this record was verified
18	score	INTEGER	A normalized score assigned to the record by the server. Each record in the result set may have a score from 1 to N where N is the normalization factor (more than one record may have the same score). The normalization factor should be specified in the schema.
19	wellKnown	Defined by schema; default Inter nationalString	When an element is defined to be "structured into locally defined elements," the server may use this tag in lieu of, or along with, locally defined tags. For example, an element named 'title' might be described to be "locally structured." The server might present the element structured into the following subelements: 'wellKnown', "spineTitle," and "variantTitle," where the latter two are string tags, server defined. In this case, 'wellKnown' is assumed to mean "title."
20	recordWrapper	structured	This element may be used to represent the root of the record, particularly when the record

Tag	Element Name	Datatype	Meaning
			otherwise has no root. The client may request the record skeleton by reference to this element.
21	defaultTagSetId	OBJECT IDENTIFIER	This element may be used in lieu of defaultTagType, to identify the default tag set.
22	languageOf Record	Same as tagSet-G element 'language'	
23	type	INTEGER or International String	
24	Scheme	INTEGER or International String	
25	costInfo	International String	
26	costFlag	BOOLEAN	'true' means there is a cost
27	Record Created By	International String	
28	Record Modified By	International String	

TAG.2 Definition of tagSet-G

TAG.2.1 Principles, Usage, and Scope of TagSet-G

TagSet-G elements may be used:

1. Generically
2. Within a specific community
3. In a schema context
4. As utility elements, within a Tag Path
5. As utility container elements

Generic Usage

TagSet-G includes elements with common usage within a significant number of user communities. In the absence of a schema that specifies stricter semantics, TagSet-G elements have broad, loose semantics. Each tagSet-G element assumes generic (unspecified) semantics when the element occurs outside the context of any specific schema. These generic semantics are (in general) weaker than the semantics of that element as defined within a schema.

Within a Specific Community

TagSet-G elements implicitly inherit more specific semantics when used within a specific user community. For example, the Museum community can use the Title element interoperably within that community. Similarly, the Genealogy community uses Title as well, but with different implied semantics.

Schema context of TagSet-G Element

Suppose, however, the “Museum of Genealogy” wants to provide records; it will need a schema that defines the semantics of Title. Thus when used across communities, a schema should be used to tighten semantics of tagSet-G elements.

The intent is that general elements may be inherited by schema for more specific usage. Thus a tagSet-G element may be attributed stronger semantics when it occurs within the context of a specific schema. The element Author (as another example) has generic semantics such that if it occurs outside the context of a schema it might be interpreted as "Author or Creator". A specific schema (for example, pertaining to museum objects) may confine its meaning to "Creator".

A tagSet-G element may occur within a retrieval record as generic (including within a specific community) or context specific, and this depends respectively on whether it occurs within or not within the context of a specific schema. A client may infer that a generic occurrence pertains to the record at large, and the client should attribute generic semantics. If the occurrence is context specific and if the client does not support the context (i.e. the schema), the client should not infer any information from the occurrence. A tagSet-G element may have both context specific and generic occurrences within the same record; in that case if the client does not support the context, the client may infer that the generic occurrences apply, while ignoring any context specific occurrences.

For example, suppose a client executes a search across databases creating a result set that represents records from potentially several domains. When the client retrieves a record from the result set, unless the retrieval record includes a schema identifier, the client might not know what schema governs the interpretation of the elements of the record. Suppose in this case the first several elements are from tagSet-G, say: Author, Title, and subject. Following these initial tagSet-G elements, assume there is a structured element with subelements, the first of which is a schemaIdentifier (tagSet-M element 1). Suppose that the client does not support the identified schema, and suppose further that there are several tagSet-G elements following the schema identifier. The client is able to discern that the Author, Title, and Subject pertain to the record, but the client is not able to process the record further, in particular, the client may not infer information from the second set of tagSet-G elements. In this case, the client may be able to inform the user that there is a potential record of interest, and that in order to interpret the record, support for the specific schema is necessary.

As another example, consider a database where an individual record corresponds to a physical object (for example, a work of art) and the record includes one or more digital renditions of the object. An abstract record structure may specify that tagSet-G elements may occur at the beginning of the record, outside the context of a specific schema, followed by a schema identifier, followed by a repeating, structured element, with a repetition for each rendition. There may be tagSet-G elements within each such "rendition" element, and these would pertain to the specific rendition. (The 'Title' or 'Author' for the first rendition may be different from those of the 'second' rendition. For example, the first rendition may be "black and white still image, 35mm" where the listed 'Author' is the photographer; while the second rendition may be a sketch of the physical object, where the listed 'Author' is the artist of the sketch. The Title and Author in both cases would be different from those of the object itself.)

TagSet-G element used as utility elements, within a Tag Path

TagSet-G elements may be used as utility elements within a tag path. For example consider an element ‘availability' subordinate to which is an element ‘distributor’ and thus is constructed the element availability/distributor. Subordinate to this element are several tagSet-G elements: name, organization, address, and telephone. Thus are constructed the following elements:

- “Availability: distributor name”, constructed as availability/distributor/name
- “Availability: distributor organization”, constructed as availability/distributor/organization
- “Availability: distributor address”, constructed as availability/distributor/address

· "Availability: distributor telephone", constructed as availability/distributor/telephone

In this scheme these four tagSet-G elements play the role of utility elements, thus avoiding the need to define special tags for these commonly used elements. The element "Availability: distributor name", is a ***schema element*** and the tagSet defined in conjunction with the schema may need to define 'availability' and 'distributor' (if there is no other known tagSet that defines these) but it will not have to define 'name' (and similarly for the other three schema elements).

A client receiving a record that includes these schema elements should not infer any information from the presence of these tagSet-G elements, unless the client supports that schema.

Utility container elements

Elements such as displayObject and documentContent are available as container elements.

TAG.2.2. TagSet-G Elements

Tag	Element Name	Datatype (/format/usage)
1	title	InternationalString, or structured into following sub-elements: 'wellKnown', tagSet-M element 19, dataType InternationalString 'type' tagSet-M element 23 'scheme' tagSet-M element 24
2	author	Same datatype definition as title
3	publicationPlace	InternationalString
4	*see note 1*	
5	*see note 2*	
6	*see note 3*	
7	Name	Same datatype definition as title
8	DateTime *See note 4*	EXTERNAL (Z3950DateTime), or GeneralizedTime, or InternationalString, or structured into following sub-elements: 'wellKnown', tagSet-M element 19, dataType 1, 2, or 3 above;'type' tagSet-M element 23;'scheme' tagSet-M element 24
9	DisplayObject *See note 5*	OCTET STRING The server might combine several elements into this single element, into a display format, for display
10	organization	InternationalString
11	postal Address	InternationalString
12	Network Address	InternationalString
13	eMail Address	InternationalString
14	phone Number	InternationalString
15	faxNumber	InternationalString
16	Country	InternationalString, or structured into following sub-elements: 'wellKnown', tagSet-M element 19, dataType InternationalString 'scheme' tagSet-M element 24
17	description	InternationalString, or structured into following sub-elements: 'wellKnown', tagSet-M element 19, dataType InternationalString 'scheme' tagSet-M element 23
18	*See note* 6	
19	DocumentContent	OCTET STRING

20	language	Same dataType definition as country
21	subject	Same dataType definition as title
22	resource Type	Same dataType definition as country
23	City	InternationalString
24	stateOr Province	InternationalString
25	zipOr PostalCode	InternationalString
26	Cost	InternationalString, or IntUnit, or structured into following sub-elements: 'wellKnown', tagSet-M element 19, dataType InternationalString or IntUnit;'costInfo' tagSet-M element 25; datatype: InternationalString;'costFlag' tagSet-M element 26; datatype: Boolean
27	Format	Same dataType definition as country
28	Identifier	Same dataType definition as title
29	Rights	Same dataType definition as title
30	Relation	Same dataType definition as title
31	Publisher	Same dataType definition as title
32	Contributor	Same dataType definition as title
33	Source	Same dataType definition as title
34	Coverage	Same dataType definition as title
35	Private	dataType definition defined by schema
36	database Name	InternationalStringFormat: Z39.50s URL. See RFC 2056. The database name (optional in RFC 2056) must be supplied. Meaning/usage: Identifies a Z39.50 database
37	Recorded	InternationalStringFormat: Z39.50r URL as described in RFC 2056. Meaning/usage: Identifies a Z39.50 database record

Notes:

1. Tag 4 was PublicationDate in Z39.50-1995. It is recommended that publicationDate not be used; it may be covered by 'date', qualified by 'type'
2. Tag 5 was documentId in Z39.50-1995. It is recommended that documentId not be used; it may be covered by 'identifier', qualified by 'type'
3. Tag 6 was abstract in Z39.50-1995. It is recommended that abstract not be used; it may be covered by 'description', qualified by 'type'
4. Tag 8 was date in Z39.50-1995. It has been generalized to dateTime
5. Tag 9 was bodyOfDisplay in Z39.50-1995. It has been renamed DisplayObject
6. Tag 18 was time in Z39.50-1995. It is recommended that time not be used; it may be covered by tag 8, 'dateTime'

Appendix 12 ERS: Extended Result Set Model

Non-normative

Section 3.1.6, Model of a Result Set, notes that in the extended result set model for searching, the server maintains unspecified information associated with each record, which may be used as a surrogate for the search that created the result set. Query specifications may indicate under what condition the extended model applies and the nature of the unspecified information. This appendix provides examples of information that the server might maintain to perform proximity operations requiring the extended model, or to evaluate restriction operands.

ERS.1 Extended Result Set Model for

Proximity

In the extended result set model for proximity, the server maintains information associated with each record represented by the result set, that may be used in a proximity operation as a surrogate for the search that created the result set.

Example:

Let R1 and R2 be result sets produced by Type-1 query searches on the terms 'cat' and 'hat'. In the extended result set model for proximity, the server maintains sufficient information associated with each entry in R1 and with each entry in R2 so that the proximity operation "R1 near R2" would be a result set equivalent to the result set produced by the proximity operation "cat near hat" ("near" is used informally to refer to a proximity test).

The manner in which the server maintains this information is not prescribed by the standard. The concept of "abstract position vectors" may be used to describe the effect of the proximity test. A server system may implement the proximity test in any way that produces the desired results.

An abstract position vector might include a proximity unit and a sequence of position identifiers.

Example:

Let R1 and R2 be result sets produced by searches on the terms 'cat' and 'hat'. Record 1000 contains 'cat' in paragraphs 10 and 100 and 'hat' in paragraphs 13 and 200. So record 1000 is represented in both R1 and R2. In R1, it might include the two position vectors (paragraph, 10) and (paragraph, 100). In R2, it might include the two position vectors (paragraph, 13) and (paragraph, 200). R3 = "R1 within 10 paragraphs of R2" would identify this record, and a position vector might be created (paragraph, 10, 13).

Subsequently, suppose R4 represents "rat before bat" and includes record 1000 with position vectors (paragraph, 5, 8) and (paragraph, 15, 18). Then:

- R3 'before and within 2 of' R4 would represent: "(cat near hat) before (rat before bat)" and in the resulting set, record 1000 might include position vector (paragraph, 10,

18);

- R3 'following and within 2 of' R4 might represent: "(cat near hat) after (rat before bat)" and in the resulting set, record 1000 might include position vector (paragraph, 5, 13).

Note: In these two examples, the position vectors might instead be (paragraph, 10, 13, 15, 18) instead of (paragraph, 10, 18); and (paragraph, 5, 8, 10, 13) instead of (paragraph, 5, 13). Different implementations might interpret extended proximity tests differently.

Neither the information that the server maintains (associated with result set entries to be used in the proximity operations) nor the manner in which the server maintains this information, is prescribed by the standard. The above is supplied as an example only.

ERS.2 Extended Result Set Model for Restriction

The Restriction operand specifies a result-set-id and a set of attributes. It might represent a set of database records identified by the specified result set, restricted by the specified attributes, as in example 1 (below). It might represent a set of records from the database specified in the Search APDU, indirectly identified by the specified result set and restricted by the specified attributes, as in example 2.

Example 1:

Let R be the result set produced by a search on the term 'cat'.

Result set position:

1. identifies record 1000, where 'cat' occurs in the title.
2. identifies record 2000, where 'cat' occurs in the title and as an author.
3. identifies record 3000, where 'cat' occurs in the title, and as an author and subject.

Then "R restricted to 'author'" might produce the result set consisting of the entries 2 and 3 of R.

In the extended result set model for restriction, the server maintains information that allows this type of search to be performed. In this example, the server might maintain the following information with the entries in result set R:

Result set position:

1. title
2. title, author
3. title, author, subject

Example 2:

In this example, R and C are two databases. R is a "registry" database containing records about chemical substances, each of which is identified by a unique registry number. C is a bibliographic database, containing bibliographic records for documents about chemical substances. The registry number is a searchable field in both databases. A registry number identifying a record in R may occur in one or more logical indexes for database C.

For example, the "preparations" index for database C contains registry numbers of substances that are cited in its documents as being used in preparations.

In this example, a search is performed against database R, creating result set L, which will in effect contain registry numbers representing records in database R, each of which uniquely

identifies a chemical substance. A second search is performed against database C with the operand "L restricted to 'preparations'." This restriction is expressed by applying the "preparations" attribute to result set L. The search is performed by looking for registry numbers from result set L that occur in the "preparations" index for database C. The result set represents the records in C where a registry number contained in result set L occurs as a preparation.

In the extended result set model for restriction, the server maintains information that allows this type of search to be performed. In this example, the server might maintain, with each entry in L, a list of identifiers of records in C for which the registry number occurs as a preparation.

Neither the information that the server maintains (associated with result set entries to be used in the evaluation of a Restriction operand), nor the manner in which the server maintains this information, is prescribed by the standard. The above are supplied as an example only.

Appendix 13 RET: Z39.50 Retrieval

Non-normative

Search and retrieval are the two primary functions of Z39.50. Searching is the selection of database records, based on client-specified criteria, and the creation by the server of a result-set representing the selected records. ***Retrieval***, idiomatically speaking, is the transfer of result set records from the server to the client.

This appendix describes retrieval, and thus assumes the existence of a result set. For simplicity, it is assumed that the result set has a single record (although Z39.50 retrieval allows a client to request the retrieval of various combinations of result set records) and this appendix focuses on the capabilities provided by Z39.50 retrieval for retrieving information from that record.

RET.1 Overview of Z39.50 Retrieval

Though retrieval is considered informally to be the transfer of result set records, a result set, logically, does not contain records. Rather, it contains logical items (sometimes referred to as "answers"); each item includes a pointer to a database record (the term "result set record" is an idiomatic expression used to mean "the database record represented by a result set item").

Moreover, a database record, as viewed by Z39.50, is purely a local data structure. In general Z39.50 retrieval does not transfer database records (that is, the server does not transfer the information according to its physical representation within the database), nor does Z39.50 necessarily transfer all of the information represented by a particular database record; it might transfer a subset of that information.

Thus the "transfer of a result set record" more accurately means: the transfer of some subset of the information in a database record (represented by that result set entry) according to some specified format. This exportable structure transferred is called a retrieval record. (Multiple retrieval requests for a given record may result in significantly different retrieval records, both in content and structure.)

Z39.50 retrieval supports the following basic capabilities:

- The client may request specific logical information elements from a record (via an element specification, described below).
- The client and server may share a name space for tagging elements (via a schema and tagsets, described below), so that elements will be properly identified: by the client, within an element specification, and by the server, within a retrieval record.
- The client may request an individual element according to a specific representation or format (via variants, described below).
- The client may specify how the elements, collectively, are to be packaged into a retrieval record (via a record syntax, described below).

Correspondingly, Z39.50 retrieval has four primary functions:

- Element ***selection*** (see note)
- Element ***tagging***
- Element ***representation***
- ***Record*** representation

Note: element selection pertains to retrieval, and should not be confused with record selection which pertains to searching. Element selection pertains to selection of information elements from already-selected database records.

RET.2 Retrieval Object Classes

This section, RET.2, describes object classes used by these retrieval functions: RET.3 describes in detail specific object definitions that are defined within this standard.

- element specifications (elementSpecs), see RET.2.1;
- tagSets, see RET.2.1;
- schema definitions, see RET.2.2;
- variant specifications (variantSpecs), see RET.2.3; and
- record syntaxes, see RET.2.4.

RET.3 describes in detail specific object definitions that are defined within this standard.

Following is a brief overview of the object classes.

An elementSpec occurs within a Z39.50 Present request, and is used primarily for selection. In its most basic form, an elementSpec is a request for specific elements (a set of elementRequests).

A tagSet defines a set of elements, and specifies names and recommended datatypes for individual elements within that set. The name of an element is called its tag, and may be used alone (in an elementRequest) or accompanying the element it names (within a retrieval record).

A schema defines an abstract record structure (see RET.2.2). The schema definition refers to one or more tagSets.

Although an elementSpec is used primarily for selection, it might have representation aspects: each elementRequest may include a variantRequest, used primarily for element representation, to specify the particular form of an element, for example how an element is to be formatted. (However, a variantRequest may include limited selection: it might ask for a specific ***piece*** or ***fragment*** of an element.)

A variantRequest is one of three usages of a variantSpec:

- A variantRequest is a variantSpec occurring within an elementRequest.
- An appliedVariant is a variantSpec applied to an element by the server, when that element is included in a retrieval record.
- The server might provide a list of the variantSpecs supported for a given element; each is referred to as a supportedVariant.

A record syntax is applied by the server to the set of elements selected by an elementSpec (and possibly transformed by appliedVariants) resulting in a retrieval record.

Summarizing:

- An elementSpec is used (primarily) for element selection;
- A variantRequest is used for element representation;
- A record syntax is used for record representation;
- A tagSet is used for element tagging, both within an elementSpec (for element selection) and a record syntax (for record representation).
- A schema defines an abstract record structure.

RET.2.1 Element Specification Features and TagSets

An elementSpec may be included in a Present request to specify the desired elements to comprise a retrieval record. For example, the client might request that the retrieval record consist of the two elements 'author' and 'title'. The elementSpec may express this in one of two ways:

- An element set name (a primitive name) might be defined, for example 'authorTitle', whose definition means "present the author and title."
- A dynamic specification may be used, allowing the client to select arbitrary elements, dynamically.

The use of an element set name as an elementSpec has a significant limitation: one would need to be defined for every possible combination of elements that might be requested.

For Z39.50 version 2, only the primitive form is allowed; the elementSpec must be an element set name (whose ASN.1 type is VisibleString). Version 3 allows the elementSpec to alternatively assume the ASN.1 type EXTERNAL (thus referencing an external definition, which is presumably, though not necessarily, described in ASN.1). The following illustrate some of the features that may be provided by an elementSpec, by progressively complex ASN.1 examples.

RET.2.1.1. Simple numeric tags

A simple elementSpec might specify a list of elements. The elementSpec definition could be:

```
ESpec ::=SEQUENCE OF ElementRequest

ElementRequest ::=INTEGER
```

In this example, each element requested is represented by an integer. Both client and server are assumed to share a common definition, a tagSet, which assigns integers to elements. The integer is the name, or tag, of the element. In this example, the tagSet might assign the integers 1 to 'title' and 2 to 'author'.

RET.2.1.2 String tags

It is not always desirable to restrict element tags to integers. String tags are useful for some applications. So the element request might take the slightly more complex form:

```
ElementRequest ::= StringOrNumeric
```

Note that StringOrNumeric is a type defined within, and exported by Z39-50-APDU, defined as:

```
StringOrNumeric ::= CHOICE{

      numeric [1] IMPLICIT INTEGER,

      string [2] IMPLICIT InternationalString}
```

In this case, the tagSet might declare that "author may also be referenced by the string tag 'author', and title by 'title'."

RET.2.1.3 Tag Types

Often it will be necessary (or useful) to request elements not all of whose tags are defined by a single tagSet. This capability presents an important benefit, allowing multiple name spaces for tags, so that tagSet definitions may be developed independently. However, it requires that tags be qualified by reference to tagSet.

A schema definition (see RET.2.2) may assign an integer to identify a tagSet (it identifies the tagSet only within the context of the schema definition). This tagSet identifier is called a tagType. Note that a tagSet definition is a registered object and thus is persistently identified by an object identifier. The (integer) tagType is used as a short-hand identifier.

Extending the above example to incorporate tagTypes, the elementRequest could be defined as:

```
ElementRequest ::= SEQUENCE{

      TagType [1] IMPLICIT INTEGER,

      TagValue [2] StringOrNumeric}
```

RET.2.1.4 Tag Occurrence

A database record often contains recurring elements. A client might want the Nth occurrence of a particular type of element (e.g. "the fourth image"). To introduce recurrence into the above example, the elementRequest could be defined as:

```
ElementRequest ::= SEQUENCE{

      TagType [1] IMPLICIT INTEGER,

      TagValue [2] StringOrNumeric,

      TagOccurrence [3] IMPLICIT INTEGER}
```

RET.2.1.5 Tag Paths

A database record is not necessarily a flat set of elements, it may be a hierarchical structure, or tree (where leaf-nodes contain information). A client might request, for example "the fourth paragraph of section 3 of chapter 2 of book 1" ('book', 'chapter', 'section', and 'paragraph' might be tags). This example introduces the concept of a tag path, which is simply a nested sequence of tags (each tag within the sequence is qualified by a type and occurrence). A tag path can be incorporated by replacing the first line of ASN.1 in the previous example, with:

```
ElementRequest ::= TagPath

TagPath ::= SEQUENCE OF SEQUENCE{
```

RET.2.1.6 VariantRequests

Finally, the client may wish to qualify an elementRequest with a variantRequest, to specify a particular composition (e.g. PDF), language, character set, formatting (e.g. line length), or fragment.

```
ESpec ::=  SEQUENCE OF ElementRequest

      ElementRequest ::= SEQUENCE{

      TagPath,

      VariantRequest OPTIONAL}
```

Where TagPath is defined as in the previous example. Variants are described in RET.2.3.

RET.2.2 Schema and Abstract Record Structure

A database schema represents a common understanding shared by the client and server of the information contained in the records of the database represented by schema. The primary component of a schema is an abstract record structure, ARS. It lists schema elements in terms of their tagPaths, and supplies information associated with each element, including whether it is mandatory, whether it is repeatable, and a definition of the element. (It also describes the hierarchy of elements within the record; see RET.2.2.5.)

An ARS is defined in terms of one or more tagSets. The schema itself may define a tagSet, and may also refer to externally defined tagSets. In the simple example of an ARS that follows, assume that the following tagSet has been defined:

Tag	Element	Recommended dataType
1	title	InternationalString
7	name	InternationalString
16	date	GeneralizedTime
18	score	INTEGER
14	recordId	InternationalString
<locally defined string tag>	objectElement	InternationalString or OCTET STRING

In the following example ARS, each "schema element" refers to an element from the above tagSet.

In this example, for objectElement, the schema would indicate that the server is to assign some descriptive string tag. For example, if the element is a fingerprint file, the tag might be 'fingerPrintFile'. (In that case, the content of element 'name', tag 7, might identify the person who is the subject of the finger prints.) Since it is the only element in the ARS with a string tag, the client will recognize it as the objectElement.

Abstract Record Structure

Schema Element	Mandatory?	Repeatable?	Definition
title	yes	no	A set of words that conveys the main idea of the record.
name	no	yes	One or more individuals associated with the object element; it could, for example, be an author or an organization.
date	no	no	A date associated with the record.
score	no	no	Represents the numerical score of the record based on its relevance to the query
recordId	no	no	An identifier of the record unique within the server system.
ObjectElement	yes	no	Contains object information for the record. It may be text, image, etc.

RET.2.2.1 Relationship of Schema and TagSet

In the above example, at first glance it appears there need not be separate tables for tagSet and ARS, they could be combined into a single table. When the tagSet is defined within a schema, then there may be no need to distinguish between the tagSet and schema. However, the tagSet might instead be defined externally and referenced by the schema.

A schema may define a tagSet as in the example above, and it need not be registered. The schema could simply assign an integer tagType to identify the tagSet. The tagSet could then be used only by that schema. But some of the elements in the above example might also be included in a different schema. For example, another schema might also define title and name, and that schema should be able to use the same tags. For this purpose, tagSets may be registered, independent of schema definitions.

It is anticipated that there will be several, but not a large number of tagsets defined, and that many schemas will be able to define an ARS referencing one or more registered tag sets, without the need to define a new tagSet. (There will be more than one tagSet defined because it would be difficult to manage a single tagSet that meets the needs of all schemas.)

RET.2.2.2 TagTypes

As noted in RET.2.1.3, within a Present request or Present response elements are identified by their tag, and tags are qualified by ***tag type***. The tag type is an integer, identifying the tagSet to which it belongs. A schema lists each tagSet referenced in its ARS and designates an integer to be used as the tag type for that tagSet.

Z39.50 currently defines two tagSets, tagSet-M and tagSet-G. These are described in RET.3.4. TagSet M includes elements to be used primarily to convey meta-information about a record, for example dateOfCreation; tagSet-G includes primarily generic elements, for example 'title', 'author'.

Among the schema elements defined in the example above, title and name are defined in tagSet-G; date, score, and recordId are defined in tagSet-M.

The schema might provide the following mapping of tagType to tagSet:

1 --> tagSet-M

2 --> tagSet-G

3 --> locally defined tags (intended primarily for string tags, but numeric tags are not precluded)

In the notation below, where (x,y) is used, 'x' is the tagType and 'y' is the tag. In the ARS above the following column would be inserted on the left:

TagPath

(2,1)

(2,7)

(1,16)

(1,18)

(1,14)

(3,<locally defined string tag>)

RET.2.2.3 Recurring objectElement

The schema becomes only slightly more complex if multiple object elements (i.e. multiple occurrences of the element objectElement) are allowed. The schema could indicate that each occurrence of objectElement is to have a different string tag. The entry in the 'repeatable' column in the ARS, for objectElement, would be changed from 'no' to 'yes'.

For example, suppose a record includes a fingerprint file, photo, and resume, all describing an individual (and the element 'name' might identify the individual that they describe). The string tags for these three elements respectively might be 'fingerPrint', 'photo', and 'resume'. The client would recognize each of these elements as an occurrence of objectElement, because the schema designates that only objectElement may have a string tag. (This is not to imply that the client would recognize the type of information, e.g. fingerprint, from its string tag; but the client might display the string tag to the user, to whom it might be meaningful.)

The ARS would be as follows (definition column omitted):

Tag path	Element	Mandatory?	Repeatable?
(2,1)	title	yes	No
(2,7)	Name	no	Yes
(1,16)	Date	no	No
(1,18)	Score	no	No
(1,14)	RecordId	no	No
(3,<stringTag>)	Object Element	yes	Yes

RET.2.2.5 Structured Elements

In the following example, hierarchy is introduced; the ARS includes structured elements (i.e. elements whose tagPath has length greater than 1). In the examples above the ARSs are flat; all elements are data- elements, i.e. leaf-nodes. The ARS below is part of a schema for a database in which each record describes an information resource. It assumes the following tagSet:

TagElement	NameRecommended	DataType
25	linkage	InternationalString
27	recordSource	InternationalString
51	purpose	InternationalString
52	clientator	InternationalString
55	orderProcess	InternationalString
70	availability	(structured)
90	distributor	(structured)
94	pointOfContact	(structured)
97	crossReference	(structured)

The notation (x,y)/(z,w) is used below to mean element (z,w) is a sub-element of element (x,y). In the "Schema Element Name" column, indentation is used to indicate subordination. For example,

distributorName, a data element, is a sub-element of the structured element distributor, which in turn is a sub-element of the structured element availability. In this example, the schema designates that the

tagType for the above defined tagSet is 4.

Several elements in the ARS below are (implicitly) imported from tagSet-G (those with tagType-2). These are: title, abstract, name, organization, postalAddress, and phoneNumber.

Abstract Record Structure:

Schema-element Tag-path	Schema-element Name
(2,1)	title
(2,6)	abstract
(4,51)	purpose
(4,52)	clientator
(4,70)	availability
(4,70)/(4,90)	distributor
(4,70)/(4,90)/(2,7)	distributorName
(4,70)/(4,90)/(2,10)	distributorOrganization
(4,70)/(4,90)/(2,11)	distributorAddress
(4,70)/(4,90)/(2,14)	distributorTelephone
(4,70)/(4,55)	orderProcess
(4,70)/(4,25)	linkage
(4,94)	pointOfContact
(4,94)/(2,7)	contactName
(4,94)/(2,10)	contactOrganization
(4,94)/(2,11)	contactAddress
(4,97)	crossReference
(4,97)/(2,1)	crossReferenceTitle
(4,97)/(4,25)	crossReferenceLinkage
(4,27)	recordSource

The ARS describes an abstract database record consisting of title, abstract, purpose, clientator, availability, point of contact, crossReference, and recordSource. These are the "top-level" elements, among which, Availability, pointOfContact, and CrossReference are structured elements, and the others are data elements. Availability consists of distributor, orderProcess, and Linkage; among these, distributor is a structured element.

RET.2.3 Variants

An element might be available for retrieval in various forms, or ***variants***. The concept of an element variant applies in three cases:

- The client may request an element (in a Present request) according to a specific variant
- The server may present an element (in a Present response) according to a specific variant
- The server may indicate what variants of a particular element are available

Correspondingly, and more formally, a variant specification (***variantSpec***) takes the form of a variantRequest, appliedVariant, or supportedVariant. In all cases, a variantSpec is a sequence of variantComponents, each of which is a triple (class, type, value). 'class' is an integer. 'type' is also an integer and a set of types are defined for each class. Values are defined for each type.

A variantSet definition is a registered object (whose object identifier is called a variantSetId) which defines a set of classes, types, and values that may be used in a variantComponent. A variantSpec is always qualified by its variantSetId, to provide context for the values that occur within the variantComponents (in the same manner that an RPN Query includes an attribute set id, to provide context for the attribute values within the attribute lists).

The variant set definition variant-1 is defined in Appendix VAR, and is described in detail, in RET.3.3

RET.2.4 Record Syntax

The server applies a record syntax to an abstract database record, forming a retrieval record. Record syntaxes fall into two categories: content-specific and generic. Content-specific record syntaxes include:

- Those of the MARC family (listed at the beginning of Appendix REC)
- Explain (REC.1)
- Extended Services (REC.4)

Generic record syntaxes are further categorized: they are structured or unstructured. Structured record syntaxes are able to identify TagSet elements. GRS-1, a generic, structured syntax, is defined in REC.3, and is described in detail in RET.3.2. SUTRS (Simple Unstructured Text Record Syntax) is a generic, unstructured syntax, defined in REC.2.

RET.3 Retrieval Objects Defined in this Standard

In the remainder of this Appendix, detailed descriptions are provided below for the following retrieval objects defined in this standard: element specification format eSpec-2, record syntax GRS-1, variant set variant-1, and tagSets tagSet-M and tagSet-G. Within these descriptions it is assumed that these objects are used together; for example, in the description of eSpec-2 it is assumed that GRS-1 is to be used as the record syntax. In general, however, no such restriction applies; eSpec-2 may be used as an element specification in conjunction with SUTRS for example.

RET.3.1 Element Specification Format eSpec-2

The element specification format eSpec-2 (which replaces eSpec-1, defined in Z39.50-1995) is defined in Appendix ESP. An element specification taking this form is basically a set of elementRequests, as seen in the last member of the main structure:

```
elements [4] IMPLICIT SEQUENCE OF   ElementRequest
```

Each elementRequest may be a "simple element" or a "composite element," as distinguished by the ElementRequest definition:

```
ElementRequest::= CHOICE{

      SimpleElement [1]

      CompositeElement [2]
```

Simple elements are described in RET.3.1.1. A composite element is constructed from one or more simple elements, described in RET.3.1.2. Note however an elementRequest which takes the form of simpleElement might actually result in a request for multiple elements. See RET.3.1.1.3.

The element specification may include additional elementRequests, resulting from 'elementSetNames' in the first member of the main sequence. All elementRequests resulting from 'elementSetNames' are simple elements.

Also included in the main structure are a default variantSetId and a default variantRequest. These are described in RET.3.1.1.5.

RET.3.1.1 Simple Element

A request for a simple element consists of the tagPath for the element, together (optionally) with a variantRequest. The tagPath identifies a node of the logical tree (or possibly several trees) representing the hierarchical structure of the abstract database record to which the element specification is applied.

A tagPath is a sequence of nodes from the root of a tree to the node that the tagPath represents, where each node is represented by a tag. The end-node of a tagPath might be a leaf-node containing data, or a non-leaf node; in the latter case, the request pertains to the entire subtree whose root is that node.

For example, suppose an Abstract Record Structure defines:

```
(4,1): A

(4,1)/(4,2): B

(4,1)/(4,3): C
```

Then a request for (4,1) is equivalent to requesting the two subelements, (4,1)/(4,2) and (4,1)/(4,3), individually.

wildThing (see RET.3.1.1.4.1) using an occurrence value of 'all', is also equivalent, when there are only these two subelements. However, if these elements are known, and are known to be the only two elements then using (4/1) is sufficient and simpler than (4,1)/wildThing[all].

GRS-1 will present the subtree recursively (see RET.3.2.1.1).

RET.3.1.1.1 Tag

Each tag is qualified by a tagType. Thus a tag consists of a tagType and a tagValue. (A tag is further qualified by its "occurrence"; see RET.3.1.1.2.) Each tagType is an integer, and each tagValue may be either an integer or string.

Every tag along a tagPath is assumed to have a tagType, either explicit or implicit; it may be supplied explicitly within the specification, and if it is omitted, a default applies (the default should be listed within the schema in use). Tags along a tagPath may have different tagTypes.

RET.3.1.1.2 Occurrence

Each node along a tagPath is distinguished not only by its tag, but also by its occurrence among siblings with the same tag. A record might contain recurring elements, and the client might wish to request the Nth occurrence of a particular element (e.g. "the fourth image"). The specification of the "occurrence" of a node may be omitted, in which case it defaults to 1. Occurrence may explicitly be specified as "last" (this capability is provided for the case where the client does not know how many occurrences there are, but however many, it wants the last).

RET.3.1.1.3 Multiple Simple Elements

In some cases a 'simpleElement' request (within the ElementRequest structure) results in multiple simple elements. This may occur in the following cases:

- If a tagPath identifies a non-leaf node, the request represents the entire subtree (it is logically equivalent to individual simple requests for each subordinate leaf-node).
- occurrence' may be specified as 'all, meaning "all nodes with a given tag."
- ‘occurrence' may be specified in the form of a range (e.g. 1 through 10).
- The tagPath may include a wild card (see RET.3.1.5) in lieu of a specific tag.

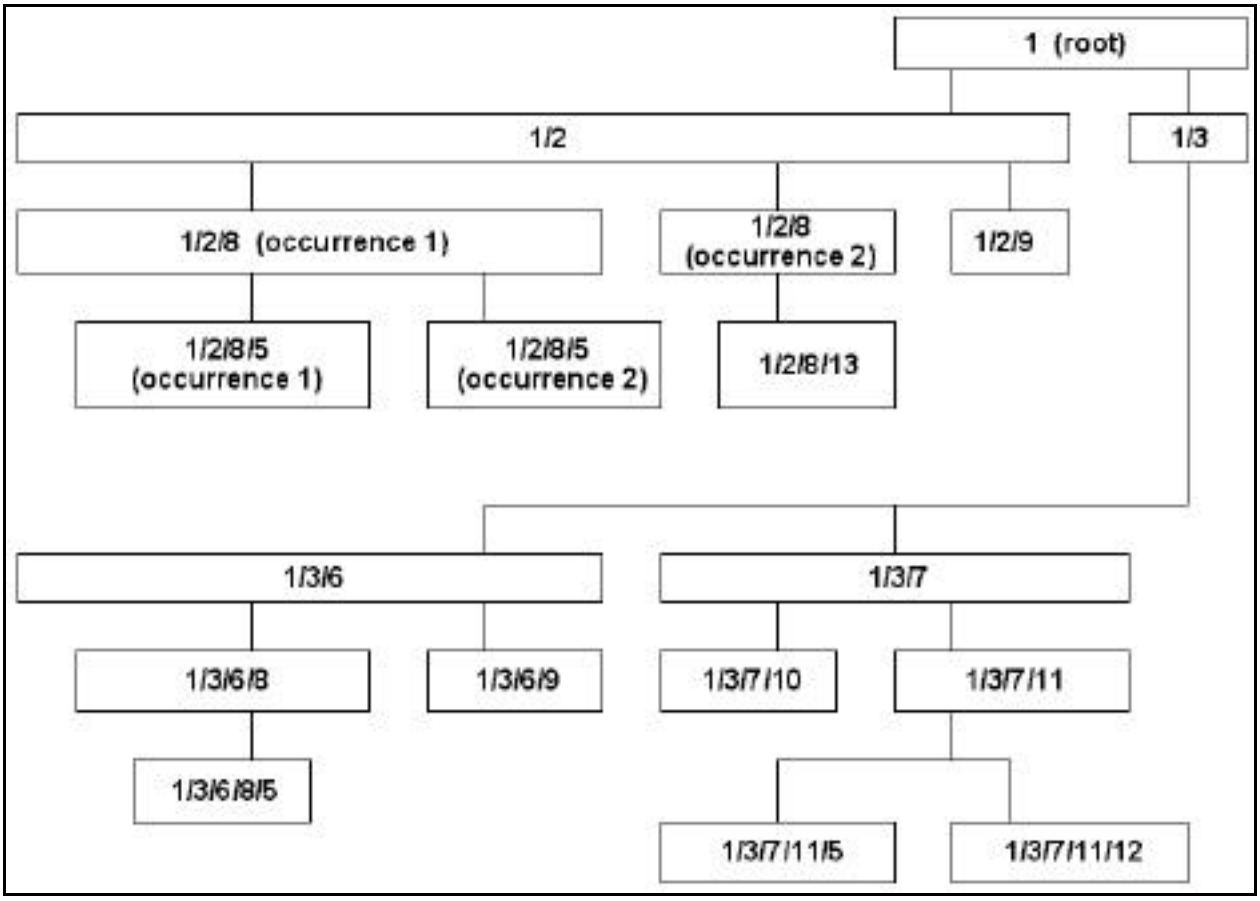
RET.3.1.1.4 Wild-cards

A tagPath may be viewed as an expression containing tags and wild cards. There are two types of wild cards, wildThing and wildPath, described in RET.3.1.1.4.1 and RET.3.1.1.4.2.

The client may request all subordinate elements (not necessarily immediately subordinate) of a particular type

using wildPath, or the first N elements (or all elements at this level) regardless of type, using wildThing.

For this discussion of wild-cards, consider the sample record whose hierarchical structure is shown in the diagram below.



Each cell in the diagram represents an element whose tagPath is indicated within the cell. The numbers within the tagPath are tagValues; for simplicity, tagTypes are omitted, and assumed all to be the same. Leaf-nodes are highlighted by double-lined cells.

For example, the tagPath 1/3/7 represents the (non-leaf-node) element with tag 7 subordinate to the element with tag 3 subordinate to the element with tag 1. 1/3/7/11/12 represents the element whose (leaf-) node has tag 12.

RET.3.1.1.4.1 WildThing

A tagPath expression may include the wild card 'wildThing' in lieu of a tag. WildThing takes the form of an occurrence specification. For example, the tagPath expression '1/2/wildThing (occurrence 3)' would represent the node 1/2/9, because it is the third child of the node 1/2.

The expression '1/wildThing (occurrence 2)' would be equivalent to the path 1/3 (it refers to the entire subtree whose node has tag 3).

RET.3.1.1.4.2 WildPath

A tagPath expression may include the wild card 'wildPath' in lieu of a tag. WildPath matches any sequence of tags, along any path such that the tag following wildPath in the expression follows that sequence in the matched path. For example, either of the expressions 'wildpath/5' or '1/wildPath/5' would result in all paths ending in 5. It would match:

1/2/8 (occurrence 1)/5 (occurrence 1)

1/2/8 (occurrence 1)/5 (occurrence 2)

1/3/6/8/5, and

1/3/7/11/5

The expression '1/2/wildPath/5' would match the first two listed above, and the expression '1/3/wildPath/5' would match the last two.

WildPath could be used, for example, to retrieve all (and only) captions, where the tag for captions is known, and where captions are spread throughout a document at various levels of its hierarchy. It may not be the last member of a tagpath sequence.

RET.3.1.1.5 Variant Request

Each request for a simple element may optionally include a variantRequest. Note that the main structure of eSpec-2 optionally includes 'defaultVariantRequest'. If the element request does not include a variantRequest then 'defaultVariantRequest' applies if it occurs in the main structure. If the element request does not include a variantRequest and 'defaultVariantRequest' does not occur in the main structure, there is no variant request associated with the element request.

The main structure also optionally includes 'defaultVariantSetId'. A variant specification may or may not include a variantSetId. If the element request includes a variantRequest which does not include a variantSetId, then 'defaultVariantSet' applies. (If the element request includes a variantRequest which does not include a variantSetId, and if 'defaultVariantSet' does not occur in the main structure then the variantRequest is in error.)

RET.3.1.2 Composite Elements

An elementRequest for a compositeElement takes the form of a list of simple elements (as described in RET.3.1; alternatively, the simple elements may be specified by one or more element set names), a delivery tag, so it can recognize the composite element on delivery, and an optional variantRequest (for example including a mime type of a format used to package the elements) . The simple elements are to be combined by the server to form a single (logical) element, to which the (optional) composite variant is to be applied, and the server is to present the element using the supplied delivery tag.

RET.3.2 Generic Record Syntax GRS-1

A GRS-1 structure is a retrieval record representing a database record. Its logical content is a tree representing the hierarchical structure of the abstract database record, or a sequence of trees if the abstract record itself does not have a root.

RET.3.2.1 General Tree Structure

The top level "SEQUENCE OF TaggedElement" might be a single instance of TaggedElement, representing the root of a single tree representing the record (in the degenerate case, the record consists of a single element). Alternatively, the top-level SEQUENCE OF might contain multiple instances of TaggedElement, in which case there is no single root for the record; the record is represented by multiple trees, any or each of which might be a single element (thus the GRS-1 structure may represent a flat sequence of elements).

Any leaf-node within the GRS-1 structure might correspond to an individual elementRequest that was included in the corresponding eSpec-2 element specification. A non-leaf node may correspond to an elementRequest; if an eSpec-2 elementRequest tagPath ends at a non-leaf node, then the request is for the entire subtree represented by that node.

RET.3.2.1.1 Recursion and SubTrees

Each instance of TaggedElement may, via recursion, contain a subtree. Beginning at the root of the tree (or at one of the top level nodes) TaggedElement identifies an immediately subordinate node, via tag and occurrence. If the CHOICE for 'content' is 'subtree', then the identified node is a non-leaf node: 'subtree' is itself defined as SEQUENCE OF TaggedElement, so the next level of nodes is thus defined. Recursion may be thus used to describe arbitrarily complex trees.

RET.3.2.1.2 Leaf-nodes

Along any path described by the GRS-1 record, eventually a leaf-node is encountered ('content' other than 'subtree').

The content of the leaf-node is one of the following:

- Data; see RET.3.2.2
- Empty, for one of the following reasons:
  - The requested element does not exist.
  - It exists, but there is no data.
  - The elementRequest specified (via a variant-1 variantRequest) that no data was to be returned. (This is probably because only meta-data was desired. So it is likely that the variantRequest also requested meta-data, and that meta-data accompanies this node; see RET.3.2.3.)
- A diagnostic

RET.3.2.2 Data

When a leaf-node contains data, then 'content' is one of the following ASN.1 types: OCTET STRING, INTEGER, GeneralizedTime, EXTERNAL, InternationalString, BOOLEAN, OBJECT IDENTIFIER, or IntUnit. That is, the CHOICE for ElementData is one of these, and the actual data must assume the chosen type. An appliedVariant may also be indicated, by including appliedVariant from the main structure.

RET.3.2.3 Meta-data

When a leaf-node contains data or is empty, 'metaData' may be included, containing meta-data for the element. The meta-data may be included along with the data, or in lieu of the data if the elementRequest asked that no data be returned (i.e. 'content' is 'noDataRequested'). Meta-data would not be included when 'content' is 'elementNotThere', 'elementEmpty', or 'diagnostic'.

MetaData for a leaf-node may be any or all of the following:

- ***usageRight***: the server may declare that the element is freely distributable, or that restrictions apply. In the latter case, the server supplies either a restriction in the form of a text message, or a license pointer.
- ***hits***; see RET.3.2.3.1
- ***displayName***: A name for the element, suggested by the server, for the client to display.
- ***supportedVariants***; see RET.3.2
- ***message***: A message for the client to display to the user, associated with this element.
- There is also one case where meta-data may be included for a non-leaf node:
- ***seriesOrder***; see RET.3.2.3.2

RET.3.2.3.1 Hits

Associated with an element may be one or more hit vectors. Each points to a fragment within the element. Each such fragment bears some relationship to the search which caused the record (to which the element belongs) to be included in the result set (from which the record is being presented). Note that the association of a hit vector to an element is meaningful only within the context of that search.

A hit vector may optionally include a 'satisfier': for example, a term from the query, which occurs within that fragment of the element (to which the hit vector points).

The server might return hit vectors along with an element, so that the client may be able to quickly locate the satisfying portions of the element, and perhaps even highlight the satisfier(s) for display to the user.

The server might return part of an element and include hit vectors, some of which point within the retrieved portion, and others which point to fragments not included, to indicate to the client what fragment to request to retrieve other relevant parts of the element.

A hit vector may include location information: offset (location within the element where the fragment begins) and length. Both are expressed in terms of IntUnit, so for example, the location information might indicate an offset of "page 10" and length of "one page," meaning that the satisfier occurs on page 10 (or that the fragment is page 10).

Note: if there are multiple hit vectors with the same satisfier, occurring on the same page, and if the server wishes to indicate 'rank' (see below), it will need to use a unit with finer-granularity than 'page'.

The hit vector may also include 'rank', relative to the other hits occurring within this set of hitVectors. Rank is a positive integer with a value less than or equal to the number of hit vectors. More than one hit may share the same rank.

Finally, the server may assign a token to the hit vector, which points to the fragment associated with the hit. The client may use the token, subsequently but within the same Z-association, within a variantRequest (in an elementRequest) to retrieve (or to refer to) the fragment.

The server might provide location information, or a token, which may be used subsequently to retrieve the specific fragment. The server might provide both location information and a token: for example, the location information might indicate "page 10"; the client may subsequently retrieve the pages before and after, inclusive (i.e pages 9-11). If the server also supplies a token, the client might retrieve the "previous fragment" or "following fragment."

Location information is always variant-specific. A token, however, may be variant-specific or variant-independent. The client might request "hits: non-variant-specific" for an element (via variant-1), and specify 'noData'. The hit vectors returned would be variant-independent (thus only a token, and no location information, would be included in each hit vector). The client could subsequently use a token in an elementRequest to retrieve the corresponding fragment, independent of what variantRequest was included in the elementRequest.

The client might request 'hits: variant-specific' for an element, for a particular variant. The server might return location information or tokens, or both, but in any case, the hit vectors would apply only for that variant. The client could subsequently use either the location information or token in an elementRequest to retrieve the corresponding fragment, but only when specifying that variant.

As an alternative to hit vectors, see "Highlighting," RET.3.3.1.8.

RET.3.2.3.2 Series Order

The server might include the meta-data 'seriesOrder' (for a non-leaf node only). It indicates how immediately-subordinate elements with the same tag are ordered. Values are listed in TAG.2.1, but may be overridden by the schema.

The values are the same as those for elementOrdering (see RET.3.4.1.2.3) which applies at the record level (i.e. it applies throughout the record, and pertains wherever sibling elements with the same tag occur).

RET.3.3 Variant Set Variant-1

This section describes the variant set variant-1.

RET.3.3.1 variant-1 Classes

This section describes the classes, types, and values defined for the variant set variant-1.

RET.3.3.1.1 VariantId

Variant-1 class 1, 'variantId', may be used to supply an identifier for a variant specification. (There is only one type within class 1, so the variantId is always class 1, type 1). It is a ***transient*** identifier; it may be used to identify a particular variant specification during a single Z-association. (A variantId should not be confused with ***variant set id***, which identifies a ***variant set definition***.)

A variantId may be included within a supportedVariant, variantRequest, or appliedVariant. The variantList for an element may be supplied by the server (see 3.3.2). It consists of a list of supportedVariants for the element. Each may include a variantId, which may be used subsequently by the client within a variantRequest (within an elementRequest), to identify that supportedVariant (i.e. that variant form of the element), in lieu of explicitly constructing a variant. A variantId may be used within an appliedVariant, supplied by the server in case the client wishes to use it in a subsequent request, possibly overriding some of the variant parameters.

RET.3.3.1.2 BodyPartType

Variant-1 class 2, 'BodyPartType', allows representation of the structure, or "body part type," of an element. It may be used within a supportedVariant, variantRequest, or appliedVariant.

There are three types: type 1 is ianaType/subType, for content types registered with IANA (Internet Assigned Numbers Authority). Type 2 is for body part types registered by the Z39.50 Maintenance Agency (type 2 is used generally for formats that have not yet been otherwise officially registered). Type 3 is for bilaterally agreed upon body part types.

Following are some of the IANA contentType/Subtypes registered.

(See http://www.iana.org/assignments/media-types/index.html)

Type	Subtype
text	plain
	richtext
	tab-separated-values
	html
	xml

application	octet-stream
	sgml
image	jpeg
	gif
	tiff
audio	basic
video	mpeg
	quicktime

SGML, for example, would be indicated by the triple (2,1, 'application/sgml'). Before SGML was registered IANA, it could be referred to as a Z39.50 body part type: (2,2, 'sgml/<dtd>') where <dtd> is the name of the SGML dtd.

A Z39.50 body part type will be registered only if it is not registered as an IANA type. If it is subsequently adopted by IANA, it is recommended that it be referenced as such.

RET.3.3.1.3 Formatting/Presentation

Variant-1 class 3, 'formatting', may be included within a variantRequest, appliedVariant, or supportedVariant. It indicates additional formatting parameters such as line length, lines per page, font style, and margins.

RET.3.3.1.4 Language/CharacterSet

Variant-1 class 4, 'language/characterSet', may be included within a variantRequest, appliedVariant, or supportedVariant. It indicates language and/or character set.

RET.3.3.1.5 Piece

Variant-1 class 5, 'piece' may be included within a variantRequest (type 1) or appliedVariant (type 2), to refer to a specific piece or fragment of an element.

The client may use type 1 to request:

- A fragment beginning at the beginning of the element ('start')
- The 'next' fragment (relative to the fragment indicated by serverToken, see type 7)
- The 'previous' fragment
- The 'current' fragment (the fragment indicated by serverToken)
- The 'last' fragment (within the element)

The server may use type 2 to indicate that the presented fragment:

- Begins at the beginning of, but is not the whole element ('start');
- Neither starts at the beginning of, nor ends at the end of the element ('middle');
- Does not begin at the beginning of, but ends at the end of the element ('end');
- Ends at the end of the element, but the element may grow in the future ('endForNow'); or
- Is the 'whole' element.

The server may use types 3, 4 (or 5), and 6 in lieu of type 2, to indicate the 'start' and 'end' (e.g. starts at page 1 and ends at page 100) or 'start' and 'howMuch' (e.g. starts at page 1, 100 pages) of the fragment and optionally, a 'step' size. For example, the server could indicate that the

fragment starts at byte 10,000 and ends at byte 20,000 (in this case a step of 1 would be indicated, or implied if 'step' is omitted); or it starts on page 100, ends on page 200, and includes every 5th page.

Similar, the client may use types 3, 4 (or 5), and 6 to request a fragment. In a variantRequest these types may be used to further qualify a fragment indicated by types 2 and 7. For example, the request might specify a serverToken, previous fragment (5,1,3), as well as a start and end, in which case the start and end are relative to the indicated fragment, i.e., relative to the fragment immediately prior to that indicated by the server token.

The server may use type 7 in an appliedVariant to supply a token as an identifier of the supplied fragment, and the client may subsequently use the token in a variantRequest to identify that fragment.

RET.3.3.1.6 MetaData Requested

Variant-1 class 6, 'meta-data requested' may be included within a variantRequest, to request meta-data associated with an element.

The client might want to know, for example, the cost to retrieve a particular element in Html, as well as the page count (of the Html form of the element). The following variant specifiers would be included within the variantRequest for that element:

(2,1, 'application/html')	-- Html
(6,1, NULL)	-- cost, please
(6,2, Unit:pages)	-- size in pages, please
(9,1, NULL)	-- no data (just the above metaData)

Alternatively, a variantId might be used in place of a set of explicit specifiers (i.e. in place of the html specifier, in this example) if the client knows the variantId of a variant for which it wants cost or size information. (Although if the client knows the variantId, it may already have cost or size information because it may have obtained that id within a variantList, and if so, the server may have included the cost and page information within the supportedVariant.)

The client might also ask for the location of hits within the element (see RET.3.2.3.1). An element might have hits which are specific to a variant, and may also have non-variant-specific hits. The request above might also ask for hits specific to the particular variant (i.e. html), using (6,3, NULL) or non-variant-specific hits, using (6,4, NULL). In either case, the request is for the server to return hit vectors within the retrieved GRS record.

The client may request that the server supply the variant list for an element via the specifier (6,5, NULL). The server would supply the variant list (consisting of a list of supportedVariants) within the GRS structure (not within the appliedVariant). See RET.3.3.2.

The client may use (6,6, NULL) to inquire whether a particular variant is supported. An example is provided in RET.3.3.2.

RET.3.3.1.7 Meta-data Returned

Variant-1 class 7, 'meta-data returned' may be included within an appliedVariant or supported-Variant. There are several categories of element MetaData. Those of class 7: cost, size, integrity, and separability, are singled out for representation within variant-1, because the server may include those within a supportedVariant. Other metaData, including hits and variantList, are included within the GRS-1 structure. Hits are described in RET.3.2.3.1.

RET.3.3.1.8 Highlighting

Variant-1 class 8, 'highlighting', may be included within a variantRequest or an appliedVariant. Highlighting may be used as an alternative, or in addition, to hit vectors, described in RET.3.2.3.1.

The client may include 'prefix' and 'postfix' in a variantRequest to request that the server insert the specified strings into the actual data, surrounding hits, so that the client, upon retrieving the data, may simply locate the strings, for fast access to the hits. The client may use 'server default' in lieu of 'prefix' and 'postfix' to indicate that it the server should select the strings for highlighting.

The server may include 'prefix' and 'postfix' in an appliedVariant to indicate the strings used within the element for highlighting hits.

RET.3.3.2 VariantList

The thoroughness of the variantList supplied by the server may depend on the implementation. For example, for an element (representing a document) which the server provides in Html, consider the following cases:

- The document might already exist in print format, and the server might support only that single html variant.
- The server might support a few variants forms, varying by language.
- The server might support many variant forms; varying by language.
- The server might support many variant forms, varying by language, and also varying by formatting/presentation parameters, including lines per page, font style, etc.

The server might list a single supportedVariant in the variant list for the element, indicating that the element is available in html. In that case the client cannot necessarily conclude which of the above cases applies. The server might instead list three supportedVariants, each indicating html and a language. In that case, it may be reasonable for the client to surmise that the element is available in those three languages only, but the client probably cannot deduce which formatting parameters apply. The server might further indicate one or more formatting parameters within each supportedVariant. Again, the extent to which the client may deduce what other variations are supported will depend on the implementation.

The client may explicitly inquire whether a particular variant is supported, by constructing the desired variant (including all of the desired formatting parameters, etc.) and indicating "is variant supported?," using the triple (6,6, NULL). The variantRequest might also request that the server provide cost (6,1, NULL) and size (6,2, NULL) information if the variant does exist. The server would respond that the requested variant is or is not supported by supplying an appliedVariant (with the element) with the same parameters, and including the triple (7,5, TRUE or FALSE). If the server indicates TRUE (that the variant is supported) it may also supply a variantId that the client may then use to request the variant.

The client may construct a variantRequest that includes a variantId along with additional variant specifiers. Suppose the server lists the following supportedVariant:

(1,1, <variantId>)	-- identifies this variant
(2,1, 'application/html')	-- in html
(4,1, 'por')	-- language: Portuguese

The element is thus available in Html, in Portuguese. The client may submit a variantRequest consisting of only:

(1,1, <variantId>)

to request the element in html, in Portuguese.

Suppose, instead, the server lists the following supportedVariant:

(1,1, <variantId>) -- identifies this variant

(2,1, 'application/html') -- in html

Thus the server indicates that the element is available in Html, but no other variant information is provided.

The client may submit a variantRequest consisting of only:

(1,1, <variantId>)

(4,1 'por')

Again, this is to request the element in html, in Portuguese.

Or, the client may submit the following variantRequest:

(1,1, <variantId>)

(4,1, 'por')

(4,2, 84) -- Portuguese character set

(5,3, page 1) -- begin on page 1

(5,4, page 100) -- end on page 100

to request the element in html, in Portuguese, Portuguese character set, pages 1-100.

RET.3.4 TagSets Defined in the Standard

Appendix Tag defines two tagSets, tagSet-M (for elements which convey meta- and related information about a record or an element within a record) and tagSet-G (primarily for generic elements). These two tagSets are described in RET.3.4.1 and RET.3.4.2.

RET.3.4.1 TagSet-M

TagSet-M defines a set of elements that the server might choose to return within a retrieval record, even though the element was not requested and in fact is not actually information contained within the database record. Rather, it is information ***about*** the database record, retrieval record, or result set record; or it might pertain to an element within a record. Within a GRS-1 record, the server returns tagSet-M elements in exactly the same manner that it returns elements from any other tagSet.

TagSet-M elements fall into three categories.

- Meta-information about the database record:
  - processingInstructions
  - recordUsage
  - restriction
  - userMessage
  - url
  - local control number

- creation date
- dateOfLastModification
- dateOfLastReview

· Elements defined to facilitate the construction and processing of the retrieval record:

- schemaIdentifier
- elementsOrdered
- elementOrdering
- defaultTagType
- defaultVariantSetId
- defaultVariantSpec
- record
- wellKnown
- recordWrapper

· Elements pertaining to the record's entry in the result Set:

- rank
- score

RET.3.4.1.1 Meta-Information

The definitions for these elements are provided in TAG.2.1. Any of these elements may or may not actually occur within the database record. However, it is emphasized that these elements describe the database record itself (or part of it) as distinguished from a resource that the database record might correspond to.

For example, tagSet-M element 'url' refers to a URL for the database record. The database record itself may contain URLs for resources that the record describes; tagSet-M element 'url' does not pertain to those.

RET.3.4.1.2 Information about the Retrieval Record

RET.3.4.1.2.1 schemaIdentifier

A retrieval record is in general meaningful within the context of a schema definition. In many (perhaps most) cases the server may reasonably expect that the client knows which schema definition applies to a particular retrieval record. In those cases the server need not explicitly identify the schema. This element is provided for cases where there is a possibility of uncertainty about which schema applies.

This element is also useful for retrieval records that include subordinate or nested records which are defined in terms of different schemas. See RET.3.4.1.2.5.

This element, if provided, will normally (but not necessarily always) occur as the first element within the retrieval record (or within a subordinate or nested record) and for that reason is assigned tag 1, in case the server wishes to present elements in numerical order (see RET.3.4.1.2.2).

Example

An abstract record structure is developed to provide different level of semantic interoperability to accommodate various levels of client-awareness, and for purposes of cross-domain searching, employing nested schemas to provide levels of semantic interoperability, including the notion of no governing schema at the most generic level, at which point generic metadata elements (for example, Dublin Core) may be inserted.

Assume two applications, A and B, with respective schemas Schema A and Schema B, where B is a special case of A, thus a system that supports Schema B always supports Schema A.

In this example, the retrieval record provides three levels of semantic interoperability. It specifies information at the beginning that a client may understand even if the client does not recognize any specific schema, followed by information that a client may understand if it recognizes Schema A but not necessarily Schema B, followed by information understandable to a client that recognizes Shema B.

The abstract record structure for application B specifies the following:

1. At the beginning of a retrieval record there may occur generic metadata (in the form of one or more tagSet-G elements).
2. Following that, Schema A is assumed, and there is information recognizable to Schema A client, including additional metadata elements.
3. Following this information, Schema B is assumed.

By including metadata at the top level of the retrieval record a generic client may be able to partially, if not fully process these records. At the next level, a Schema-A-aware (but not Schema-B-aware) client who performs a distributed search over multiple databases involving multiple disciplines may locate a Schema B record, and discover that there is a potential record of interest, even though the client is not able to fully process the record. At the highest level of semantic interoperability, a Schema-B-aware client may be able to fully process a Schema B retrieval record.

So, when the schemaIdentifier occurs as (and only as) the very first element, it applies to the entire retrieval record. However the schemas may change in the middle of a retrieval record, subject to the following guidelines.

A schema identifier may legally occur anywhere within a retrieval record as long as it is not preceded by siblings (leaf or non-leaf). That is, it must be either the first occurring element in the record or the first occurring element subordinate to its parent. The identified schema governs all elements at the same level as, or subordinate to, the schema identifier, unless superseded by a subordinate schema identifier. For example, consider the following retrieval record:

element 1: Schema Identifier A

element 2 (leaf)

element 3 (structured)

element 4 (leaf)

element 5 (structured)

element 6: Schema Identifier B

element 7 (leaf)

element 8 (structured)

element 9 (leaf)

element 10 (structured)

element 11: Schema Identifier C

element 12 (leaf)

element 13 (leaf)

element 14 (leaf)

element 15 (leaf)

element 16 (leaf)

element 17 (leaf)

Schema A (identified by element 1) governs elements 2, 3, 4, 5, 16, and 17

Schema B (identified by element 6) governs elements 7, 8, 9, 10, 14, and 15

Schema C (identified by element 11) governs elements 12 and 13

If there is no schema identifier at the top of a retrieval record, then two cases should be considered:

1. A schema identifier occurs later within the record.
2. There is no schema identifier at all within the record.

In the first case, the elements that occur prior to the first schema identifier should be assumed to occur outside the context of any specific schema, so these should include tagSet-G and TagSet-M elements only. Thus as a rule of thumb, whenever a retrieval record includes a schema identifier not as the first element, then it should also include a schema identifier as the first element, unless it intends that no schema be in effect prior to encountering the schema identifier.

In the second case, it is possible that there is a known schema in effect, either because a schema identifier was included in the retrieval request, or because there is a prior understanding between client and server about what schema is in effect. As a rule of thumb, if the server is not certain that there is a prior understanding, or if the schema in effect is not the schema requested, then the server should insert the schema identifier at the beginning of the record.

RET.3.4.1.2.2 elementsOrdered

This is a BOOLEAN flag indicating whether the elements of the retrieval record are presented in order by tag. The ordering is described in TAG.2.1. This element is defined because it may be useful for a client to know whether elements are presented in order, when trying to locate a particular element within the retrieval record.

This element, if provided, should normally be occur as the first element within the retrieval record, or the second if schemaIdentifier is provided, and for that reason is assigned tag 2.

RET.3.4.1.2.3 elementOrdering

For a retrieval record containing recurring elements, i.e. sibling elements with the same tag, the server might present these elements according to some logical order, for example, chronological,

increasing generality, concentric object snapshots, or normal consumption (i.e. pages, frames). This element indicates the order; values are listed in TAG.2.1. Note that the values are the same as those for seriesOrder (see RET.3.2.3.2) which applies at the element level, i.e. it pertains to sub-elements of an element. This element, elementOrdering, applies at the record level, i.e. it applies throughout the record, and pertains wherever sibling elements with the same tag occur.

RET.3.4.1.2.4 Defaults (tagType, variantSetId, and variantSpec)

defaultTagType, if provided, is the assumed tag Type for presented elements where the tagType is omitted. It is defined solely to allow simplification of the retrieval record. If there is a predominant tagType within the retrieval record, this meta-element allows the server to omit the tagType for those element with that tagType.

Note that the schema may also list a default tagType. If so, then defaultTagType, if it occurs, overrides the schema-listed default. If the schema does not list a default tagType, and if this element does not occur, then every tag within the retrieval record must include a tagType.

defaultVariantSetId is the assumed variantSetId for appliedVariants within the retrieval record that omit the variantSetId. defaultVariantSpec, if provided, is the assumed appliedVariant for all elements within the retrieval for which an appliedVariant is not provided. The schema may also list a default variantSetId and/or appliedVariant. If so, then these elements if they occur, override the schema-listed default. If the schema does not list a default variantSetId and default Variant SetId is not provided, then every applied Variant within the retrieval record must include a variantId. If the schema does not list a default applied Variant and defaultVariantSpec is not provided, then for elements within the retrieval record for which an appliedVariant is not supplied, no appliedVariant is assumed to apply.

RET.3.4.1.2.5 Record

The tagSet-M element 'record' may be used to present nested or subordinate records.

A retrieval record represents a single database record, but that database record may contain elements which in turn represent database records (possibly replicated from a different database). For example, a database may contain records representing queued database updates. Each such record might contain a set of database records to be contributed to some other database. As another example, an OPAC database might have records defined to each include a bibliographic record and a corresponding holdings record, and the holdings record in turn might include a series of circulation records.

It is important to note that although a single retrieval record may include an arbitrary number of subordinate records, or arbitrarily nested records, the retrieval record nevertheless represents a single result set record.

A subordinate (or nested) record defined in this manner may be presented according to a schema different from the schema applying to the retrieval record. The tagSet-M element schemaIdentifer may be included within the element representing a record, and if so, it applies only within that element.

RET.3.4.1.2.6 wellKnown

Some schema developers anticipate that for certain elements, different servers will want to provide several alternative forms of the element. The element 'wellKnown' is defined in order to support this flexibility.

Suppose a schema defines the element 'title'. The intent may be that the server simply return a single value, what the server considers to be the title. In that case, 'title' should be a leaf-node defined from tagSet-G, and 'wellKnown' does not apply.

But suppose the server wishes to return the element 'title' encompassing several forms of the title, including one which the client will recognize to be the default in case it does not understand any of the others (in which case it may ignore all except the default, or may still display them to the end-user, who might understand them even if the client does not). The client returns the single element 'title', which is structured into the following sub-elements:

- the default title
- 'abbreviatedKeyTitle'
- 'formerTitle'
- 'augmentedTitle'
- 'romanizedTitle'
- 'shortenedTitle'

The additional forms of title (i.e. those other than the default title) might use the above string tags, locally defined, or they may be known tags defined in other tag sets. However, the default title has a distinguished integer tag, that assigned to the tagSet-M element wellKnown, to distinguish it.

The element wellKnown is thus always subordinate to a parent element whose semantics are known (e.g. 'title', 'address', 'name'), and the parent element is structured into one or more forms of that element, one of which is a default form, distinguished by the tag for the element wellKnown. The context of the element wellKnown is known from its parent.

RET.3.4.1.2.7 recordWrapper

This element is defined for use in presenting a record with no root (e.g. a flat record, or a record whose hierarchical structure is that of multiple trees). When the client requests this element, the request is interpreted as a request for the entire record to be presented subordinate to this element. It is defined primarily to be used in conjunction with a variantRequest specifying 'noData', for the purpose of retrieving a skeleton record (i.e. tags only, no data). If a record does have a root, then if this element occurs, the record's real root is presented subordinate to this element.

RET.3.4.1.3 Information about Result Set Record

TagSet-M elements rank and score provide information pertaining to a record's entry in the result Set. A record may have both a rank and a score. The rank of a result set record is an integer from 1 to N, where there are N entries in the result set (each record should have a unique rank). The score of a result set record is an integer from 1 to M where M is the normalization factor, which may be independent of the size of the result set, and more than one record may have the same score. The normalization factor should be specified in the schema.

RET.3.4.2 TagSet-G

TagSet-G includes generic elements which may be of general use for schema definitions. They are all self-explanatory, except perhaps the element displayObject (which was called ‘bodyOfDIsplay’ in Z39.50-1995).

RET.3.4.2.1 displayObject

The server might combine several elements of a record into this single element, into a display format, for the client to display to the user.

For a given schema, perhaps for a particular application, some clients may need the server to distinguish all elements in a retrieval record, perhaps because the client is going to replicate the record. In other cases, the client is satisfied for the server to package all elements into display format for direct display to the end-user. In either of these cases, displayObject is not applicable (in the latter case the server may use the SUTRS record syntax instead of GRS-1).

In some cases though, the client may need some of the elements distinguished, but is satisfied to have the server package the remaining elements into a single retrieval element for display. In these cases displayObject may be useful.

Suppose the server wishes to present 20 elements of a record, but only the first three elements are intended for client use, and the remaining elements are intended to be transparently passed to the user. Rather than packaging all 20 elements, the server instead may send 4 elements, where the 4th delivery element packages the latter 17 original elements, in a display format.

The displayObject element is similar to a composite element (as described in RET.3.1.2) in the fact that a single retrieval element packages multiple logical element. But displayObject differs from a composite element in three respects:

- The server, not the client, selects the subset of elements for packaging.
- In a composite element there may be semantics conveyed by the tag that the client or user might understand. For example a request for a composite element may ask for the b subfield of the 245 field concatenated with c subfield of 246 sent back as deliveryElement called 'title' (there may be some recognizable semantics associated with the tag 'title'). The displayObject element has no semantics other than telling the client "here is a composite element for display."
- The resultant element should always be in display format. A composite element may assume display format, but it may also assume other formats, as determined by the variant.

Appendix 14 NEGO: Z39.50 Negotiation Model

Normative

Negotiation between a Z39.50 client and server may be carried out during initialization of a z-association, via the InitRequest and InitResponse.

Negotiation of protocol version, message size, and options (via option bits) is supported by explicit parameters within the Init messages. Additional negotiation may be carried out via the use of the otherInfo parameter in the InitRequest and InitResponse (or by simulation of otherInfo using the userInformationField; see USR.2 "Use of Init Parameters for User Information").

This model pertains to the use of otherInfo in an InitRequest and InitResponse for purposes of negotiation.

NEGO.1 Negotiation Records

The otherInfo parameter is defined as:

```
OtherInformation   ::= [201] IMPLICIT SEQUENCE OF SEQUENCE{
      category   [1]   IMPLICIT InfoCategory OPTIONAL,
      information      CHOICE{
            characterInfo    [2] IMPLICIT InternationalString,
            binaryInfo       [3] IMPLICIT OCTET STRING,
            externallyDefinedInfo [4] IMPLICIT EXTERNAL,
            oid                    [5] IMPLICIT OBJECT IDENTIFIER}}
--
      InfoCategory ::= SEQUENCE{
      categoryTypeId    [1]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
      categoryValue     [2]   IMPLICIT INTEGER}
```

Thus it is one or more occurrences of the following structure:

```
  SEQUENCE{
      category  [1]   IMPLICIT InfoCategory OPTIONAL,
      information       CHOICE{
      characterInfo           [2] IMPLICIT InternationalString,
      binaryInfo              [3] IMPLICIT OCTET STRING,
      externallyDefinedInfo   [4] IMPLICIT EXTERNAL,
```

```
oid                          [5] IMPLICIT OBJECT IDENTIFIER}}
```

Refer to each instance of this structure as an **otherInfo unit**.

An otherInfo unit in an InitRequest or InitResponse is considered to be a **Negotiation Record**, if the following two conditions are met:

1. It conforms to the following sub-structure:

   ```
   SEQUENCE{
   externallyDefinedInfo [4] IMPLICIT EXTERNAL}
   ```

   That is, 'category' is omitted and the CHOICE for 'information' is 'externallyDefinedInfo' .

2. The definition explicitly states that it is to be used as a negotiation record as defined in this model.

The InitRequest and Response otherInfo parameter may include one or more otherInfo units, some of which may be negotiation records. This model describes the exchange of negotiation records only. OtherInfo units that are not negotiation records (as defined in this model), may be interspersed arbitrarily among the negotiation records without violating this model.

An example of a negotiation record is character set/language negotiation.

NEGO.2 Rules Pertaining to the Use of Negotiation Records

1. When the client includes a negotiation record in the InitRequest, if the server does not recognize the negotiation record type (i.e. its object identifier) it should ignore the record, and should not include a negotiation record of that type in the InitResponse.
2. A server should never include a negotiation record in an InitResponse unless the client has included a negotiation record of that type in the InitRequest. See, however, "Server-Mandated Negotiation" below.
3. When the client includes a negotiation record of a particular type in the InitRequest, negotiation (as defined in the definition of the negotiation record) is considered to be carried out if the server also includes a negotiation record of that type in the InitResponse. Note that "carried out" does not necessarily mean "successful".
4. If the client includes a negotiation record of a particular type in the InitRequest and the server does not include a corresponding negotiation record in the InitResponse, then no negotiation (as defined in the definition of the negotiation record) is assumed to take place; for practical purposes, the client may simply assume that the server does not recognize the negotiation record.
5. If the server includes an OtherInfo unit in the InitResponse that the client does not recognize, the client should ignore it.

NEGO.3 Server-Mandated Negotiation

This model does not support server-initiated negotiation. Thus, as stated in rule 2 above, the server should not supply in the InitResponse a negotiation record of a type that the client has not supplied in the request, because there is no way the server can determine whether the client even recognized the negotiation record.

However there may be instances where the server is not willing to enter into a z-association without certain negotiable rules established, and where the server cannot effect the necessary negotiation because the client has not supplied the appropriate negotiation record in the InitRequest. In this case, it is recommended that the server reject the z-association with diagnostic 1054: “required negotiation record not included” indicating the object identifier of the required negotiation record.

There may also be instances where the server cannot ascertain whether the client even supports this model. 5.2 below (Dynamic Adherence) addresses this.

NEGO.4 Adherence to this Model

NEGO.4.1 Static Adherence

A negotiation record definition conforms to this model if the two conditions listed in section 1 are met; that is: it conforms to the structure shown in point 1, and the definition explicitly states that it is to be used as a negotiation record as defined in this model.

NEGO.4.2 Dynamic Adherence

Z39.50 option bit 17 is assigned to correspond to this negotiation model. When the client sets this option bit, it signifies adherence to the model. If the client and server both set the option bit (in the InitRequest and Response respectively) both may assume that negotiation is carried out in accordance with this model. If the client sets this option bit and the server does not, the client should not assume that negotiation has been carried out in accordance with this model.

If the client does not set this option bit, but the server requires that negotiation be carried out in accordance with this model, the server may reject the z-association and supply diagnostic 1055: negotiation option required.

The reason an option bit is necessary is that a server might operate according to some other (perhaps implicit) model for information exchange during initialization. For example, suppose a server routinely echoes, in the InitResponse, all of the information supplied in the InitRequest. In the absence of an explicit mechanism to determine whether or not this model is in effect, the client may be falsely led to believe that negotiation has been carried out.

Appendix 15 NEGO2: Development and Registration of Negotiation Records

Non-normative

NEGO2.1 Negotiating Behavior

The negotiation model (see appendix NEGO) is intended to support negotiation of behavior elements, where a behavior element is identified by an object identifier that corresponds to a definition of the behavior. For example, a behavior element definition might state that its object identifier "may be used within a negotiation record such as BehaviorNegotiation-1 to negotiate the rules specified in section x.y.z of profile xyz" where definition BehaviorNegotiation-1 is a (hypothetical) negotiation record that might be defined as follows:

```
BehaviorNegotiation-1

{Z39-50-negotiationRecord negotiateBehaviorElements (x)} DEFINITIONS
::=

BEGIN

NegotiateBehaviorElements ::= CHOICE{

   proposal [1]   IMPLICIT SEQUENCE OF OBJECT IDENTIFIERS,

   response [2]   IMPLICIT SEQUENCE OF OBJECT IDENTIFIERS

        -- The server response must be a subset of the set proposed

        -- by the client. if the server requires a particular behavior

        -- element to be in effect that the client has not supplied,

      -- then the server should reject the Init, with bib-1 diagnostic

        -- 1054 indicating the oid(s) of the required

        -- behavior element.

                          }

END
```

NEGO2.1.1 Registration of Behavior Elements

The Z39.50 Maintenance Agency will not attempt to register all behavior elements. As alluded to in the example above, it is assumed that there will be behavior elements defined within profiles. The Maintenance Agency will assign an object identifier to a profile, upon request by the editor of (or individual or organization responsible for) the profile, who is then the registration authority for that object identifier and may assign object identifiers subordinate to that object identifier, defining behavior elements. Those definitions might refer to sections in the profile that describe levels of conformance or functional units for the profile.

If there is a need to define behavior elements outside of profiles, they will be registered by the Z39.50 Maintenance Agency (or they may be registered privately by implementors). For example, if there is a need to negotiate that a particular Implementor Agreement be in effect, it will be assigned an object identifier.

NEGO2.1.2 Negotiating Support

The following hypothetical example of a negotiation record illustrates negotiation of support for specific functionality.

```
SupportNegotiation-1

{Z39-50-negotiationRecord negotiateBehaviorElements (y)} DEFINITIONS
::=

BEGIN

NegotiateSupport ::= CHOICE{

   proposal [1]   IMPLICIT SEQUENCE OF DatabaseTriple,

   response [2]   IMPLICIT SEQUENCE OF DatabaseTriple}

-- Client proposes a set of triples, and server responds with

-- a set of triples that will be supported. Server set

-- need not be a subset of the client set, nor need it be

-- exhaustive (absence of a database, or an attribute set id

-- or record syntax for a given database, does not necessarily

-- mean that it will not be supported).


DatabaseTriple ::=  SEQUENCE OF SEQUENCE{

           databaseName    [1] IMPLICIT InternationalString,

           attributeSets   [2] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER,

           recordSyntaxes  [3] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER}

END
```

In this hypothetical example the client proposes a set of databases available for searching, and for each, a list of proposed attribute set ids and a list of proposed record syntaxes. The server responds with a list of databases, and for each, a list of attribute set ids and a list of record syntaxes that will be supported.

In this particular example the server list may or may not be a subset of the client list. Alternatively, a different negotiation definition might mandate that the server explicitly respond -- for each database, attribute set, and record syntax -- whether or not it will be supported.

Note that specific behavior is not negotiated by this (example) negotiation record. If support, for example, for a particular record syntax (for a database) is negotiated, that does not mean that the server will support that syntax for every record (in the database).

Appendix 16 PRO: Z39.50 Profiles

Non-normative

PRO.1 Introduction

The Z39.50 standard defines a range of services useful in information retrieval applications. For each of the services, the standard provides choices and options for parameters in individual protocol messages. There are many objects used in conjunction with the standard (e.g., attribute sets, record syntaxes, etc.). The result is a comprehensive information retrieval protocol with built in flexibility to allow implementors to choose selected services, parameters, and objects for specific applications. In general an implementation does not support the complete standard, but rather a conforming subset corresponding to specific relevant requirements. As a consequence, interoperability between implementations is not always optimal.

To guide or prescribe the use of the Z39.50 standard in applications and to improve interoperability, implementor groups define profiles. Profiles define a subset of specifications from one or more standards (e.g., selected services and required values for specific parameters) and associated objects to be used in specific applications. The overall goal of profiles is to improve interoperability between systems conforming to a specific profile. The implication is that an implementor does not "implement the standard" but rather, configures a Z39.50 client and/or Z39.50 server to conform to one or more profiles.

PRO.2 Profiles Respond to Community Needs

Motivations for creating a profile include:

- To introduce and prescribe how Z39.50 should be used in a particular application environment.
- To solve interoperability problems with existing Z39.50 implementations within a community (e.g., libraries) or across two or more communities (e.g., library and museums).
- To provide a specification for vendors to build to, so the resulting products will interoperate.
- To provide a specification that customers may reference for procurement.

The first Z39.50 profiles emerged in the early to mid–1990s (e.g., the GILS Profile, the WAIS Profile, the CIMI Profile). These profiles served to introduce how Z39.50 could be used in specific application environments (e.g., a government information locator application, network publishing systems, the cultural heritage museum environment). More recently, major profiling efforts have focused on solving interoperability problems in library applications (i.e., the Bath Profile, the U.S. National Z39.50 Profile for library applications). In the latter case, Z39.50 is already installed in the community but interoperability has not been optimal because common agreements on configuring Z39.50 clients and servers were absent.

Groups have used several approaches in developing profiles. A common first step is to define the requirements for the application. Building consensus on what requirements must be supported by the Z39.50 standard is critical. Once the requirements are identified the next step in the profile development is to specify the details of the Z39.50 standard (and/or other standards) to support the requirements. Thus, profiles can be characterized as a response to community needs; they provide a solution path towards improved interoperability in specific applications.

When a profile is completed, customers can use the profile to aid in purchasing decisions. For example, individual library managers can reference a profile in a Request for Proposal. This

saves the manager from having to specify the details of Z39.50 in its procurement. A profile provides the details necessary for developers and vendors to build and configure Z39.50 clients and servers.

PRO.3 Applications Addressed By Profiles

The Z39.50 Maintenance Agency monitors profile development and maintains a list of the profiles. The following list illustrates the range of profiles that have been developed in response to application and community needs (see the complete list at http://lcweb.loc.gov/z3950/agency/profiles/profiles):

- **The Bath Profile: An International Z39.50 Specification for Library Applications and Resource Discovery.** Addresses a core set of specifications for search and retrieval across online library catalogs, holdings information, and cross–domain resource discovery. A number of regional and national profiles use the Bath Profile specifications as a foundation including:
  - The U.S. National Z39.50 Profile for Library Applications
  - The ONE-2 Profile
  - The DanZIG Profile.
- **The Union Catalogue Profile.** Allows a database creator to update a database in a distributed environment.
- **The Government Information Locator Service (GILS) Profile**. Addresses the search and retrieval of government information resources. Other profiles have extended the GILS Profile including:
  - Z39.50 Application Profile for Geospatial Metadata (Geo Profile).
- **The Z39.50 Application Profile for Cultural Heritage Information (CIMI Profile).** Addresses search and retrieval within museum and cultural heritage information applications.
- **Zthes: a Z39.50 Profile for Thesaurus Navigation.** Describes an abstract model for representing and searching thesauri.
- **Z39.50 Profile for Access to Digital Collections.** Provides access to digital collections organized via descriptive information and semantics for navigating digital collections to locate and retrieve objects of interest.

The common element in these profiles is the specification of a subset of the Z39.50 standard. In many cases, profiles extend the utility of the standard by developing abstract models for an application (e.g., the Zthes and Digital Collections Profiles), and developing schemas, element sets, and abstract record structures (e.g., the CIMI and GILS Profiles). In other cases, the profiles simply prescribe choices for options in the standard and the use of associated existing objects (e.g., the Bath Profile and the Union Catalogue Profile).

PRO. 4 Development and Approval of Profiles

The Z39.50 Maintenance Agency does not develop profiles but may work with groups who are developing profiles. The Maintenance Agency publishes procedures related to the profile development and approval (see the page at: http://lcweb.loc.gov/z3950/agency/proced.html#profiles). A group of people or a formal standards body may develop a profile and request the Maintenance Agency to list it.

The international Z39.50 Implementors Group (ZIG) does not develop profiles but may work with groups who are developing profiles. Profile developers are encouraged to submit drafts of the

profile for review and comment by the ZIG, and profile developers can request that the ZIG endorse the profile.

A profile developer may submit a profile to ISO TC46 for review and approval as an Internationally Registered Profile (IRP). (See the procedures for this route of profile approval at: http://lcweb.loc.gov/z3950/agency/profiles/irp.html.). The review leading to approval as an IRP is for technical conformance to the relevant standard(s).

Occasionally, a formal standards body may develop a profile and approve it as a national or international standard. Such development follows the procedures of the standards body. Once approved as a standard, the profile developer can request that the Z39.50 Maintenance Agency list the profile.

PRO. 5 Examples of Profiling Z39.50 Standard Services and Specifications

To illustrate how profiles can specify the use of Z39.50 for particular application, this section provides examples of what may be included in profiles. This is for illustration only and does prescribe what must be included or addressed by a profile.

PRO.5.1 Protocol Version and Services

To ensure that a Z-client and Z-server support a common protocol version (e.g., Version 2 or Version 3), a profile may require a specific version of the protocol.

The Z39.50 standard offers numerous protocol services for information retrieval. Z-clients and Z-servers must support some common set of services if interoperability at the service level is to occur. Typically, implementations will support Init, Search, and Present services. A profile can explicitly name these and other services that both Z-clients and Z-servers must support. It is especially important to specify in the profile any additional services that are required for an application (e.g., Sort, Scan, etc.).

PRO.5.2 Z39.50 Objects

The Z39.50 Maintenance Agency registers objects that may be used in applications. A complete list of object classes and objects is available at: http://lcweb.loc.gov/z3950/agency/defns/oids.html (object classes defined at the time of publication of this standard are listed in appendix OID). The Maintenance Agency assigns each object a unique identification number (i.e., Object Identifier or OID). Profiles may list all the objects and their OIDs used in an application. For example, a profile may list the attribute sets, record syntaxes, schemas, and other objects.

PRO.5.3 Specifying the Use of Z39.50 Objects

In some cases, a profile can simply indicate the Z39.50 objects that Z-clients and Z-servers must support. In other cases, additional specification of the use of a Z39.50 object is warranted. For example, a profile may require that Z-clients and Z-serves support a particular record syntax. In the case of one of the registered MARC record syntaxes, no additional specification of the use of that syntax may be required. However, if a profile requires the use of the GRS-1 or XML record syntaxes, further specification is necessary. The profile may designate (or may create and request the Z39.50 Maintenance Agency to register) a schema and tagset to be used in conjunction with an application's implementation of GRS-1. In the case of XML, reference to a document type definition or XML schema is necessary.

Requiring Z-clients and Z-servers to support one or more Z39.50 attribute sets usually implies a further specification of what supporting an attribute set means. For example, the Bib-1 attribute set contains hundreds of use attributes, and it is unlikely that Z-clients and Z-servers are configured to process all use attributes. A profile can identify the specific attribute types, selected attribute values, and appropriate combinations that Z-clients and Z-servers must support. Such specification does not preclude Z-clients and Z-servers from implementing other attribute sets, attribute types, values, and combinations not listed in a profile.

PRO.5.4 Referencing Amendments, Implementor Agreements, and Clarifications

Clarifications, commentaries, .amendments, implementor agreements, and defect reports can be issued after the publication of the Z39.50 standard. The Z39.50 Maintenance Agency has procedures for developing these types of changes and maintains them at: http://lcweb.loc.gov/z3950/agency/related.html. During the revision of the Z39.50 standard, some or all of these changes and agreements may be integrated into the revised standard. Prior to their appearance in a revised standard, a profile may reference these changes and agreements.

For example, the Maintenance Agency approved in 1999 an amendment that defines encapsulation of Z39.50 APDUs. (Encapsulation is a Z39.50 feature that allows the Z-client to group together several APDUs and the Z-server to similarly group the response APDUs in order to carry out multiple Z39.50 operations in a single transaction.) Encapsulation is an amendment to Z39.50-1995 but is part of the Z39.50-2003 standard. Thus a profile that refers to Z39.50-1995 as the base standard may refer to the amendment, while a profile that refers to Z39.50-2003 may refer to the encapsulation feature as specified within the standard. (In any case, as with other Z39.50 specifications, simply requiring support for encapsulation may not be sufficient, and a profile that specifies encapsulation may provide guidance regarding the intent of the usage of encapsulation.)

PRO.6 Negotiation

When a Z-client's requirements are represented by a specific profile, then the Z-client might determine if a particular Z-server supports its requirements by ascertaining whether the Z-server supports that profile. The Z-server administrator might advertise support for the profile, or the Z-client and Z-server administrators might exchange information about what profiles are supported. However, this approach has a number of limitations.

- It requires out-of-band communication.
- The Z-client may wish to determine Z-server capabilities with finer granularity than at the profile level. A profile might specify a number of conformance levels, where conformance-at-large to the profile does not necessarily require conformance at all levels. A statement by a Z-server that it supports the profile cannot necessarily be construed to mean that it conforms at all levels.
- Discovering that a Z-server implements certain capabilities may not be sufficient: the Z-client might want to ascertain that the Z-server is actually willing to provide these capabilities. For example, suppose a Z-client determines that a Z-server supports sorting a result set (on a specific key). The Z-client might submit a complex (and costly) query, then request that the result set be sorted, and the Z-server responds that the Z-client is not authorized to sort result sets (or that the sort facility is currently unavailable). The Z-client has already incurred the cost of the query and would not have done so if it had known it could not sort the results.
- A Z-client may be interested in a certain set of capabilities, however there isn't any (known) profile for which support of that particular set alone constitutes conformance to the profile. For example, suppose the Z-client wants feature A from Profile A and feature B from profile B. There may be a Z-server that supports features A and B but which does not support either

profile, A or B.

For these reasons, Z39.50 provides a mechanism for a Z-client and Z-server to negotiate functionality for a specific session. This standard provides a model for negotiation between a Z39.50 Z-client and Z-server (carried out during initialization of a z-association via the InitRequest and InitResponse). See Appendix NEGO.

The negotiation model is intended to support negotiation of behavior elements. A behavior element is identified by an object identifier corresponding to a definition of the behavior. Negotiation of behavior elements is carried out via the exchange of negotiation records. The development and registration of negotiation records is described in Appendix NEGO2.

Profile developers are encouraged to take advantage of this negotiation facility. A profile might include a section which defines or refers to behavior elements corresponding to requirements and features specified by the profile, as well as to negotiation records. It should describe how negotiation is to be carried out, in terms of the behavior elements and negotiation records.

A behavior element might group several features together at the discretion of the profile developer. Each behavior element should be assigned an object identifier. The Maintenance Agency will assign a single object identifier to a profile, upon request of the party responsible for the profile, who is then the registration authority for that object identifier and may assign object identifiers subordinate to that object identifier, defining behavior elements. Alternatively, a behavior element may already be defined (for the benefit of a different profile, or for general use, by the Maintenance Agency) that meets the need of the given profile, in which case the profile may simply refer to that behavior element.

PRO.7 Summary

Profiles respond to the needs of a community or application. They may vary in their structure and contents. But they have common goals:

- To indicate how Z39.50 should be implemented and how Z-clients and Z-servers should be configured to support the needs of a community or application
- To improve interoperability between Z-clients and Z-servers within a community or for an application.

A well-developed profile will likely have the following characteristics:

- Developed by a group that reflects stakeholders of a community or an application
- Clear statement of purpose, scope, area of application
- Detailed indications of Z39.50 specifications used
- Mechanisms for public review and comment
- Procedures for maintaining the profile once approved by the developing group, as an Internationally Register Profile, or as a formal standard.

Because of the Z39.50 standard's rich functionality and options, profiles provide a necessary level of specification to achieve interoperability.

Profile developers are encouraged to take advantage of the 39.50 negotiation facility, which includes a model for negotiation of behavior elements between Z-clients and Z-servers.

Appendix 17 Z39.50 Attribute Architecture

Non-normative

ARCH 1 Introduction and Preliminary Notes

ARCH 1.1 Historical Background

The initial attributes for the bib-1 attribute set were developed by representatives of the Library of Congress, RLG, OCLC and WLN in the mid-1980s. This U.S. set was merged with a similar set from European library system developers to become bib-1. It was the only attribute set definition included in the published version of Z39.50-1992 (version 2).

Problems with the bib-1 attribute set began to surface at that time (1992). Within the bibliographic community, implementors had no published definitions of the bib-1 attribute semantics, thus vendors implemented the bib-1 attribute set with their own interpretations of the attribute usage. A document was produced to clarify this (Bib-1 Semantics Document), although it was never formally included as part of the standard.

As the Internet grew, more communities wanted to implement Z39.50 and, in turn, needed additional attributes (beyond those already in bib-1) to reflect the types of data they wanted to exchange. This proved difficult as Z39.50-1992 did not allow a query to include attributes from more than a single attribute set. Since bib-1 was the only publicly visible set, it was expanded to accommodate the needs of these communities. Thus, bib-1 grew without plan or rigor, evolving away from the bibliographic community where it had started, and "bib-1" became somewhat of a misnomer as it grew into a global set of attributes.

In 1994 and 1995, as Z39.50-1995 was being finalized and as Z39.50 began to be widely implemented, additional concerns arose over the relationships among attribute sets that other groups were developing, notably the STAS and GILS attribute sets. The Z39.50 Implementors Group (ZIG) had many questions about the development and implementation of multiple attribute sets, including duplication of attributes across sets. In early 1996 a discussion paper detailed the issues:

1. Duplication of common attributes in specialized attribute sets, due to the limitations of the Type-1 query imposed by version 2 of the protocol.
2. Interoperability problems due to attribute set proliferation, for example, how to know which basic attributes were imbedded in specialized sets.
3. Ambiguities in the semantics of attributes.
4. Lack of rigorous semantics in the bib-1 attribute set; lack of a scope statement for the bib-1 attribute set; lack of consultation with the broad community concerned with bibliographic records.
5. Lack of guidance about the semantics of mixing attributes from different attribute sets in a single Z39.50 query (and in particular, in a single query operand).

An informal committee formed to recommend resolutions of the issues met several times, preparing interim reports discussed at subsequent ZIG meetings. The final report of the group was presented at the January 1998 ZIG meeting. The major conclusion of the group was that a new architecture for attribute sets should be developed; they went on to recommend an architecture based on classes of attribute sets, with expanded attribute types. Another major conclusion was that expert communities, rather than the ZIG, should be responsible for developing and maintaining attribute sets (following the example set by GILS and STAS). Notably, they recommended that the bibliographic community, rather than the ZIG, develop the next generation of bibliographic attributes. The ZIG should continue to be responsible for attributes that are general to Z39.50, that is, not specific to a given community.

ARCH 1.2 Brief Technical Background

Z39.50 defines a number of query types, and requires support for the **type-1 query** (support for other defined query types is optional). This document addresses the Z39.50 type-1 query only.

The type-1 query consists of one or more search terms, each with a set of attributes, specifying, for example, the type of term (author, title, subject, etc.), whether the term is truncated, its structure, etc. The server is responsible for mapping attributes to the logical design of the database.

A term in a type-1 query, together with its accompanying collection of attributes, is called an ***operand***. Operands may be combined in a type-1 query, linked by boolean operators (And, Or, And-not, and Proximity).

Each attribute is a pair: an **attribute type** and a value of that type. An **Attribute set** defines a set of attribute types, and for each type defines the set of possible values.

An attribute set definition is assigned an object identifier, referred to as its **attribute set identifier**.

> **Example**: The ***bib-1 attribute set*** defines a number of ***attribute type***s; one of which is ***Use***. For bib-1 Use attributes, many ***attribute value***s are defined, one of which is ***personal name***. Each type is assigned a numeric value, and each value is assigned a numeric value: type ***Use*** is assigned the value 1, and Use attribute ***Personal Name*** is assigned the value 1. Thus bib-1 Use attribute Personal Name is represented as the pair (1,1). This pair is further qualified by the bib-1 attribute set identifier (1.2.840.10003.3.1) to distinguish it from the pair (1,1) that may be defined by another attribute set.

Version 2 of Z39.50 has two serious limitations inhibiting the development of attribute architecture, both corrected in version 3:

In version 2, all attributes within a query must belong to the same attribute set (the query accommodates only a single, global attribute set id). In version 3, attributes may be combined from different attribute sets, within a single query, even within a single operand (an attribute set id may accompany every attribute). This is a significant enhancement, for two reasons: First, it is useful when searching multiple databases. (Although version 2 supports multi-database searching, all attributes within a query must belong to a single attribute set, which inhibits the

ability to search multiple, heterogeneous databases.) Second, new attribute sets may be defined with less replication.

Version 2 allows only a single ASN.1 representation for search terms, namely ASN.1 type OCTET STRING. In version 3, new data types for terms are defined, for example, integer and character string.

ARCH 1.3 Limitations and Restrictions

ARCH 1.3.1 Version 3 Assumption

There are several enhancements in version 3 pertaining to attribute sets and query construction; the two enhancements described at the end of 18.2 are certainly the most important, and are seen to be functional prerequisites for the development of an attribute architecture. For this reason, version 3 is assumed by this architecture, and version 2 is not addressed.

ARCH 1.3.2 Type-1 Query Limitation

The Z39.50 type-1 query has known limitations, and the architecture specified in this document is restricted by these limitations. As the standard evolves and new versions are approved, the architecture may be expanded. See section 4: "Lessons Learned: Recommendations for Future Enhancements to the Z39.50 Query".

ARCH 1.3.3 Semantic Indicator

In order to compensate for some of the type-1 limitations, it may be necessary to utilize the semantic indicator (provided within version 3) for purposes that would otherwise be accomplished by more coherent mechanisms if these limitations were not present. It is intended that these limitations will be addressed in future versions of Z39.50, obviating the need for extensive use of the semantic indicator at the attribute level.

ARCH 2. Attribute Set Class Definitions

The attribute architecture allows definition of multiple attribute set classes. An attribute set class provides an umbrella context for the definition of an attribute set belonging to a particular class. It defines attribute types that may be included in an attribute set for that class. Attribute set Class 1 is defined as part of this architecture document (Section 3).

This architecture strongly recommends that an attribute set definition that conforms to a particular class but defines attribute types that are not defined for that class should carefully define the interactions between the new attribute types and existing types defined for that class.

The architecture provides the attribute-set-class approach to allow flexibility and future expansion within the existing architecture. It is believed that attribute set Class 1 meets all known needs for an attribute class at this time. There may be other approaches developed which partition the set of attributes into fundamentally different types. This might result in the definition of a new attribute class inconsistent with Class 1. However, no need for such a separate class has been identified and it is not known whether additional classes will be necessary.

ARCH 2.1 Attribute Values

These rules for construction of attribute values pertain to all classes.

An Attribute set may define the set of values for a particular attribute type as follows:

1. The attribute set definition may supply a finite list (where individual members of the list may be numbers or character strings).
2. The attribute set definition may define the type as numeric. For example, the value of an 'occurrence' attribute may simply be the actual occurrence, that is, to indicate "second occurrence of field N" the value of the Occurrence attribute would be 2.
3. The attribute set definition may specify that a locally defined value, either a number or string, may be used as the value of the attribute for that type.
4. The attribute set definition may specify that the attribute may take on a sequence of values, where each is any of the above (1, 2, or 3).

An Attribute value in an operand may thus be a number, string, or sequence of numbers and strings. A number value might take the role of 1 or 2 above, and a string value might take the role of 1 or 3; in each case, the role is interpreted by the attribute set definition.

ARCH 3. Attribute Set Class 1

Class 1 is intended to cover all known, existing requirements, at the time that this attribute architecture was developed. (Existing attribute sets may need to be re-specified within this framework.)

The purpose of enumerating all of the possible attribute types within this "universal" attribute class is to provide a template for developers of attribute sets, and to set up a framework for interoperability among independently defined attribute sets which are intended to serve various communities. In particular, it should be possible for groups of content experts to develop new Access Point attributes, ASN.1 datatypes, comparison operators, and perhaps Format/Structure attributes which fit comfortably within this framework. Based on the template defined here, server developers may recognize attribute types omitted in a query operand, as well as illegal repetitions or combinations of attributes of given types that would indicate a malformed query operand.

ARCH 3.1 General Rules for Class 1

ARCH 3.1.1 Semantic Precedence and Interaction among Sets

The context of this attribute class is in effect for a query when the OID of an attribute set conformant with Class 1 specified as the global OID (the object identifier within the type-1 query that does not accompany a specific attribute). For Class 1, the global OID is referred to as the ***dominant OID*** for the query. When attributes from different attribute sets are mixed within a query, and when the respective attribute set definitions conflict such that the resulting semantics are ambiguous, the semantics of the dominant set prevail. As an example, suppose attribute set definition A declares that the Language type is mandatory in an attribute list, while attribute set definition B declares it to be optional. If attribute set A is used as the dominant set for a query, then the Language attribute would have to be supplied within every operand; if attribute set B is the dominant set, it would not.

When an attribute set is developed in accordance with Class 1, its definition should state that it is a Class 1 attribute set. In addition, the definition may describe the rules that apply (when it is used as the dominant set) for intermixing of attributes from different sets within an operand or query. Attributes from attribute sets conformant to Class 1 should not be mixed, within a query operand, together with attributes from historical attribute sets defined prior to the development of this attribute architecture (e.g. bib-1).

ARCH 3.1.2 Populating Class 1 Attribute Sets

An attribute set consistent with this attribute class will define attributes of one or more of the types specified in section 3.2.

Any Class 1 attribute set follows the rules prescribed for Class 1 that apply to attribute types defined for that set. However, a Class 1 attribute set need not define nor populate every attribute type defined for Class 1. A Class 1 attribute set may define as few as one attribute type, or as many as all of the attribute types defined for Class 1.

Thus no specific attribute type is ***mandatory*** in the sense that it must be included in an attribute set definition. (This use of the term ***mandatory*** is different from the use of ***mandatory*** to mean that a particular attribute type may not be omitted in an operand, as used in the Occurrence column of the table in section 3.3. For example, the Comparison attribute type is mandatory in an operand.)

However, a Class 1 attribute set must use the numeric values in the "Type Number" column in the table in section 3.3 to represent the types; if any of these types is omitted in the attribute set definition, the definition should skip the value for that type rather than renumber.

An attribute set might be developed for an application or profile and may refer to values of a particular attribute type that are defined by a different attribute set. If all of the values of that type that are required by the application are already defined by that other attribute set, then that attribute type need not be defined for the new set.

There may often be a close relationship between the development of a profile for a particular application, and the development of an attribute set definition to support the application. The profile might refer to several attribute sets in describing how to construct query operands (or entire queries). Thus the attribute set definition is not, itself, responsible for specifying all of the details of searching for the application when those details involve attributes from different attribute sets; however, the attribute set may offer as much commentary as it deems necessary and appropriate, for example, it may explain why a particular attribute has been omitted from its definition (for example, because another attribute set has defined it). It might explain how certain attributes that are defined in the set are to be combined with attributes from other sets.

ARCH 3.1.3 Omitted Attributes

An attribute set definition should not specify a default value for an attribute type to be applied when that attribute type is omitted from an operand. Each individual server may determine the semantics of omitted attributes. Thus when a client omits an attribute of a given type from an operand (unless that type is not applicable for the given attribute combination, or unless the attribute type is mandatory) the client is, in effect, leaving it to the server to select a value. See also section 3.2.1.3, "Omitted Attributes in Conjunction with Nested Access Point Attributes".

The presence or absence of any attribute should not imply the presence of any other attribute, whether of the same or a different type. (For example, the presence of an Access Point attribute

should not imply the presence of an otherwise omitted Format/Structure attribute, even if the relationship seems obvious.)

ARCH 3.1.4 Syntactic Content of Search Term

A query operand should be constructed such that the server may determine the syntactic content of the search term based on the ASN.1 datatype of the term as well as the Format/Structure attribute, if supplied (and if the Format/Structure attribute is not supplied, by the ASN.1 datatype alone). In general, the value of the Access Point attribute should not contribute to this determination.

Even in cases where there is only one legal value of a Format/Structure attribute, and when the client might expect the server to deduce that value, it should be explicitly supplied. An exception is when the ASN.1 type completely and unambiguously determines the format, for example when the ASN.1 type is INTEGER or GeneralizedTime, or when the Z39.50 Date/Time definition is supplied (as EXTERNAL); in these cases the Format/Structure attribute may be omitted. ASN.1 type InternationalString does not unambiguously determine the format.

An attribute set developer should determine all of the Format/Structure attribute values necessary to fully specify the term formats relevant to the attribute set, and for each, either include it as a Format/Structure value in the attribute set definition, or ensure that it is defined in another attribute set (and provide appropriate reference within the attribute set definition).

ARCH 3.1.5 Repeatability

In general, if an attribute type is allowed to be repeatable within an operand, the semantics of repeating the attribute type must be well-defined.

While repeatability may be permissible for a given attribute type, as a general principle, an attribute type should not be repeated as a substitute for Boolean operations. To amplify this point, an attribute definition might prescribe how to interpret, for example, multiple Access Point attributes in a single operand. The definition might prescribe (as examples):

- Multiple Access Point attributes may be supplied in order of preference, so if a server does not support the first supplied, then use the second, etc.; or
- if multiple Access Point attributes are supplied, the server is to choose the "best" among the set; or
- if multiple access point attributes are supplied, they are to be treated as nested access points (see Access Point Attribute Type).

The above three examples are for illustration only. There may be other possible interpretations for multiple Access Point attributes.

The definition may include a semantic indicator, allowing a client to select among several semantic alternatives. However, none of those alternatives should be to construct separate operands (linked by boolean 'and' or 'or') for each Access Point attribute -- the type-1 query supports boolean operations, so allowing another means of specifying boolean operations would add unnecessary complexity (in contrast to potential semantic interpretations of multiple Access Point attributes which cannot be otherwise represented via the type-1 query, as in the examples above).

ARCH 3.1.5.1 Mechanism for Repeating Attributes

There are two mechanisms supplied by the Z39.50 standard for providing multiple attributes of the same type within an operand:

1. Via 'list' within 'complex' CHOICE of 'attributeValue' within AttributeElement; defined in section 4.1 of Z39.50 Abstract Syntax and ASN.1 Specification of Z39.50 APDUs. (This mechanism is provided by version 3, and not supported in version 2.)
2. Via separate instances of AttributeElement.

Although Z39.50 provides both of these mechanisms, the first mechanism is prescribed for Class 1.

ARCH 3.2 Attribute Types Defined within the Attribute Class

ARCH 3.2.1 Access Point Attribute Type

The Access Point attribute defines either an intellectual access point (for applications that work with abstract database definitions) or an access point corresponding to a database fieldname (for applications where searching is defined in conjunction with a specific database schema, or defined to correspond to a specific Z39.50 tag set).

The presence of an Access Point attribute is mandatory in an operand.

An attribute set definition that defines this type should include a discrete list of values.

ARCH 3.2.1.1 Nesting and Anchoring of Access Point Attributes

Nesting of Access Point attributes may be supported by an attribute set definition. If so, nesting should be indicated by repetition of the Access Point attribute type (as prescribed in 3.1.5.1), where the order of nesting is as in the following example: field 1, field 2, and field 3, supplied in that order, means "field 3 within field 2 within field 1". An example of the use of nesting might be a field path within an SGML database.

An Access Point attribute may be indicated as **not anchored** (matching may occur beginning at any node within the element tree) by nesting it within an Access Point attribute of value 'wildpath' (for example as defined in the Utility attribute set). In the absence of a wildpath attribute, it is considered **anchored** (matching must occur from the root of the element tree).

Example of Anchored vs. Not Anchored:

Suppose a schema includes elements Description, Contact, and Availability, where Description is unstructured (has no sub-elements), Contact is structured into sub-elements Name, eMail, and Description, and Availability is structured into sub-elements, one of which is Contact, similarly structured (leaf elements shown in bold):

Description
Contact
Name
Email
Description

Availability
Contact
Name
Email
Description

When the single Access Point attribute Description is specified as anchored, then it is intended to match first-level element Description; if multiple Access Point attributes Contact and Description are specified as anchored, then it is intended to match Description within first-level element Contact. If Contact and Description are specified as not anchored, then it may match Description within first-level element Contact, or Description within Contact within Availability. If the single Access Point attribute Description is specified as not anchored, then it may match first-level element Description, Description within first-level element Contact, or Description within Contact within Availability.

ARCH 3.2.1.2 Mixing Access Point Attributes from Multiple Attribute Sets

Mixing Access Point attributes from multiple attribute sets, within an operand, is permissible. Attribute sets might be defined that correspond directly to tag sets (which define Z39.50 retrieval elements). A search field might be defined corresponding to an element path defined by a retrieval schema. A type-1 query operand might correspondingly be constructed with nested Access Point attributes corresponding to the elements in the tag path for the field. Those elements may be from different tag sets, where the different tag sets correspond to different attribute sets. Correspondingly, the Access Point attributes would belong to different attribute sets.

ARCH 3.2.1.3 Omitted Attributes in Conjunction with Nested Access Point Attributes

When an attribute type is omitted, and when nested access points are specified (via multiple Access Point attributes values), the server will choose values for the omitted type based on the most specific access point in the list. For example, when searching field-1 within field-2, and the language attribute is omitted and the server must then select one, it should select it based on field 1, not field 2.

ARCH 3.2.2 Qualifying Attribute Types

- **Semantic Qualifier** Attribute Type

  One or more Semantic Qualifier attributes may be included in a query operand. The server is to pair each supplied value with the Access Point attribute to try to find a best match with its indexes.

  When the operand includes nested access points, each semantic qualifier value applies to the entire access point specification, that is, to the set of nested Access Point attributes.

  The client may indicate that the server may ignore the Semantic Qualifier(s) by including a null Semantic Qualifier (see Utility attribute set) thus allowing the server to match the Access Point attribute value with null, in effect rendering it unqualified. The Semantic Qualifier attributes are in no sense combined among themselves. They are not presented as a list of increasingly precise qualifiers. An attribute set definition that defines this type should include a discrete list of values. This attribute is

repeatable.

The Semantic Qualifier attribute is distinguished from the Functional Qualifier attribute, and the distinction is described below.

- **Functional Qualifier** Attribute Type

  One or more Functional Qualifier attributes may be included in a query operand. The mechanical aspects of the usage of the Funtional Qualifier type are the same as those of the Semantic Qualifier type: the server is to pair each supplied value with the Access Point attribute to try to find a best match; when the operand includes nested access points, each functional qualifier value applies to the entire access point specification; the Null value (from the Utility set) may be included to indicate that the server may ignore the functional qualifier(s); an attribute set definition that defines this type should include a discrete list of values; the attribute is repeatable.

  The Semantic Qualifier and Functional Qualifer types correspond to "type" and "role" respectively. The Semantic Qualifier describes the term itself, while the Functional Qualifier describes the relationship of the term to the object being searched. For example, consider a search on an author, where the author is a person and thus the term is a personal name. The Access Point attribute value would be 'name', the Semantic Qualifier value would be 'personal name' and the Functional Qualifier value would be 'author'.

  When a qualifier (semantic or functional) is to be defined and it is unclear whether it should be a Semantic Qualifier or Functional Qualifier, it is recommended that it be defined as a Semantic Qualifier.

- **Language** Attribute Type

  The value of this attribute indicates the language of the supplied term. An attribute set definition that defines this type should either include a discrete list of values, derived from some standard source, or else refer to some standard source for values. In the interests of simplicity it is recommended that this attribute be non-repeatable, though there may be situations where repeatable Language values could be meaningfully interpreted.

- **Content Authority** Attribute Type

  The value of this attribute indicates the source of the supplied term. An attribute set definition that defines this type should include a discrete list of values. In the interests of simplicity it is recommended that it be non-repeatable, though there may be situations where repeatable content authority could be meaningfully interpreted.

- **Expansion/Interpretation** Attribute Type

  This attribute may be used to indicate, for example, that thesaural expansion, singular/plural matching, part of speech qualification, phonetic matching, case sensitivity, stemming, truncation (including left and/or right anchored as well as word-by-word truncation), or various loose forms of phrase matching, should be used in the query evaluation.

  Server preprocessing instructions should be included in this type, for

example, "do not treat any words in this term as a stopword" and "do not remove punctuation".

An attribute set definition that defines this type should include a discrete list of values. This attribute is repeatable.

ARCH 3.2.3 Query Management Attribute Types

These attributes have the property that they can be rewritten by the server as part of a revised query that the server returns to the client.

- **Normalized Weight** Attribute Type

  The value of this attribute is the weight of the operand (in a weighted boolean query). An attribute set definition that includes this type should specify a normalization value (e.g. 1000). This is a non-repeating, numeric attribute.

- **Hit Count** Attribute Type

  The value of this attribute is the number of records satisfying the operand. This attribute is intended to convey information from server to client, but it may be passed back from client to server when the client simply wants to turn around a reformulated search -- in that case, it is to be ignored by the server. This is a non-repeating, numeric attribute.

ARCH 3.2.4 Comparison Attribute Type

The Comparison attribute defines the relationship between the term in the operand and the term in the term list at the server.

The presence of a Comparison attribute is mandatory in an operand, as it is presumed that there is always a relationship between the term and the value of the access point to which the term is compared (otherwise there would be no basis for comparison) and that the client knows the relationship; therefore, based on the principle stated in section 3.1.3, Omitted Attributes, the client should always supply the relationship.

The Comparison attribute is a generalization of the bib-1 Relation attribute, though named differently to avoid confusion. (The bib-1 Relation attribute is not mandatory in bib-1, as bib-1 has no such rules of occurrence, nevertheless, there is always a relationship, implied or explicit. One of the problems with bib-1, that Class 1 tries to correct, is the potential ambiguity when the relationship is not supplied.)

An attribute set definition that defines this type should include a discrete list of values. This attribute is non-repeatable.

Sample values might include:

- complete match
- does not match
- contains
- contained in bounding-polygon
- match via regular expression
- relevance feedback

- equal, not equal, greater than, etc.
- between (range operations in conjunction with a range datatype)

ARCH 3.2.5 Format/Structure Attribute Type

This attribute is used primarily to help with the interpretation of a character-string term; it provides guidance for the datatype conversion process.

Developers of specific Access Point attributes should consider defining (or utilizing existing) ASN.1 datatypes to support their applications -- for example, personal names, dates, geospatial information (points and polygons). There will of course be cases where the ASN.1 approach to datatyping will be too heavy-weight; in those cases the Format/Structure attribute type can be used in conjunction with ASN.1 type InternationalString to indicate that the content of a string represents data in a specific format. However, a character string term should not be used to represent an integer (e.g. to represent the integer 123, the term should assume ASN.1 type INTEGER, rather than the character string '123'.)

Personal names are an interesting boundary case where one might argue either for an ASN.1 based definition or a Format/Structure attribute indicating a normalized name according to some rules; the choice of the appropriate approach is best left to a bibliographic-attribute-definition working-group.

An attribute set definition that defines this type should include a discrete list of values. This attribute is non-repeatable.

ARCH 3.2.5.1 Dates

A date/time value might be expressed in any of the following forms:

1. ASN.1 type GeneralizedTime,
2. The Z39.50 ASN.1 Date/time definition,
3. Some other EXTERNAL definition for date and/or time,
4. ASN.1 type InternationalString. In case 4, a Format/Structure attribute should accompany the term, indicating for example, that the term is a normalized date. For cases 1 through 3, no Format/Structure attribute should be supplied.

ARCH 3.2.5.2 Character String

A term which is to be treated as a literal character string, or as a word-oriented phrase subject to preprocessing by the server, should be accompanied by the Format/Structure attribute 'Character String'. Whether and what type of preprocessing applies should be indicated by an Expansion/Interpretation attribute.

Whenever the 'Character String' Format/Structure attribute is supplied:

1. The order of the words in the supplied term is to be preserved (when preprocessing applies).
2. The Term should be represented as ASN.1 type InternationalString.

ARCH 3.2.6 Occurrence Attribute Type

The value if this attribute is the desired occurrence of an access point. For example "second occurrence of field-1". This is a non-repeating, numeric attribute.

ARCH 3.2.7 Indirection Attribute Type

The presence of this attribute indicates that the actual content of the term is not supplied, but instead, a pointer (e.g. url) to the term is supplied in lieu of the actual term. An attribute set definition that defines this type should include a discrete list of values; e.g. URL, URN. This attribute is non-repeatable.

ARCH 3.3 Enumeration and Summary of Class 1 Attribute Types

The table below enumerates and summarizes the Class 1 Attribute types.

An attribute set definition must use the numeric values in the "Type Number" column below to represent the types. If any of these types is omitted in an attribute set definition, the definition should skip the value for that type rather than renumber.

In the "value" column, 'list' means that when an attribute set defines that type, the attribute set definition should include a discrete list of values for the type.

In the Repeatable column, if the value is "yes" (meaning that the attribute type is repeatable) an attribute set definition may declare the type to be non-repeatable, but if the value is "no", an attribute set may not declare the type to be repeatable. (When an attribute set definition declares a type non-repeatable, this means that the attribute type may not repeat within any operand of a query, when the attribute set is specified as the dominant set for the query.)

In the Occurrence column, "mandatory" means that the attribute type must occur in an operand (it does not mean that a given attribute set must define that type). "Optional" means that in general the attribute type need not occur in every operand; however, a specific attribute set definition may declare that the attribute is mandatory (or mandatory in certain circumstances) in which case, the rules specified would be in effect when the attribute set is specified as the dominant set for a query. An attribute set definition may not declare an attribute type to be optional if it is listed as "mandatory" in this table.

Note: The numerical order of attributes types as listed in this table differs from the order in which they are defined in section 3.2.1: attribute type Functional Qualifier was added late, and was added to the table at the end, to avoid re-numbering.

Attribute Type	Type Number	Value	Repeatable	Occurrence	Roughly- corresponding Bib-1 Type
Access Point	1	list	yes	mandatory	Use
Semantic Qualifier	2	list	yes	optional	(new)

Attribute Type	Type Number	Value	Repeatable	Occurrence	Roughly- corresponding Bib-1 Type
Language	3	list	yes	optional	(new)
Content Authority	4	list	yes	optional	(new)
Expansion/Interpretation	5	list	yes	optional	Truncation and some of Relation
Normalized Weight	6	numeric	no	optional	(new)
Hit Count	7	numeric	no	optional	(new)
Comparison	8	list	no	mandatory	most of Relation and part of Completeness
Format/ Structure	9	list	no	optional	Structure
Occurrence	10	numeric	no	optional	(loosely) Completeness
Indirection	11	list	no	optional	(new)
Functional Qualifier	12	list	yes	optional	(new)

ARCH 3.4 Attribute List Construction

Within a properly constructed operand, the attribute list within an operand should:

1. Include attributes in ascending order by attribute type;
2. Include all types listed as mandatory;
3. Include no more than a single occurrence of a given type for any type listed as not repeatable; and
4. Conform to any further restrictions (not specified at the Class 1 level) on allowable combinations of attribute types, as specified by the attribute set definition for the dominant attribute set for the query.

ARCH 3.5 Utility and Cross Domain Attribute Sets

Both a Utility attribute set and a Cross Domain attribute set will be maintained; these will be Class 1 attribute sets.

The Utility set will define commonly used values for the Class 1 types. In addition it will include metatdata access points for ***records***, as distinguished from metatdata access points for ***resources***; the latter is the province of the Cross Domain set.

This distinction between record and resource is characterized by the example of a MARC record that describes a document. The MARC record is the "record" and the document is the "resource".

This is not to imply that this architecture (or that Class 1) models record and resource as always distinct. When the record ***is*** the resource (e.g. in a document database) the metadata access points are the province of the Cross Domain set.

An example of an access point that characterizes the difference in purpose of the two sets is 'language': There will be a language access point in both sets. Utility set Access Point Language will refer to the language of the database record, while Cross Domain set Access Point Language will refer to the value of the language field. For example a MARC record, created in English, might describe a French book. The Utility Access Point attribute Language would refer to the language of the MARC record, while the Cross Domain Access Point attribute Language would refer to the language of the book (English and French, respectively). Another example: the "creator" of the MARC record is similarly distinguished from the "creator" (or author) of the book that the record describes.

The Cross-domain set is defined for use by cross-domain queries (where a single query is applied to multiple domains) and more generally, for attributes that apply to multiple domains, whether to be used in a cross-domain query or not. For example, Access point 'title' will be defined in the Cross Domain set and then need not be defined in any domain-specific set. Thus a bibliographic set need not define 'title'; rather, it could define semantic and/or functional qualifiers for 'title'; then, effective access point "bibliographic title" would be constructed using 'title' from the Cross Domain set and appropriate semantic and/or functional qualifiers from the bibliographic set.

ARCH 4. Lessons Learned: Recommendations for Future Enhancements to the Z39.50 Query

As a result of the deliberations over this architecture, limitations posed by the type-1 query have resulted in identification of recommended enhancements that should be considered for a future version of Z39.50. These are documented here; additional contributions to this list are welcome and will be added:

1. The term in an operand should be replaced by a sequence of Terms. In the interim, ASN.1 definitions such as MultipleSearchTerms-1may be used.
2. Explicit range operators will be useful and should be added in favor of boolean combinations of operators that result in range definitions.
3. Attributes on operators should be supported.
4. Nesting should be handled at the operator level rather than by repeating attributes. That is, for "field 1 within field 2", 'within' should be an operator.

Appendix 18 ASN1: Z39.50 ASN.1

Normative

ASN1.1 Z39.50 APDUs

In this version (Z39.50-2003) some of the longer comments are presented outside of the actual ASN.1 module below, and for each of these there is a reference to the comment, within the ASN.1 at the point where the comment applies. These comments are numbered separately for each ASN.1 module (they are numbered beginning with 1 for each module). Every ASN.1 module concludes with the single line "END" (and conversely, "END" always ends a module). Comments for each module immediately precede the END line. Therefore, every reference of the form "See Comment n" refers to the comment with that number immediately preceding the next END line. All such comments, as well as comments included within the actual ASN.1, are normative and are part of the standard.

```
Z39-50-APDU-2001 -OID for this definition, assigned in OID.3.1, is {Z39-50 2 1}
DEFINITIONS ::=
BEGIN  -- November 2001
EXPORTS OtherInformation, Term, AttributeSetId, AttributeList, AttributeElement, ElementSetName,
SortElement, DatabaseName, CompSpec, Specification, Permissions, InternationalString, IntUnit, Unit,
StringOrNumeric, Query, Records, ResultSetId, DefaultDiagFormat, DiagRec;
APDU ::= CHOICE{
        initRequest                     [20] IMPLICIT InitializeRequest,
        initResponse                    [21] IMPLICIT InitializeResponse,
        searchRequest                   [22] IMPLICIT SearchRequest,
        searchResponse                  [23] IMPLICIT SearchResponse,
        presentRequest                  [24] IMPLICIT PresentRequest,
        presentResponse                 [25] IMPLICIT PresentResponse,
        deleteResultSetRequest          [26] IMPLICIT DeleteResultSetRequest,
        deleteResultSetResponse         [27] IMPLICIT DeleteResultSetResponse,
        accessControlRequest            [28] IMPLICIT AccessControlRequest,
        accessControlResponse           [29] IMPLICIT AccessControlResponse,
        resourceControlRequest          [30] IMPLICIT ResourceControlRequest,
        resourceControlResponse         [31] IMPLICIT ResourceControlResponse,
        triggerResourceControlRequest   [32] IMPLICIT TriggerResourceControlRequest,
        resourceReportRequest           [33] IMPLICIT ResourceReportRequest,
        resourceReportResponse          [34] IMPLICIT ResourceReportResponse,
        scanRequest                     [35] IMPLICIT ScanRequest,
        scanResponse                    [36] IMPLICIT ScanResponse,
                        -- [37] through [42] not used
        sortRequest                     [43] IMPLICIT SortRequest,
        sortResponse                    [44] IMPLICIT SortResponse,
        segmentRequest                  [45] IMPLICIT Segment,
        extendedServicesRequest         [46] IMPLICIT ExtendedServicesRequest,
        extendedServicesResponse        [47] IMPLICIT ExtendedServicesResponse,
        close                           [48] IMPLICIT Close,
        duplicateDetectionRequest       [49] IMPLICIT DuplicateDetectionRequest,
        duplicateDetectionResponse      [50] IMPLICIT DuplicateDetectionResponse}

-- Initialize APDUs
InitializeRequest ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        protocolVersion                         ProtocolVersion,
        options                                 Options,
```

```
        preferredMessageSize        [5]     IMPLICIT INTEGER,
        exceptionalRecordSize       [6]     IMPLICIT INTEGER,
        idAuthentication            [7]     ANY OPTIONAL,
                -- See comment 8
        implementationId            [110]   IMPLICIT InternationalString OPTIONAL,
        implementationName          [111]   IMPLICIT InternationalString OPTIONAL,
        implementationVersion       [112]   IMPLICIT InternationalString OPTIONAL,
        userInformationField        [11]    EXTERNAL OPTIONAL,
        otherInfo                           OtherInformation OPTIONAL}

InitializeResponse ::= SEQUENCE{
        referenceId                         ReferenceId OPTIONAL,
        protocolVersion                     ProtocolVersion,
        options                             Options,
        preferredMessageSize        [5]     IMPLICIT INTEGER,
        exceptionalRecordSize       [6]     IMPLICIT INTEGER,
                result              [12]    IMPLICIT BOOLEAN,
                        -- reject = FALSE; Accept = TRUE
        implementationId            [110]   IMPLICIT InternationalString OPTIONAL,
        implementationName          [111]   IMPLICIT InternationalString OPTIONAL,
        implementationVersion       [112]   IMPLICIT InternationalString OPTIONAL,
        userInformationField        [11]    EXTERNAL OPTIONAL,
        otherInfo                           OtherInformation OPTIONAL}

    --Begin auxiliary definitions for Init APDUs

ProtocolVersion  ::=[3] IMPLICIT BIT STRING{
        version-1       (0),        -- This bit should always be set,
                                    --but does not correspond to any Z39.50 version.
        version-2       (1),        -- "Version 2 supported." This bit should always be set.
        version-3       (2)         -- "Version 3 supported."
    -- See comment 9
    }

Options  ::= [4] IMPLICIT BIT STRING{
        search                  (0),
        present                 (1),
        delSet                  (2),
        resourceReport          (3),
        triggerResourceCtrl     (4),
        resourceCtrl            (5),
        accessCtrl              (6),
        scan                    (7),
        sort                    (8),
                -- (not used)   (9),
        extendedServices        (10),
        level-1Segmentation     (11),
        level-2Segmentation     (12),
        concurrentOperations    (13),
        namedResultSets         (14),
        encapsulation           (15),
        resultCountInSort       (16),
        negotiation             (17),
        dedup                   (18),
        query104                (19),
        pqesCorrection          (20),
        stringSchema            (21)}
```

```
-- End auxiliary definitions for Init APDUs

-- Search APDUs
SearchRequest ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        smallSetUpperBound              [13]    IMPLICIT INTEGER,
        largeSetLowerBound              [14]    IMPLICIT INTEGER,
        mediumSetPresentNumber          [15]    IMPLICIT INTEGER,
        replaceIndicator                [16]    IMPLICIT BOOLEAN,
        resultSetName                   [17]    IMPLICIT InternationalString,
        databaseNames                   [18]    IMPLICIT SEQUENCE OF DatabaseName,
        smallSetElementSetNames         [100]   ElementSetNames OPTIONAL,
        mediumSetElementSetNames        [101]   ElementSetNames OPTIONAL,
        preferredRecordSyntax           [104]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
        query                           [21]    Query,
-- Following two parameters may be used only if version 3 is in force.
        additionalSearchInfo            [203]   IMPLICIT OtherInformation OPTIONAL,
        otherInfo                               OtherInformation OPTIONAL}

--Query Definitions
Query  ::=          CHOICE{
        type-0  [0]         ANY,
        type-1  [1]         IMPLICIT RPNQuery,
        type-2  [2]         OCTET STRING,
        type-100[100]       OCTET STRING,
        type-101[101]       IMPLICIT RPNQuery,
        type-102[102]       OCTET STRING,
        type-104[104]       IMPLICIT EXTERNAL}

--Definitions for RPN query
RPNQuery ::= SEQUENCE{
        attributeSet        AttributeSetId,
        rpn                 RPNStructure}
RPNStructure ::= CHOICE{
        op                  [0] Operand,
        rpnRpnOp            [1] IMPLICIT SEQUENCE{
                                rpn1                    RPNStructure,
                                rpn2                    RPNStructure,
                                op                      Operator }}
Operand ::= CHOICE{
        attrTerm            AttributesPlusTerm,
        resultSet           ResultSetId,
--If version 2 is in force:
--If query type is 1, one of the above two must be chosen
--resultAttr (below) may be used only if query type is 101
resultAttr                  ResultSetPlusAttributes}
AttributesPlusTerm ::= [102] IMPLICIT SEQUENCE{
        attributes                  AttributeList,
        term                        Term}
ResultSetPlusAttributes ::= [214] IMPLICIT SEQUENCE{
        resultSet           ResultSetId,
        attributes          AttributeList}
AttributeList ::=           [44]    IMPLICIT SEQUENCE OF AttributeElement
Term ::= CHOICE{
        general                     [45]    IMPLICIT OCTET STRING,
-- Values below may be used only if version 3 is in force
        numeric                     [215]   IMPLICIT INTEGER,
```

```
        characterString         [216]   IMPLICIT InternationalString,
        oid                     [217]   IMPLICIT OBJECT IDENTIFIER,
        dateTime                [218]   IMPLICIT GeneralizedTime,
        external                [219]   IMPLICIT EXTERNAL,
        integerAndUnit          [220]   IMPLICIT IntUnit,
        null                    [221]   IMPLICIT NULL}
Operator ::= [46] CHOICE{
        and                     [0] IMPLICIT NULL,
        or                      [1] IMPLICIT NULL,
        and-not                 [2] IMPLICIT NULL,
--If version 2 is in force:
--For query type 1, one of the above three must be chosen;
--prox (below) may be used only if query type is 101.
        prox                    [3] IMPLICIT ProximityOperator}
AttributeElement  ::=  SEQUENCE{
        attributeSet    [1]     IMPLICIT AttributeSetId OPTIONAL,
--Must be omitted if version 2 is in force.
--If included, overrides value of attributeSet in RPNQuery above, but only for this attribute.
        attributeType   [120]   IMPLICIT INTEGER,
        attributeValue          CHOICE{
                                numeric [121] IMPLICIT INTEGER,
                -- If version 2 is in force, must select 'numeric' for attributeValue
                                complex[224] IMPLICIT SEQUENCE{
                                        list    [1]  IMPLICIT SEQUENCE OF StringOrNumeric,
                                semanticAction  [2]  IMPLICIT SEQUENCE OF INTEGER
                                                        OPTIONAL}
        -- See comment 10.
        }}
ProximityOperator ::= SEQUENCE{
        exclusion               [1] IMPLICIT BOOLEAN OPTIONAL,
        distance                [2] IMPLICIT INTEGER,
        ordered                 [3] IMPLICIT BOOLEAN,
        relationType            [4] IMPLICIT INTEGER{
                        lessThan                (1),
                        lessThanOrEqual         (2),
                        equal                   (3),
                        greaterThanOrEqual      (4),
                        greaterThan             (5),
                        notEqual                (6)},
proximityUnitCode               [5] CHOICE{
                                known   [1] IMPLICIT KnownProximityUnit,
                                private [2] IMPLICIT INTEGER}}
KnownProximityUnit ::= INTEGER{
        character       (1),
        word            (2),
        sentence        (3),
        paragraph       (4),
        section         (5),
        chapter         (6),
        document        (7),
        element         (8),
        subelement      (9),
        elementType     (10),
        byte            (11)    -- Version 3 only
        }
--End definitions for RPN Query
```

```
SearchResponse ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        resultCount                 [23]        IMPLICIT INTEGER,
        numberOfRecordsReturned     [24]        IMPLICIT INTEGER,
        nextResultSetPosition       [25]        IMPLICIT INTEGER,
        searchStatus                [22]        IMPLICIT BOOLEAN,
        resultSetStatus             [26]        IMPLICIT INTEGER{
                                                    subset  (1),
                                                    interim (2),
                                                    none    (3)} OPTIONAL,
        presentStatus                           PresentStatus OPTIONAL,
        records                                 Records OPTIONAL,
-- Following two parameters may be used only if version 3 is in force.
        additionalSearchInfo    [203]           IMPLICIT OtherInformation OPTIONAL,
        otherInfo                               OtherInformation OPTIONAL}

-- Retrieval APDUs
PresentRequest ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        resultSetId                             ResultSetId,
        resultSetStartPoint         [30]        IMPLICIT INTEGER,
        numberOfRecordsRequested    [29]        IMPLICIT INTEGER,
                additionalRanges    [212]       IMPLICIT SEQUENCE OF Range OPTIONAL,
-- additionalRanges may be included only if version 3 is in force.
        recordComposition           CHOICE{
                                    simple      [19]    ElementSetNames,
                                        -- Must choose 'simple' if version 2 is in force
                                    complex     [209]   IMPLICIT CompSpec} OPTIONAL,
        preferredRecordSyntax       [104]   IMPLICIT OBJECT IDENTIFIER OPTIONAL,
        maxSegmentCount             [204]   IMPLICIT INTEGER OPTIONAL,--level 1 or 2
                maxRecordSize       [206]   IMPLICIT INTEGER OPTIONAL,--level 2 only
                maxSegmentSize      [207]   IMPLICIT INTEGER OPTIONAL,--level 2 only
        otherInfo                           OtherInformation OPTIONAL}
Segment ::= SEQUENCE{
-- Segment APDU may only be used when version 3 is in force, and only when segmentation is in effect
        referenceId                             ReferenceId OPTIONAL,
        numberOfRecordsReturned     [24]        IMPLICIT INTEGER,
        segmentRecords              [0]         IMPLICIT SEQUENCE OF NamePlusRecord,
        otherInfo                               OtherInformation OPTIONAL}
PresentResponse ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        numberOfRecordsReturned     [24]        IMPLICIT INTEGER,
        nextResultSetPosition       [25]        IMPLICIT INTEGER,
        presentStatus                           PresentStatus,
        records                                 Records OPTIONAL,
        otherInfo                               OtherInformation OPTIONAL}

--Begin auxiliary definitions for Search and Present APDUs
--Begin definition of records
Records ::= CHOICE{
        responseRecords             [28]        IMPLICIT SEQUENCE OF NamePlusRecord,
        nonSurrogateDiagnostic      [130]       IMPLICIT DefaultDiagFormat,
        multipleNonSurDiagnostics   [205]       IMPLICIT SEQUENCE OF DiagRec}
```

```
NamePlusRecord ::= SEQUENCE{
        name                    [0] IMPLICIT DatabaseName OPTIONAL,
        record                  [1] CHOICE{
                        retrievalRecord         [1] EXTERNAL,
                        surrogateDiagnostic     [2] DiagRec,
                --Must select one of the above two, retrievalRecord or surrogateDiagnostic,
                --unless 'level 2 segmentation' is in effect.
                        startingFragment        [3] FragmentSyntax,
                        intermediateFragment    [4] FragmentSyntax,
                        finalFragment           [5] FragmentSyntax}}
        FragmentSyntax ::= CHOICE{
                        externallyTagged                EXTERNAL,
                        notExternallyTagged             OCTET STRING}
DiagRec ::= CHOICE{
                defaultFormat           DefaultDiagFormat,
-- Must choose defaultFormat if version 2 is in effect
                externallyDefined       EXTERNAL}
DefaultDiagFormat::= SEQUENCE{
        diagnosticSetId         OBJECT IDENTIFIER,
        condition               INTEGER,
        addinfo                 CHOICE{
                        v2Addinfo       VisibleString,          --Version 2
                        v3Addinfo       InternationalString     --Version 3
-- SEE COMMENT 1
        }}
--End definition of records
Range  ::= SEQUENCE{
        startingPosition                [1] IMPLICIT INTEGER,
        numberOfRecords                 [2] IMPLICIT INTEGER}
ElementSetNames ::= CHOICE {
        genericElementSetName   [0] IMPLICIT InternationalString,
        databaseSpecific        [1] IMPLICIT SEQUENCE OF SEQUENCE{
                        dbName          DatabaseName,
                        esn             ElementSetName}}
PresentStatus ::= [27]      IMPLICIT INTEGER{
        success  (0),
        partial-1       (1),
        partial-2       (2),
        partial-3       (3),
        partial-4       (4),
        failure         (5)}

-- Begin definition of composition specification
CompSpec ::= SEQUENCE{
        selectAlternativeSyntax     [1] IMPLICIT BOOLEAN,
--See comment for recordSyntax, below
        generic         [2] IMPLICIT Specification OPTIONAL,
        dbSpecific              [3] IMPLICIT SEQUENCE OF SEQUENCE{
                        db      [1] DatabaseName,
                        spec    [2] IMPLICIT Specification} OPTIONAL,
         -- At least one of generic and dbSpecific must occur,
        --  and both may occur. If both, then for any record no
        --  in the list of databases within dbSpecific, generic applies
                recordSyntax    [4] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL
        -- For each record, the server selects the first record syntax in
         -- this list that it can support.  If the list is exhausted, the server
        -- may select an alternative syntax if selectAlternativeSyntax is 'true'.
```

```
        }
Specification ::= SEQUENCE{
        schema CHOICE{
                oid [1] IMPLICIT OBJECT IDENTIFIER,
                uri [300] IMPLICIT InternationalString
                -- only if option bit  21 has been negotiated
        } OPTIONAL,
        elementSpec             [2] CHOICE{
                        elementSetName  [1] IMPLICIT InternationalString,
                        externalEspec   [2] IMPLICIT EXTERNAL} OPTIONAL}
-- End definition of composition specification
-- End auxiliary definitions for search and response APDUs

-- Delete APDUs
DeleteResultSetRequest ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        deleteFunction          [32]    IMPLICIT INTEGER{
                        list    (0),
                        all     (1)},
                resultSetList                   SEQUENCE OF ResultSetId OPTIONAL,
                otherInfo                       OtherInformation OPTIONAL}
DeleteResultSetResponse ::= SEQUENCE{
                referenceId                     ReferenceId OPTIONAL,
                deleteOperationStatus   [0]     IMPLICIT DeleteSetStatus,
                deleteListStatuses      [1]     IMPLICIT ListStatuses OPTIONAL,
                numberNotDeleted        [34]    IMPLICIT INTEGER OPTIONAL,
                bulkStatuses            [35]    IMPLICIT ListStatuses OPTIONAL,
                deleteMessage           [36]    IMPLICIT InternationalString OPTIONAL,
                otherInfo                       OtherInformation OPTIONAL}
ListStatuses ::= SEQUENCE OF SEQUENCE{
        id              ResultSetId,
        status          DeleteSetStatus}
DeleteSetStatus ::= [33] IMPLICIT INTEGER{
        success                                 (0),
        resultSetDidNotExist                    (1),
        previouslyDeletedByServer               (2),
        systemProblemAtServer                   (3),
        accessNotAllowed                        (4),
        resourceControlAtClient                 (5),
        resourceControlAtServer                 (6),
        bulkDeleteNotSupported                  (7),
        notAllRsltSetsDeletedOnBulkDlte         (8),
        notAllRequestedResultSetsDeleted        (9),
        resultSetInUse                          (10)}

--Access- and Resource-control APDUs
AccessControlRequest ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        securityChallenge                       CHOICE{
                        simpleForm              [37]    IMPLICIT OCTET STRING,
                        externallyDefined       [0]     EXTERNAL},
        otherInfo                               OtherInformation OPTIONAL}

AccessControlResponse ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        securityChallengeResponse               CHOICE{
                        simpleForm              [38]    IMPLICIT OCTET STRING,
```

```
                externallyDefined               [0]     EXTERNAL} OPTIONAL,
                --Optional only in version 3; mandatory in version 2.
                -- If omitted (in version 3) then diagnostic must occur.
            diagnostic                          [223]   DiagRec OPTIONAL,
                --Version 3 only

        otherInfo                                       OtherInformation OPTIONAL}

ResourceControlRequest ::= SEQUENCE{
        referenceId                                     ReferenceId OPTIONAL,
        suspendedFlag                   [39]            IMPLICIT BOOLEAN OPTIONAL,
        resourceReport                  [40]            ResourceReport OPTIONAL,
        partialResultsAvailable         [41]            IMPLICIT INTEGER{
                                                        subset  (1),
                                                        interim (2),
                                                        none    (3)} OPTIONAL,
        responseRequired                [42]            IMPLICIT BOOLEAN,
        triggeredRequestFlag            [43]            IMPLICIT BOOLEAN OPTIONAL,
        otherInfo                                       OtherInformation OPTIONAL}
ResourceControlResponse ::= SEQUENCE{
        referenceId                                     ReferenceId OPTIONAL,
        continueFlag                    [44]            IMPLICIT BOOLEAN,
        resultSetWanted                 [45]            IMPLICIT BOOLEAN OPTIONAL,
        otherInfo                                       OtherInformation OPTIONAL}
TriggerResourceControlRequest ::= SEQUENCE{
        referenceId                                     ReferenceId OPTIONAL,
        requestedAction                 [46]            IMPLICIT INTEGER{
                                                        resourceReport  (1),
                                                        resourceControl (2),
                                                        cancel          (3)},
prefResourceReportFormat        [47]    IMPLICIT ResourceReportId OPTIONAL,
        resultSetWanted         [48]    IMPLICIT BOOLEAN OPTIONAL,
        otherInfo                       OtherInformation OPTIONAL}
ResourceReportRequest ::= SEQUENCE{
        referenceId                                             ReferenceId OPTIONAL,
            opId                                        [210]   IMPLICIT ReferenceId OPTIONAL,
            prefResourceReportFormat                    [49]    IMPLICIT ResourceReportId OPTIONAL,
        otherInfo                                               OtherInformation OPTIONAL}
ResourceReportResponse ::= SEQUENCE{
        referenceId                                     ReferenceId OPTIONAL,
        resourceReportStatus                [50]        IMPLICIT INTEGER{
                        success  (0),
                        partial  (1),
                        failure-1(2),
                        failure-2(3),
                        failure-3(4),
                        failure-4(5),
                        failure-5(6),
                        failure-6(7)},
        resourceReport                      [51]        ResourceReport OPTIONAL,
        otherInfo                                       OtherInformation OPTIONAL}
ResourceReport  ::=     EXTERNAL
ResourceReportId        ::=     OBJECT IDENTIFIER

--Scan APDUs
ScanRequest ::= SEQUENCE{
        referenceId                                     ReferenceId OPTIONAL,
```

```
                databaseNames                   [3]     IMPLICIT SEQUENCE OF DatabaseName,
                attributeSet                            AttributeSetId OPTIONAL,
-- SEE COMMENT 2
        termListAndStartPoint                   AttributesPlusTerm,
                stepSize                        [5]     IMPLICIT INTEGER OPTIONAL,
        numberOfTermsRequested                  [6]     IMPLICIT INTEGER,
                preferredPositionInResponse     [7]     IMPLICIT INTEGER OPTIONAL,
                otherInfo                               OtherInformation OPTIONAL}
ScanResponse ::= SEQUENCE{
        referenceId                             ReferenceId OPTIONAL,
        stepSize                        [3]     IMPLICIT INTEGER OPTIONAL,
        scanStatus                      [4]     IMPLICIT INTEGER {
                                                success         (0),
                                                partial-1       (1),
                                                partial-2       (2),
                                                partial-3       (3),
                                                partial-4       (4),
                                                partial-5       (5),
                                                failure         (6) },
        numberOfEntriesReturned         [5]     IMPLICIT INTEGER,
        positionOfTerm                  [6]     IMPLICIT INTEGER OPTIONAL,
                entries                 [7]     IMPLICIT ListEntries  OPTIONAL,
                attributeSet            [8]     IMPLICIT AttributeSetId OPTIONAL,
-- SEE COMMENT 3
        otherInfo                               OtherInformation OPTIONAL}

--Begin auxiliary definitions for Scan
ListEntries ::= SEQUENCE{
                entries                 [1]     IMPLICIT SEQUENCE OF Entry OPTIONAL,
                nonsurrogateDiagnostics [2]     IMPLICIT SEQUENCE OF DiagRec OPTIONAL
--At least one of entries and nonsurrogateDiagnostics must occur
        }
Entry  ::= CHOICE {
        termInfo                [1]     IMPLICIT TermInfo,
        surrogateDiagnostic     [2]     DiagRec}
TermInfo ::= SEQUENCE {
        term                            Term,
                displayTerm             [0]     IMPLICIT InternationalString OPTIONAL,
        -- See comment 11
        suggestedAttributes             AttributeList OPTIONAL,
        alternativeTerm         [4]     IMPLICIT SEQUENCE OF AttributesPlusTerm OPTIONAL,
        globalOccurrences       [2]     IMPLICIT INTEGER OPTIONAL,
        byAttributes            [3]     IMPLICIT OccurrenceByAttributes OPTIONAL,
        otherTermInfo                   OtherInformation OPTIONAL}
OccurrenceByAttributes ::= SEQUENCE OF SEQUENCE{
        attributes              [1]     AttributeList,
        occurrences             CHOICE{
                                global          [2]     INTEGER,
                                byDatabase      [3]     IMPLICIT SEQUENCE OF SEQUENCE{
                                                        db              DatabaseName,
                                                        num     [1]     IMPLICIT INTEGER OPTIONAL,
                                otherDbInfo             OtherInformation OPTIONAL}} OPTIONAL,
        otherOccurInfo          OtherInformation OPTIONAL}
--End auxiliary definitions for Scan

--Sort APDUs
SortRequest  ::= SEQUENCE{
```

```
        referenceId                     ReferenceId OPTIONAL,
        inputResultSetNames     [3]     IMPLICIT SEQUENCE OF InternationalString,
        sortedResultSetName     [4]     IMPLICIT InternationalString,
        sortSequence            [5]     IMPLICIT SEQUENCE OF SortKeySpec,
--Separate instance of SortKeySpec  for each sort key
--Order of occurrence is from major to minor
        otherInfo                       OtherInformation OPTIONAL}
SortResponse ::= SEQUENCE{
        referenceId                     ReferenceId OPTIONAL,
        sortStatus              [3]     IMPLICIT INTEGER{
                                        success  (0),
                                        partial-1 (1),
                                        failure   (2)},
        resultSetStatus         [4]     IMPLICIT INTEGER{
                                        empty           (1),
                                        interim         (2),
                                        unchanged       (3),
                                        none            (4)} OPTIONAL,
                diagnostics             [5]     IMPLICIT SEQUENCE OF DiagRec
                OPTIONAL,
        resultCount             [6] IMPLICIT INTEGER OPTIONAL,
                -- Size of the output result set.
                --Server is never obligated to supply this parameter,
                --there is no default value, and the client should
                -- not draw any conclusion by its omission
        otherInfo                       OtherInformation OPTIONAL}

--Begin auxiliary definitions for Sort
SortKeySpec ::= SEQUENCE{
        sortElement                     SortElement,
        sortRelation            [1]     IMPLICIT INTEGER{
                                        ascending               (0),
                                        descending              (1),
                                        ascendingByFrequency    (3),
                                        descendingByfrequency   (4)},
-- SEE COMMENT 4
        caseSensitivity         [2]     IMPLICIT INTEGER{
                                        caseSensitive           (0),
                                        caseInsensitive         (1)},
        missingValueAction              [3]     CHOICE{
                                                abort   [1] IMPLICIT NULL,
                                                null    [2] IMPLICIT NULL,
--Supply a null value for missing value
        missingValueData                [3] IMPLICIT OCTET STRING} OPTIONAL}
SortElement ::=   CHOICE{
        generic                 [1] SortKey,
        datbaseSpecific         [2] IMPLICIT SEQUENCE OF SEQUENCE{
                        databaseName    DatabaseName,
                        dbSort          SortKey}}
SortKey ::= CHOICE{
-- See comment 12
        privateSortKey          [0]     IMPLICIT InternationalString,
        elementSpec             [1]     IMPLICIT Specification,
        sortAttributes          [2]     IMPLICIT SEQUENCE{
                        id      AttributeSetId,
                        list    AttributeList}}
--End auxiliary definitions for sort
```

```
--Extended Service APDUs
ExtendedServicesRequest ::= SEQUENCE{
        referenceId                     ReferenceId OPTIONAL,
        function                [3]     IMPLICIT INTEGER {
                                        create  (1),
                                        delete  (2),
                                        modify  (3)},
        packageType             [4]     IMPLICIT OBJECT IDENTIFIER,
                packageName     [5]     IMPLICIT InternationalString OPTIONAL,
                        --PackageName is mandatory for 'modify' or 'delete'; optional for 'create'.
                        --Following four parameters mandatory for 'create'; should be included on
                        --'modify' if being modified; it is not needed on 'delete'.
                userId          [6]     IMPLICIT InternationalString OPTIONAL,
        retentionTime           [7]     IMPLICIT IntUnit OPTIONAL,
        permissions             [8]     IMPLICIT Permissions OPTIONAL,
        description             [9]     IMPLICIT InternationalString OPTIONAL,
        taskSpecificParameters  [10]    IMPLICIT EXTERNAL OPTIONAL,
                --Mandatory for 'create'; included on 'modify' if specific parameters being modified;
                -- not necessary on 'delete'. For the 'EXTERNAL,' use OID of specific
                --ES definition and select CHOICE [1]: 'esRequest'.
        waitAction                      [11]    IMPLICIT INTEGER{
                                                        wait                    (1),
                                                        waitIfPossible          (2),
                                                        dontWait                (3),
                                                        dontReturnPackage       (4)},
        elements                                IMPLICIT ElementSetName OPTIONAL,
        otherInfo                               IMPLICIT OtherInformation OPTIONAL}
ExtendedServicesResponse ::= SEQUENCE{
        referenceId                     ReferenceId OPTIONAL,
        operationStatus         [3]     IMPLICIT INTEGER{
                                                done            (1),
                                                accepted        (2),
                                                failure         (3)},
                diagnostics     [4]     IMPLICIT SEQUENCE OF DiagRec OPTIONAL,
        taskPackage             [5]     IMPLICIT EXTERNAL OPTIONAL,
        -- Use OID: {Z39-50-recordSyntax (106)} and corresponding syntax.
        -- For the EXTERNAL, 'taskSpecific,' within that definition, use OID
        -- of the specific ES, and choose [2], 'taskPackage'.
        otherInfo                               IMPLICIT OtherInformation OPTIONAL}
Permissions ::= SEQUENCE OF SEQUENCE{
        userId                  [1] IMPLICIT InternationalString,
        allowableFunctions      [2] IMPLICIT SEQUENCE OF INTEGER{
                                        delete                  (1),
                                        modifyContents          (2),
                                        modifyPermissions       (3),
                                        present                 (4),
                                        invoke                  (5)}}

Close ::= SEQUENCE{
                referenceId                             ReferenceId OPTIONAL,
                -- See 3.2.11.1.5
        closeReason                     CloseReason,
        diagnosticInformation   [3]     IMPLICIT InternationalString OPTIONAL,
        resourceReportFormat    [4]     IMPLICIT ResourceReportId OPTIONAL,
                --For use by client only, and only on Close request
                --Client requests server to include report in response
```

```
        resourceReport          [5]     ResourceReport OPTIONAL,
                --For use by server only, unilaterally on Close request
                --On Close response may be unilateral or in response to client request
        otherInfo                               OtherInformation OPTIONAL}
CloseReason ::=         [211]   IMPLICIT INTEGER{
        finished                (0),
        shutdown                (1),
        systemProblem           (2),
        costLimit               (3),
        resources               (4),
        securityViolation       (5),
        protocolError           (6),
        lackOfActivity          (7),
        responseToPeer          (8),
        unspecified             (9)}
DuplicateDetectionRequest ::= SEQUENCE {
        referenceId                             ReferenceId OPTIONAL,
        inputResultSetIds               [3] IMPLICIT SEQUENCE OF InternationalString,
        outputResultSetName             [4] IMPLICIT InternationalString,
        applicablePortionOfRecord       [5] IMPLICIT EXTERNAL OPTIONAL,
        duplicateDetectionCriteria      [6] IMPLICIT SEQUENCE OF
                                                DuplicateDetectionCriterion OPTIONAL,
        clustering                      [7] IMPLICIT BOOLEAN OPTIONAL,
                --'true' means "clustered".
                --This parameter may be omitted only if retentionCriteria
                -- CHOICE is 'numberOfEntries' and its value is 1
        retentionCriteria               [8] IMPLICIT SEQUENCE OF RetentionCriterion,
        sortCriteria                    [9] IMPLICIT SEQUENCE OF SortCriterion OPTIONAL,
        otherInfo                               OtherInformation OPTIONAL}
DuplicateDetectionCriterion ::= CHOICE{
        levelOfMatch            [1] IMPLICIT INTEGER,
                --A percentage; 1-100
        caseSensitive           [2] IMPLICIT NULL,
        punctuationSensitive    [3] IMPLICIT NULL,
        regularExpression       [4] IMPLICIT EXTERNAL,
        rsDuplicates            [5] IMPLICIT NULL
                                        --Values 6-100 reserved for future assignment
        }
RetentionCriterion ::= CHOICE{
        numberOfEntries         [1] IMPLICIT INTEGER,
                                        --Greater than 0
        percentOfEntries        [2] IMPLICIT INTEGER,
                                        --1-100
        duplicatesOnly          [3] IMPLICIT NULL,
                                        --Should not be chosen if clustering is 'true'
        discardRsDuplicates     [4] IMPLICIT NULL
                                        --Values 5-100 reserved for future assignment
        }
SortCriterion ::= CHOICE{
        mostComprehensive               [1] IMPLICIT NULL,
        leastComprehensive              [2] IMPLICIT NULL,
        mostRecent                      [3] IMPLICIT NULL,
        oldest                          [4] IMPLICIT NULL,
        leastCost                       [5] IMPLICIT NULL,
        preferredDatabases              [6] IMPLICIT SEQUENCE OF InternationalString
                                        --Values 7-100 reserved for future assignment
        }
```

```
DuplicateDetectionResponse ::= SEQUENCE {
        referenceId             ReferenceId  OPTIONAL,
        status          [3]     IMPLICIT INTEGER{
                                        success  (0),
                                        failure  (1)},
        resultSetCount  [4]     IMPLICIT INTEGER OPTIONAL,
        diagnostics     [5]     IMPLICIT SEQUENCE OF DiagRec OPTIONAL,
        otherInfo               OtherInformation OPTIONAL}

--Global auxiliary definitions
ReferenceId             ::=             [2]     IMPLICIT OCTET STRING
ResultSetId             ::=             [31]    IMPLICIT InternationalString
ElementSetName          ::=             [103]   IMPLICIT InternationalString
DatabaseName            ::=             [105]   IMPLICIT InternationalString
AttributeSetId          ::=                     OBJECT IDENTIFIER

--OtherInformation
--SEE COMMENT 5
OtherInformation ::= [201] IMPLICIT SEQUENCE OF SEQUENCE{
        category                        [1]     IMPLICIT InfoCategory OPTIONAL,
        information     CHOICE{
                        characterInfo           [2]     IMPLICIT InternationalString,
                        binaryInfo              [3]     IMPLICIT OCTET STRING,
                        externallyDefinedInfo   [4]     IMPLICIT EXTERNAL,
                        oid                     [5]     IMPLICIT OBJECT IDENTIFIER}}
InfoCategory ::= SEQUENCE{
        categoryTypeId  [1]     IMPLICIT OBJECT IDENTIFIER OPTIONAL,
        categoryValue   [2]     IMPLICIT INTEGER}

--Units
--SEE COMMENT 6
IntUnit ::= SEQUENCE{
        value           [1] IMPLICIT INTEGER,
        unitUsed        [2] IMPLICIT Unit}
Unit ::= SEQUENCE{
        unitSystem              [1] InternationalString OPTIONAL,       --e.g. 'SI'
        unitType                [2] StringOrNumeric OPTIONAL,           --e.g. 'mass'
        unit                    [3] StringOrNumeric OPTIONAL,           --e.g. 'kilograms'
        scaleFactor             [4] IMPLICIT INTEGER OPTIONAL           --e.g. 9 means 10**9
        }

--CharacterString
InternationalString ::= GeneralString
--SEE COMMENT 7
StringOrNumeric ::= CHOICE{
        string    [1] IMPLICIT InternationalString,
        numeric   [2] IMPLICIT INTEGER}
END IR DEFINITIONS
```

Comments

Comments referred to within the text of the ASN.1 are presented here.

Comment 1

A server should always include addInfo, even if it doesn't have any meaningful additional information to provide, including the case where the diagnostic's definition doesn't even specify the nature of the additional information. This is for historical reasons, for compatibility with earlier versions. (A client may, if it wishes, accept a diagnostic where addInfo is omitted, without considering this to be a protocol error.)

When a diagnostic's definition does not specify the nature of the additional information, it is recommended that the server either supply a text string, for example, "no further information available" (and it is further recommended in this case that the client show the message to the user) or supply a well-known value, in lieu of, and to mean "no additional information available". The well-known value would be either "null" or a zero-length string. The first option is recommended in cases where the server knows what language the client prefers, as in the case where a language has been negotiated. When the server does not know what language the client prefers, the second option is recommended; note, however (for the well-known value) a zero-length string poses a problem for some implementations.
When a diagnostic's definition does specify a particular type of additional information - as for example, diagnostic 109: "database unavailable", which specifies addInfo as "database name" - the use of a well-known value is inappropriate, because it cannot be distinguished from a real value (for example, "null" could be a database name). However, in this case the server should usually be capable of providing the specified information. For example, if a server declares "database unavailable" it should be able to say which database.

Comment 2
This is optional because an attribute set id may be provided within termListandStartPoint, which is of type AttributesPlusTerm, which includes AttributeList, which provides for an attribute set id to be included for every attribute type/value pair. The attribute set id may be omitted here only if an attribute set id is attached individually to every attribute pair. No attribute pair should remain unqualified by an attribute set id. If a Scan request is sent where one or more attribute pairs remains unqualified (that is, where the attribute set id parameter is omitted in the Scan request and is also omitted for one or more attribute pairs) the server should treat this condition as an error. The server, at its discretion, may treat this either as a protocol error or may fail the Scan and return a diagnostic. Diagnostic 1051: " Scan: Attribute set id required -- not supplied" has been defined for this condition.

Comment 3
This pertains to attribute lists referenced elsewhere in the ScanResponse. Both in TermInfo, and in OccurrenceByAttributes, where AttributeList or AttributePlusTerm are referenced respectively, both reference attributeElement, which may or may not include an attribute set identifier; attributeSet specifies the attribute set identifier in cases where it is omitted in attributeElement.

Comment 4
"Ascending by frequency" (or "descending by frequency") is distinguished from "ascending" (or "descending") as illustrated in this example. Suppose we do a search on Moby Dick, get six hits and sort by author, ascending (conventionally ascending, i.e. not by frequency), and the resulting order is:
1. Author: Bradbury, Ray
Title: Moby Dick and that Bride of Time
2. Author: Bradbury, Ray
Title: Moby Dick and the Dandelion Wine
3. Author: Herbert, T. Walter
Title: Moby-Dick and Calvinism
4. Author: Melville, Herman
Title: Moby Dick : or, The whale : Paintings byLeRoy Neiman
5. Author: Melville, Herman
Title: Moby-Dick : or, The whale; Commentary by Howard Mumford Jones
6. Author: Melville, Herman, 1819-1891.
Title: Moby Dick : or, The whale; Introd. by A. S. W. Rosenbach.
If we do a "descending by frequency" sort, the order (of authors) would be Melville, Bradbury, Walker; because Melville had the highest frequency as author (3), Bradbury second (2), and Walker third (1). (Note that the ordering of items for a given author, e.g. for the three Melville items, is determined by the server and cannot necessarily be predicted by the client.)

Comment 5
OtherInfo was originally developed with as much flexibility as possible, since the scope of its usage was not clear at the time. It was anticipated that "categories" would be useful, but no known values for

categoryValue have yet been adopted or assigned, and none (known) in use, nor are there any categories (known) in use. The category itself is optional, and if used, categoryTypeId is optional. So there may be:
(1) No category at all,
(2) A category without qualification, or
(3) A category with qualification.
(1) likely will be used predominantly. (2) would be used between partners who have a prior agreement. For (3), the intention is that categoryTypeId would identify some registration agent. It is not intended as a classification mechanism.

Comment 6
IntUnit is used when value and unit are supplied together. Unit, alone, is used when just specifying a unit (without a value). For example, IntUnit is used in Term, in an RPNQuery, or it can be the datatype of an element within a retrieval record. Unit (alone) would be used in an element request, when requesting data be returned according to a particular unit.

Comment 7
When version 2 is in force, this collapses to VisibleString. That is, only characters in the visibleString repertoire may be used. (Datatype compatibility with version 2 is not affected, because references are IMPLICIT.) When version 3 is in force, the semantics of the GeneralString content may be altered by negotiation during initialization. If no such negotiation is in effect, then GeneralString semantics are in force.
InternationalString was defined first in Z39.50-1995, and the intent was to depricate VisibleString. There are asn.1 identifiers from the 1992 version of type visibleString, so the 1995 version stipulates that when v2 is in effect (because if v2 is in effect one of the two systems may be based on the 1992 version), those parameters are restricted to the visibleString repertoire. Choice of tag, 26 or 27, is not an issue where identifiers are defined as IMPLICIT. In two cases they are not, and these are treated separately.

Comment 8
For idAuthentication, the type ANY is retained for compatibility with earlier versions. For interoperability, the following is recommended:

```
idAuthentication [7] CHOICE{
        open      VisibleString,  --  for compatibility with 1992 version
        idPass    SEQUENCE {
        groupId                   [0]     IMPLICIT InternationalString OPTIONAL,
        userId                    [1]     IMPLICIT InternationalString OPTIONAL,
        password                  [2]     IMPLICIT InternationalString OPTIONAL },
anonymous               NULL,
other                   EXTERNAL
--May use access control formats for 'other'. See Appendix  ACC.
```

Comment 9
Values higher than 'version-3' should be ignored. Both the Initialize request and Initialize Response APDUs include a value string corresponding to the supported versions. The highest common version is selected for use. If there are no versions in common, "Result" in the Init Response should indicate "reject."
Note: Versions 1 and 2 are identical. Systems supporting version 2 should indicate support for version 1 as well, for interoperability with systems that indicate support for version 1 only (e.g. ISO 10163-1991 implementations).

Comment 10
The attribute set definition (attributeSet) describes how these are to be used, for example whether there may be multiple values (for ‘list’) for a given type, whether they are string or numeric or a combination, the allowable values, semantics for each value, values of ‘semanticAction’, and semantics for each.

Comment 11
Presence of displayTerm means that term is not considered by the server to be suitable for display, and displayTerm should instead be displayed.
'term' is the actual term in the term list; 'displayTerm' is for display purposes only, and is not an actual term in the term list.

Comment 12
PrivateSortKey is used when the client and server have a prior agreement about what the supplied key means. For example the client may supply the string 'author', where that string doesn't denote any particular field in the record as defined by the schema, and it isn't an attribute (no attribute set id) but the server knows (because of prior agreement) that that string, when supplied as a private sort key, and when applied to the records in question, refers to a specific field.

ElementSpec is a retrieval element of the record, as defined perhaps by a schema, and specified by an element specification, such as eSpec-1, or by an element set name. The element specification or name should resolve to a single element; if it resolves to several elements, the sort key is not well-defined.

The purpose of sortAttributes is to specify a search field; this may require multiple attributes. However if several attributes are supplied, they should resolve to a single abstract access point.

ASN1.2 Z39.50 ASN.1 Object Identifier Definitions for Object Classes

```
ANSI-Z39-50-ObjectIdentifier  DEFINITIONS ::=
BEGIN
Z39-50  OBJECT IDENTIFIER ::=
{ iso (1) member-body (2) US (840) ANSI-standard-Z39.50 (10003)}
-- thus {Z39-50} is shorthand for {1 2 840 10003}
Z39-50-APDU             OBJECT IDENTIFIER ::= {Z39-50 2}      -- See OID.3
-- and {Z39-50 2} is shorthand for {1 2 840  10003 2} and so on.
Z39-50-attributeSet     OBJECT IDENTIFIER ::= {Z39-50 3}      -- See Appendix ATR
Z39-50-diagnostic       OBJECT IDENTIFIER ::= {Z39-50 4}      -- See Appendix DIAG
Z39-50-recordSyntax     OBJECT IDENTIFIER ::= {Z39-50 5}      --See Appendix REC
Z39-50-resourceReport   OBJECT IDENTIFIER ::= {Z39-50 7}      --See Appendix RSC
Z39-50-accessControl    OBJECT IDENTIFIER ::= {Z39-50 8}      --See Appendix ACC
Z39-50-extendedService  OBJECT IDENTIFIER ::= {Z39-50 9}      --See Appendix EXT
Z39-50-userInfoFormat   OBJECT IDENTIFIER ::= {Z39-50 10}     --See Appendix USR
Z39-50-elementSpec      OBJECT IDENTIFIER ::= {Z39-50 11}     --See Appendix ESP
Z39-50-variantSet       OBJECT IDENTIFIER ::= {Z39-50 12}     --See Appendix VAR
Z39-50-schema           OBJECT IDENTIFIER ::= {Z39-50 13}     --See Appendix TAG
Z39-50-tagSet           OBJECT IDENTIFIER ::= {Z39-50 14}     --See Appendix TAG
Z39-50-negotiation      OBJECT IDENTIFIER ::= {Z39-50 15}
Z39-50-query                    OBJECT IDENTIFIER ::= {Z39-50 16}
END
```

ASN1.3 Z39.50 ASN.1 Definition for GeneralDiagnosticContainer

```
GeneralDiagnosticContainer
{Z39-50-diagnosticFormat generalDiagnosticContainer (4)} DEFINITIONS ::=
BEGIN
IMPORTS DiagRec, DefaultDiagFormat FROM Z39-50-APDU-2001;
DiagnosticContainer ::=  SEQUENCE OF DiagRec
END
```

ASN1.4 Z39.50 ASN.1 Definition for Explain

```
RecordSyntax-explain
{Z39-50-recordSyntax explain (100)} DEFINITIONS ::=
BEGIN
IMPORTS AttributeSetId, Term, OtherInformation, DatabaseName, ElementSetName, IntUnit, Unit,
StringOrNumeric, Specification, InternationalString, AttributeList, AttributeElement FROM
Z39-50-APDU-2001;
Explain-Record ::= CHOICE{
--Each of these may be used as search term when Use attribute is 'explain-category'.
        targetInfo              [0]     IMPLICIT TargetInfo,
        databaseInfo            [1]     IMPLICIT DatabaseInfo,
        schemaInfo              [2]     IMPLICIT SchemaInfo,
        tagSetInfo              [3]     IMPLICIT TagSetInfo,
        recordSyntaxInfo        [4]     IMPLICIT RecordSyntaxInfo,
        attributeSetInfo        [5]     IMPLICIT AttributeSetInfo,
        termListInfo            [6]     IMPLICIT TermListInfo,
        extendedServicesInfo    [7]     IMPLICIT ExtendedServicesInfo,
        attributeDetails        [8]     IMPLICIT AttributeDetails,
        termListDetails         [9]     IMPLICIT TermListDetails,
        elementSetDetails       [10]    IMPLICIT ElementSetDetails,
        retrievalRecordDetails  [11]    IMPLICIT RetrievalRecordDetails,
        sortDetails             [12]    IMPLICIT SortDetails,
        processing              [13]    IMPLICIT ProcessingInformation,
        variants                [14]    IMPLICIT VariantSetInfo,
        units                   [15]    IMPLICIT UnitInfo,
        categoryList            [100]   IMPLICIT CategoryList}

    -- See Comment 1

TargetInfo ::= SEQUENCE {
        commonInfo                      [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        name                            [1]     IMPLICIT InternationalString,
--See comment 2
--Non-key brief elements follow:
        recent-news                     [2]     IMPLICIT HumanString OPTIONAL,
        icon                            [3]     IMPLICIT IconObject OPTIONAL,
--Element set name “brief-1' is defined for use when client wants icon to be omitted;
--otherwise ‘brief-1' is identical to ‘brief’.
        namedResultSets         [4]     IMPLICIT BOOLEAN,
        multipleDBsearch        [5]     IMPLICIT BOOLEAN,
        maxResultSets           [6]     IMPLICIT INTEGER OPTIONAL,
        maxResultSize           [7]     IMPLICIT INTEGER OPTIONAL,
        maxTerms                [8]     IMPLICIT INTEGER OPTIONAL,
        timeoutInterval         [9]     IMPLICIT IntUnit OPTIONAL,
        welcomeMessage          [10]    IMPLICIT HumanString OPTIONAL,
--Non-brief elements follow:
--'description' esn retrieves the following two (as well as brief):
        contactInfo             [11]    IMPLICIT ContactInfo OPTIONAL,
        description             [12]    IMPLICIT HumanString OPTIONAL,
        nicknames               [13]    IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        usage-restrictions      [14]    IMPLICIT HumanString OPTIONAL,
        paymentAddr             [15]    IMPLICIT HumanString OPTIONAL,
        hours                   [16]    IMPLICIT HumanString OPTIONAL,
        dbCombinations          [17]    IMPLICIT SEQUENCE OF DatabaseList OPTIONAL,
        addresses               [18]    IMPLICIT SEQUENCE OF NetworkAddress OPTIONAL,
        languages               [101]   IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
```

```
--Languages supported for message strings. Each is a three-character language code from Z39.53-1994.
    -- characterSets    [102]
--This tag reserved for "character sets supported for name and message strings"
    -- commonAccessInfo elements list objects the server supports.
--  All objects listed in AccessInfo for any individual database should also be listed here.
        commonAccessInfo                [19]    IMPLICIT AccessInfo OPTIONAL}
DatabaseInfo ::= SEQUENCE {
            --A server may provide "virtual databases" that are combinations of individual database.
            --These databases are indicated by the presence of subDbs in the combination database's
            --DatabaseDescription.
        commonInfo                      [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        name                            [1]     IMPLICIT DatabaseName,
--Non-key brief elements follow:
        explainDatabase         [2]     IMPLICIT NULL OPTIONAL,
--See comment 3
        nicknames               [3]     IMPLICIT SEQUENCE OF DatabaseName OPTIONAL,
        icon                    [4]     IMPLICIT IconObject OPTIONAL,
-- Element set name “brief-1' is defined for use when client wants icon to be omitted;
--otherwise ‘brief-1' is identical to ‘brief’
        user-fee                [5]     IMPLICIT BOOLEAN,
        available               [6]     IMPLICIT BOOLEAN,
        titleString             [7]     IMPLICIT HumanString OPTIONAL,
--Non-brief elements follow:
        keywords                [8]     IMPLICIT SEQUENCE OF HumanString OPTIONAL,
        description             [9]     IMPLICIT HumanString OPTIONAL,
        associatedDbs           [10]    IMPLICIT DatabaseList OPTIONAL,
--Databases that may be searched in combination with this one
        subDbs                  [11]    IMPLICIT DatabaseList OPTIONAL,
--When present, this database is a composite representing the combined databases 'subDbs'.
--The individual subDbs are also available
        disclaimers             [12]    IMPLICIT HumanString OPTIONAL,
        news                    [13]    IMPLICIT HumanString OPTIONAL,
        recordCount             [14]    CHOICE {
        actualNumber            [0]     IMPLICIT INTEGER,
        approxNumber            [1]     IMPLICIT INTEGER} OPTIONAL,
        defaultOrder            [15]    IMPLICIT HumanString OPTIONAL,
        avRecordSize            [16]    IMPLICIT INTEGER OPTIONAL,
        maxRecordSize           [17]    IMPLICIT INTEGER OPTIONAL,
        hours                   [18]    IMPLICIT HumanString OPTIONAL,
        bestTime                [19]    IMPLICIT HumanString OPTIONAL,
        lastUpdate              [20]    IMPLICIT GeneralizedTime OPTIONAL,
        updateInterval          [21]    IMPLICIT IntUnit OPTIONAL,
        coverage                [22]    IMPLICIT HumanString OPTIONAL,
        proprietary                     [23]    IMPLICIT BOOLEAN OPTIONAL,
                --mandatory in full record
        copyrightText           [24]    IMPLICIT HumanString OPTIONAL,
        copyrightNotice         [25]    IMPLICIT HumanString OPTIONAL,
        producerContactInfo     [26]    IMPLICIT ContactInfo OPTIONAL,
        supplierContactInfo     [27]    IMPLICIT ContactInfo OPTIONAL,
        submissionContactInfo   [28]    IMPLICIT ContactInfo OPTIONAL,
-- accessInfo lists items connected with the database.
--All listed items should be in the server's AccessInfo.
        accessInfo                      [29]    IMPLICIT AccessInfo OPTIONAL}
SchemaInfo ::= SEQUENCE {
        commonInfo                      [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
```

```
        schema                          [1]     IMPLICIT OBJECT IDENTIFIER,
--Non-key brief elements follow:
        name                            [2]     IMPLICIT InternationalString,
--Non-brief elements follow:
        description                     [3]     IMPLICIT HumanString OPTIONAL,
        tagTypeMapping                  [4]     IMPLICIT SEQUENCE OF SEQUENCE {
                                                        tagType         [0] IMPLICIT INTEGER,
                                                        tagSet          [1] IMPLICIT OBJECT
                                                        IDENTIFIER OPTIONAL,
--If tagSet is omitted, then this tagType is for a tagSet
-- locally defined within the schema that cannot be referenced by another schema.
                                                        defaultTagType  [2] IMPLICIT NULL
                                                        OPTIONAL} OPTIONAL,
            recordStructure             [5]     IMPLICIT SEQUENCE OF ElementInfo PTIONAL}
--ElementInfo referenced in SchemaInfo and RecordSyntaxInfo
ElementInfo ::= SEQUENCE {
        elementName                     [1] IMPLICIT InternationalString,
        elementTagPath                  [2] IMPLICIT Path,
                dataType                [3] ElementDataType OPTIONAL,
                                                -- If omitted, not specified
        required                        [4] IMPLICIT BOOLEAN,
        repeatable                      [5] IMPLICIT BOOLEAN,
        description                     [6] IMPLICIT HumanString OPTIONAL}
--Path is referenced by ElementInfo as well as PerElementDetails
Path ::= SEQUENCE OF SEQUENCE{
        tagType             [1] IMPLICIT INTEGER,
        tagValue            [2] StringOrNumeric}
ElementDataType ::= CHOICE{
        primitive                       [0] IMPLICIT PrimitiveDataType,
        structured                      [1] IMPLICIT SEQUENCE OF ElementInfo}
PrimitiveDataType ::= INTEGER{
        octetString         (0),
        numeric             (1),
        date                (2),
        external            (3),
        string              (4),
        trueOrFalse         (5),
        oid                 (6),
        intUnit             (7),
        empty               (8),
        noneOfTheAbove      (100)   -- See 'description'
        }
TagSetInfo ::= SEQUENCE {
        commonInfo          [0]     IMPLICIT CommonInfo OPTIONAL,
-- Key elements follow:
        tagSet              [1]     IMPLICIT OBJECT IDENTIFIER,
--Non-key brief elements follow:
        name                [2]     IMPLICIT InternationalString,
--Non-brief elements follow:
        description         [3]     IMPLICIT HumanString OPTIONAL,
        elements            [4]     IMPLICIT SEQUENCE OF SEQUENCE {
                    elementname     [1]     IMPLICIT InternationalString,
                    nicknames       [2]     IMPLICIT SEQUENCE OF
                                            InternationalString OPTIONAL,
                    elementTag      [3]     StringOrNumeric,
                    description     [4]     IMPLICIT HumanString OPTIONAL,
                    dataType        [5]     PrimitiveDataType OPTIONAL,
```

```
--If the data type is expected to be structured,
--that is described in the schema info, and datatype is omitted here.
                        otherTagInfo      OtherInformation OPTIONAL} OPTIONAL}
RecordSyntaxInfo ::= SEQUENCE {
        commonInfo              [0] IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        recordSyntax            [1] IMPLICIT OBJECT IDENTIFIER,
--Non-key brief elements follow:
        name                    [2] IMPLICIT InternationalString,
--Non-brief elements follow:
        transferSyntaxes        [3] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
        description             [4] IMPLICIT HumanString OPTIONAL,
                asn1Module      [5] IMPLICIT InternationalString OPTIONAL,
                abstractStructure [6] IMPLICIT SEQUENCE OF ElementInfo OPTIONAL
--Omitting abstractStructure only means server isn't using Explain
--to describe the structure, not that there is no structure
        }
AttributeSetInfo ::= SEQUENCE {
        commonInfo              [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        attributeSet            [1]     IMPLICIT AttributeSetId,
--Non-key brief elements follow:
        name                    [2]     IMPLICIT InternationalString,
--Non-brief elements follow:
                attributes      [3]     IMPLICIT SEQUENCE OF AttributeType OPTIONAL,
--Mandatory in full record
        description             [4] IMPLICIT HumanString OPTIONAL}
--AttributeType referenced in AttributeSetInfo
AttributeType ::= SEQUENCE {
        name                    [0] IMPLICIT InternationalString OPTIONAL,
        description             [1] IMPLICIT HumanString OPTIONAL,
        attributeType           [2] IMPLICIT INTEGER,
        attributeValues         [3] IMPLICIT SEQUENCE OF AttributeDescription}
AttributeDescription ::= SEQUENCE {
        name                    [0] IMPLICIT InternationalString OPTIONAL,
        description             [1] IMPLICIT HumanString OPTIONAL,
        attributeValue          [2] StringOrNumeric,
        equivalentAttributes    [3] IMPLICIT SEQUENCE OF StringOrNumeric OPTIONAL
-- See comment 4
        }
TermListInfo ::= SEQUENCE{
        commonInfo                      [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        databaseName            [1]     IMPLICIT DatabaseName,
--Non-key brief elements follow:
        termLists       [2] IMPLICIT SEQUENCE OF SEQUENCE{
        name            [1] IMPLICIT InternationalString,
        title           [2] IMPLICIT HumanString OPTIONAL,
        -- see comment 5
        searchCost      [3] IMPLICIT INTEGER {
        -- see comment 6
                                optimized       (0),
                                normal          (1),
                                expensive       (2),
                                filter          (3) } OPTIONAL,
        scanable        [4]     IMPLICIT BOOLEAN,
                                --      'true' means this list can be scanned
```

```
                        --  see comment 7
        broader         [5] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        narrower        [6] IMPLICIT SEQUENCE OF InternationalString OPTIONAL
                --Broader and narrower list alternative term lists related to this one.
                --The term lists so listed should also be in this termLists structure.
                        }
--No non-brief elements
}
ExtendedServicesInfo ::= SEQUENCE {
        commonInfo                      [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        type                            [1]     IMPLICIT OBJECT IDENTIFIER,
--Non-key brief elements follow:
        name                    [2]     IMPLICIT InternationalString OPTIONAL,
--Should be supplied if privateType is 'true'
        privateType                     [3]     IMPLICIT BOOLEAN,
        restrictionsApply       [5]     IMPLICIT BOOLEAN, -- if 'true' see 'description'
        feeApply                [6]     IMPLICIT BOOLEAN, -- if 'true' see 'description'
        available               [7]     IMPLICIT BOOLEAN,
        retentionSupported      [8]     IMPLICIT BOOLEAN,
        waitAction              [9]     IMPLICIT INTEGER{
                                        waitSupported           (1),
                                        waitAlways              (2),
                                        waitNotSupported        (3),
                                        depends                 (4),
                                        notSaying               (5)},
--Non-brief elements follow:
--To get brief plus 'description' use esn 'description'
        description             [10]    IMPLICIT HumanString OPTIONAL,
--To get above elements and 'specificExplain' use esn 'specificExplain'
        specificExplain         [11]    IMPLICIT EXTERNAL OPTIONAL,
--Use oid of specific ES, and select choice [3] 'explain'.
--Format to be developed in conjunction with the specific ES definition.
--To get all elements except 'specificExplain', use esn 'asn'
                esASN                   [12]    IMPLICIT InternationalString OPTIONAL
                        --The ASN.1 for this ES
        }
--Detail records
-- See comment 8
AttributeDetails ::= SEQUENCE {
        commonInfo                      [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        databaseName            [1]     IMPLICIT DatabaseName,
--Non-brief elements follow:
        attributesBySet         [2]     IMPLICIT SEQUENCE OF AttributeSetDetails OPTIONAL,
--Mandatory in full record
        attributeCombinations   [3]     IMPLICIT AttributeCombinations OPTIONAL}
--AttributeSetDetails referenced by AttributeDetails
AttributeSetDetails ::= SEQUENCE {
        attributeSet            [0]     IMPLICIT AttributeSetId,
        attributesByType        [1]     IMPLICIT SEQUENCE OF AttributeTypeDetails }
AttributeTypeDetails ::= SEQUENCE {
        attributeType           [0]     IMPLICIT INTEGER,
        defaultIfOmitted        [1]     IMPLICIT OmittedAttributeInterpretation OPTIONAL,
        attributeValues         [2]     IMPLICIT SEQUENCE OF AttributeValue OPTIONAL }
--If no attributeValues are supplied, all values of this type are fully supported,
--and the descriptions in AttributeSetInfo are adequate.
```

```
OmittedAttributeInterpretation ::= SEQUENCE {
        defaultValue            [0]     StringOrNumeric OPTIONAL,
--A default value is listed if that's how the server works
        defaultDescription      [1]     IMPLICIT HumanString OPTIONAL }
-- See comment 9
AttributeValue ::= SEQUENCE {
        value                   [0]     StringOrNumeric,
        description             [1]     IMPLICIT HumanString OPTIONAL,
                subAttributes   [2]     IMPLICIT SEQUENCE OF StringOrNumeric OPTIONAL,
                superAttributes [3]     IMPLICIT SEQUENCE OF StringOrNumeric OPTIONAL,
        partialSupport          [4]     IMPLICIT NULL OPTIONAL }
                -- See comment 10
TermListDetails ::= SEQUENCE{  -- one for each termList in TermListInfo
        commonInfo              [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        termListName            [1]     IMPLICIT InternationalString,
--Non-key elements (all non-brief) follow:
        description             [2]     IMPLICIT HumanString OPTIONAL,
        attributes              [3]     IMPLICIT AttributeCombinations OPTIONAL,
--Pattern for attributes that hit this list. Mandatory in full record
        scanInfo                [4]     IMPLICIT SEQUENCE {
        maxStepSize             [0]     IMPLICIT INTEGER OPTIONAL,
        collatingSequence       [1]     IMPLICIT HumanString OPTIONAL,
        increasing              [2]     IMPLICIT BOOLEAN OPTIONAL} OPTIONAL,
--Occurs only if list is scanable.
--If list is scanable and if scanInfo is omitted, server doesn't consider these important.
        estNumberTerms          [5]     IMPLICIT INTEGER OPTIONAL,
        sampleTerms             [6]     IMPLICIT SEQUENCE OF Term OPTIONAL}
ElementSetDetails ::= SEQUENCE {
        -- see comment 11
        commonInfo              [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        databaseName            [1]     IMPLICIT DatabaseName,
        elementSetName          [2]     IMPLICIT ElementSetName,
        recordSyntax            [3]     IMPLICIT OBJECT IDENTIFIER,
--Non-key Brief elements follow:
        schema                  [4]     IMPLICIT OBJECT IDENTIFIER,
--Non-brief elements follow:
        description             [5]     IMPLICIT HumanString OPTIONAL,
        detailsPerElement       [6]     IMPLICIT SEQUENCE OF PerElementDetails OPTIONAL
        --mandatory in full record
        }
RetrievalRecordDetails ::= SEQUENCE {
        commonInfo              [0]     IMPLICIT CommonInfo OPTIONAL,
-- Key elements follow:
        databaseName            [1]     IMPLICIT DatabaseName,
        schema                  [2]     IMPLICIT OBJECT IDENTIFIER,
        recordSyntax            [3]     IMPLICIT OBJECT IDENTIFIER,
--Non-brief elements follow:
        description             [4]     IMPLICIT HumanString OPTIONAL,
        detailsPerElement       [5]     IMPLICIT SEQUENCE OF PerElementDetails OPTIONAL
--Mandatory in full record
        }
--PerElementDetails is referenced in RetrievalRecordDetails and ElementSetDetails.
PerElementDetails ::= SEQUENCE {
                name            [0]     IMPLICIT InternationalString OPTIONAL,
--If the name is omitted, the record syntax's name for this element is appropriate.
```

```
        recordTag               [1]     IMPLICIT RecordTag OPTIONAL,
--The record tag may be omitted if tags are inappropriate
--for the record syntax, or if the client can be expected to know it for some other reason.
        schemaTags              [2]     IMPLICIT SEQUENCE OF Path OPTIONAL,
--The information from the listed schema elements is combined in some way to produce the data sent in the
--listed record tag. The 'contents' element below may describe the logic used.
        maxSize                 [3]     IMPLICIT INTEGER OPTIONAL,
        minSize                 [4]     IMPLICIT INTEGER OPTIONAL,
        avgSize                 [5]     IMPLICIT INTEGER OPTIONAL,
        fixedSize               [6]     IMPLICIT INTEGER OPTIONAL,
        repeatable              [8]     IMPLICIT BOOLEAN,
        required                [9]     IMPLICIT BOOLEAN,
--'required' really means that server will always supply the element
        description             [12]    IMPLICIT HumanString OPTIONAL,
        contents                [13]    IMPLICIT HumanString OPTIONAL,
        billingInfo             [14]    IMPLICIT HumanString OPTIONAL,
        restrictions            [15]    IMPLICIT HumanString OPTIONAL,
        alternateNames          [16]    IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        genericNames            [17]    IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        searchAccess            [18]    IMPLICIT AttributeCombinations OPTIONAL }
--RecordTag referenced in PerElementDetails above
RecordTag ::= SEQUENCE {
        qualifier               [0]     StringOrNumeric OPTIONAL,
--e.g. tag set for GRS-1
        tagValue                [1] StringOrNumeric}
SortDetails ::= SEQUENCE {
        commonInfo              [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        databaseName            [1]     IMPLICIT DatabaseName,
--No non-key brief elements
--Non-brief elements follow:
sortKeys                        [2]     IMPLICIT SEQUENCE OF SortKeyDetails OPTIONAL
--Mandatory in full record
        }
SortKeyDetails ::= SEQUENCE {
        description             [0]     IMPLICIT HumanString OPTIONAL,
        elementSpecifications   [1]     IMPLICIT SEQUENCE OF Specification OPTIONAL,
--Each specification is a way of specifying this same sort key
        attributeSpecifications [2]     IMPLICIT AttributeCombinations OPTIONAL,
--Each combination is a way of specifying this same sort key
        sortType        [3]     CHOICE {
                                        character       [0]     IMPLICIT NULL,
                                        numeric         [1]     IMPLICIT NULL,
                                        structured      [2]     IMPLICIT HumanString}
                                OPTIONAL,
        caseSensitivity         [4]     IMPLICIT INTEGER {
                                always  (0),    --always case-sensitive
                        never           (1),    --never case-sensitive
                        default-yes     (2),    --case-sensitivity is as specified on request,
                                                --and if not specified, case-sensitive
                        default-no      (3)}    --case-sensitivity is as specified on request,
                                                --and if not specified, not case-sensitive
                        OPTIONAL}
ProcessingInformation ::= SEQUENCE{
        commonInfo              [0]     IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        databaseName            [1]     IMPLICIT DatabaseName,
```

```
        processingContext       [2]     IMPLICIT INTEGER {
                                access                  (0),    --e.g. choosing databases
                                search                  (1),    --e.g. "search strategies" or search
                                                                --forms
                                retrieval               (2),    --e.g. recommended element
                                                                --combinations
                                record-presentation     (3),    --display of retrieved records
                                record-handling         (4)     --handling (e.g. saving) of retrieved
                                                                --records
                                },
        name                            [3]     IMPLICIT InternationalString,
        oid                             [4]     IMPLICIT OBJECT IDENTIFIER,
--No non-key brief elements
--Non-brief elements follow:
        description                     [5]     IMPLICIT HumanString OPTIONAL,
--Use element set name 'description' to retrieve all except instructions.
        instructions                    [6]     IMPLICIT EXTERNAL OPTIONAL
                                                --mandatory in full record
        }
VariantSetInfo ::= SEQUENCE {
        -- SEE COMMENT 12
        commonInfo                      [0] IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        variantSet                      [1] IMPLICIT OBJECT IDENTIFIER,
--Non-key brief elements follow:
        name                            [2] IMPLICIT InternationalString,
--Non-brief elements follow:
                variants                [3] IMPLICIT SEQUENCE OF VariantClass OPTIONAL
                                                --Mandatory in full record
        }
--Subsidiary structures for VariantSetInfo
VariantClass ::= SEQUENCE {
        name                    [0] IMPLICIT InternationalString OPTIONAL,
        description             [1] IMPLICIT HumanString OPTIONAL,
        variantClass            [2] IMPLICIT INTEGER,
        variantTypes            [3] IMPLICIT SEQUENCE OF VariantType}
VariantType ::= SEQUENCE {
        name                    [0] IMPLICIT InternationalString OPTIONAL,
        description             [1] IMPLICIT HumanString OPTIONAL,
        variantType             [2] IMPLICIT INTEGER,
        variantValue            [3] IMPLICIT VariantValue OPTIONAL}
VariantValue ::= SEQUENCE {
        dataType                [0] PrimitiveDataType,
        values                  [1] ValueSet OPTIONAL }
ValueSet ::= CHOICE {
        range                   [0] IMPLICIT ValueRange,
        enumerated              [1] IMPLICIT SEQUENCE OF ValueDescription }
ValueRange ::= SEQUENCE {
--At last one the following must be supplied, both may be supplied.
        lower                   [0] ValueDescription OPTIONAL,
        upper                   [1] ValueDescription OPTIONAL }
ValueDescription ::= CHOICE{
        integer                 INTEGER,
        string                  InternationalString,
        octets                  OCTET STRING,
        oid                     OBJECT IDENTIFIER,
        unit            [1]     IMPLICIT Unit,
```

```
        valueAndUnit    [2]     IMPLICIT IntUnit
--oid and unit can't be used in a ValueRange
        }
UnitInfo ::= SEQUENCE {
        commonInfo                      [0] IMPLICIT CommonInfo OPTIONAL,
--Key elements follow:
        unitSystem                      [1] IMPLICIT InternationalString OPTIONAL,
--Changed to OPTIONAL in 2001 version as result of defect report
--No non-key brief elements
--Non-brief elements follow:
        description             [2] IMPLICIT HumanString OPTIONAL,
        units                   [3] IMPLICIT SEQUENCE OF UnitType OPTIONAL
--Mandatory in full record
        }
--Subsidiary structures for UnitInfo
UnitType ::= SEQUENCE {
        name                            [0] IMPLICIT InternationalString OPTIONAL,
        description                     [1] IMPLICIT HumanString OPTIONAL,
        unitType                        [2] StringOrNumeric,
        units                           [3] IMPLICIT SEQUENCE OF Units}
Units ::= SEQUENCE {
        name                            [0] IMPLICIT InternationalString OPTIONAL,
        description                     [1] IMPLICIT HumanString OPTIONAL,
        unit                            [2] StringOrNumeric}
CategoryList ::= SEQUENCE {
        commonInfo                      [0] IMPLICIT CommonInfo OPTIONAL,
--Only one record expected per Explain database. All elements appear in brief presentation
        categories                      [1] IMPLICIT SEQUENCE OF CategoryInfo }
CategoryInfo ::= SEQUENCE {
        category                [1] IMPLICIT InternationalString,
        originalCategory        [2] IMPLICIT InternationalString OPTIONAL,
        description             [3] IMPLICIT HumanString OPTIONAL,
        asn1Module              [4] IMPLICIT InternationalString OPTIONAL}
--Subsidiary definitions
CommonInfo ::= SEQUENCE {
                dateAdded               [0] IMPLICIT GeneralizedTime OPTIONAL,
                dateChanged             [1] IMPLICIT GeneralizedTime OPTIONAL,
                expiry                  [2] IMPLICIT GeneralizedTime OPTIONAL,
                humanString-Language    [3] IMPLICIT LanguageCode OPTIONAL,
--Following not to occur for brief:
                otherInfo               OtherInformation OPTIONAL}
HumanString ::= SEQUENCE OF SEQUENCE {
        language                [0] IMPLICIT LanguageCode OPTIONAL,
        text                    [1] IMPLICIT InternationalString}
IconObject ::= SEQUENCE OF SEQUENCE{
--Note that the "SEQUENCE OF" is to allow alternative
--representations of the same Icon; it is not intended to allow multiple icons.
        bodyType        [1] CHOICE{
                                ianaType        [1] IMPLICIT InternationalString,
                                z3950type       [2] IMPLICIT InternationalString,
                                otherType       [3] IMPLICIT InternationalString},
        content         [2] IMPLICIT OCTET STRING}
LanguageCode ::= InternationalString        -- from ANSI/NISO Z39.53-1994
ContactInfo ::= SEQUENCE {
        name                    [0] IMPLICIT InternationalString OPTIONAL,
        description             [1] IMPLICIT HumanString OPTIONAL,
        address                 [2] IMPLICIT HumanString OPTIONAL,
```

```
        email                   [3] IMPLICIT InternationalString OPTIONAL,
        phone                   [4] IMPLICIT InternationalString OPTIONAL}
NetworkAddress ::= CHOICE {
        internetAddress         [0] IMPLICIT SEQUENCE {
                                        hostAddress     [0] IMPLICIT InternationalString,
                                        port            [1] IMPLICIT INTEGER},
        depricated              [1] IMPLICIT SEQUENCE {
                        depricated0     [0] IMPLICIT InternationalString,
                        depricated1     [1] IMPLICIT InternationalString OPTIONAL,
                        depricated2     [2] IMPLICIT InternationalString OPTIONAL,
                        depricated3     [3] IMPLICIT InternationalString},
-- This element depricated in Z39.50-2003
        other                   [2] IMPLICIT SEQUENCE {
                                        type    [0] IMPLICIT InternationalString,
                                        address [1] IMPLICIT InternationalString}}
AccessInfo ::= SEQUENCE {
        -- see comment 13
queryTypesSupported     [0] IMPLICIT SEQUENCE OF QueryTypeDetails OPTIONAL,
diagnosticsSets         [1] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
attributeSetIds         [2] IMPLICIT SEQUENCE OF AttributeSetId OPTIONAL,
schemas                 [3] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
recordSyntaxes          [4] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
resourceChallenges      [5] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
restrictedAccess        [6] IMPLICIT AccessRestrictions OPTIONAL,
costInfo                [8] IMPLICIT Costs OPTIONAL,
variantSets             [9] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
elementSetNames         [10] IMPLICIT SEQUENCE OF ElementSetName OPTIONAL,
unitSystems             [11] IMPLICIT SEQUENCE OF InternationalString}
--Begin auxiliary definitions for AccessInfo
--Begin Query Details
QueryTypeDetails ::= CHOICE {
        private                 [0]     IMPLICIT PrivateCapabilities,
        rpn                     [1]     IMPLICIT RpnCapabilities,
        iso8777                 [2]     IMPLICIT Iso8777Capabilities,
        z39-58                  [100]   IMPLICIT HumanString,
        erpn                    [101]   IMPLICIT RpnCapabilities,
        rankedList              [102]   IMPLICIT HumanString}
PrivateCapabilities ::= SEQUENCE {
        operators       [0] IMPLICIT SEQUENCE OF SEQUENCE {
                                operator        [0] IMPLICIT InternationalString,
                                description     [1] IMPLICIT HumanString OPTIONAL }
                                OPTIONAL,
        searchKeys      [1] IMPLICIT SEQUENCE OF SearchKey OPTIONAL,
                --field names that can be searched
        description     [2] IMPLICIT SEQUENCE OF HumanString OPTIONAL }
RpnCapabilities ::= SEQUENCE {
        operators       [0] IMPLICIT SEQUENCE OF INTEGER{
                                and             (0),
                                or              (1),
                                and-not         (2),
                                prox            (3)}OPTIONAL,
--Omitted means all operators are supported
        resultSetAsOperandSupported     [1] IMPLICIT BOOLEAN,
        restrictionOperandSupported     [2] IMPLICIT BOOLEAN,
                proximity               [3] IMPLICIT ProximitySupport OPTIONAL}
Iso8777Capabilities ::= SEQUENCE {
                searchKeys      [0] IMPLICIT SEQUENCE OF SearchKey,
```

```
                    -- field names that may be searched
        restrictions        [1] IMPLICIT HumanString OPTIONAL
--Omitted means supported, not specifying units
        }
ProximitySupport ::= SEQUENCE {
        anySupport                      [0]   IMPLICIT BOOLEAN,
            -- 'false' means no proximity support, in which case unitsSupported not supplied
unitsSupported  [1]   IMPLICIT SEQUENCE OF CHOICE{
                    known   [1] IMPLICIT INTEGER,
                    --Values from KnownProximityUnit
                    private   [2] IMPLICIT SEQUENCE{
                                unit            [0] IMPLICIT INTEGER,
                                description     [1] HumanString OPTIONAL}}
                                OPTIONAL}
SearchKey ::= SEQUENCE {
        searchKey               [0] IMPLICIT InternationalString,
        description             [1] IMPLICIT HumanString OPTIONAL }
        --End Query details

AccessRestrictions ::= SEQUENCE OF SEQUENCE {
        accessType              [0] INTEGER {
                    any                     (0),
                    search                  (1),
                    present                 (2),
                    specific-elements       (3),
                    extended-services       (4),
                    by-database             (5)},
        accessText              [1] IMPLICIT HumanString OPTIONAL,
        accessChallenges        [2] IMPLICIT SEQUENCE OF OBJECT IDENTIFIER OPTIONAL}
Costs ::= SEQUENCE {
        connectCharge           [0] IMPLICIT Charge OPTIONAL,--Per-connection charge
        connectTime             [1] IMPLICIT Charge OPTIONAL,--Time-based charge
        displayCharge           [2] IMPLICIT Charge OPTIONAL,--Per-record charge
        searchCharge            [3] IMPLICIT Charge OPTIONAL,--Per-search charge
        subscriptCharge         [4] IMPLICIT Charge OPTIONAL --Subscription charges
        otherCharges            [5] IMPLICIT SEQUENCE OF SEQUENCE{
                    forWhat     [1]     IMPLICIT HumanString,
                    charge      [2]     IMPLICIT Charge} OPTIONAL}
Charge ::= SEQUENCE{
        cost                            [1] IMPLICIT IntUnit,
        perWhat                 [2] IMPLICIT Unit OPTIONAL,
--e.g. "second," "minute," "line," "record"...
        text                            [3] IMPLICIT HumanString OPTIONAL}
--End Auxiliary definitions for AccessInfo
DatabaseList ::= SEQUENCE OF DatabaseName
AttributeCombinations ::= SEQUENCE {
        defaultAttributeSet             [0] IMPLICIT AttributeSetId,
                --Default for the combinations.
                --Also probably a good choice for the default in searches, but that isn't required.
                legalCombinations       [1] IMPLICIT SEQUENCE OF AttributeCombination }
AttributeCombination ::= SEQUENCE OF AttributeOccurrence
--An AttributeCombination is a pattern for legal combination of attributes
AttributeOccurrence ::= SEQUENCE {
--An AttributeOccurrence lists the legal values for a specific attribute type in a combination
        attributeSet            [0]     IMPLICIT AttributeSetId OPTIONAL,
        attributeType           [1]     IMPLICIT INTEGER,
        mustBeSupplied          [2]     IMPLICIT NULL OPTIONAL,
```

```
        attributeValues         CHOICE {
        any-or-none             [3] IMPLICIT NULL,
                                --All supported values are OK
        specific                [4] IMPLICIT SEQUENCE OF StringOrNumeric}}
--Only these values allowed
END
```

Comment 1

Element set name 'B' (brief) retrieves:
'commonInfo' (except for otherInfo within commonInfo)
Key elements
Other elements designated as 'non-key brief elements'
Esn 'description' retrieves brief elements as well as 'description', and specific additional descriptive elements if designated.
Element set name 'F' (full) retrieves all of the above, as well as those designated as "non-brief elements". Some elements designated as OPTIONAL may be mandatory in full records, and are so identified. (Note that all elements that are not part of the brief element set must be designated as OPTIONAL in the ASN.1, otherwise it would be illegal to omit them.)
Other esns are defined (below) as needed.
Info Records
Info records are mainly for software consumption. They describe individual entities within the server system:
The server itself
Individual databases
Schemas
Tag sets
Record syntaxes
Attribute sets
Term lists
Extended services
The information about each Schema, Tag Set, Record Syntax and Attribute Set should match the universal definitions of these items. The only exception is that a server may omit any items it doesn't support, for example the description of the bib-1 attribute set may omit attributes that the server does not support under any circumstances.
Databases that may be searched together can be listed in the dbCominations element of the TargetInfo record.

Comment 2

There should generally be only a single TargetInfo record for a given Explain database, so as a practical matter in most cases this record may be considered not to have a key. However in some cases a key may be useful, for example to allow the client to ascertain that it in fact connected to the right server, or is accessing the right Explain database. In any case, to search for the (single) TargetInfo record in an Explain database, a query need not include an operand with Use attribute serverName and term = server name. A single operand with Use attribute ExplainCategory and term = 'TargetInfo' is sufficient.

Comment 3

If explainDatabase is present, this database is the Explain database, or an Explain database for a different server, possibly on a different host. The means by which that server may be accessed is not addressed by this standard. One suggested possibility is an implementor agreement whereby the database name is a url which may be used to connect to the server.

Comment 4
Each occurrence of equivalentAttributes is an occurrence of 'attributeValue' from AttributeDescription for a different attribute. Equivalences listed here should be derived from the attribute set definition, not from a particular server's behavior.

Comment 5
Title is for users to see and can differ by language. Name, on the other hand is typically a short string not necessarily meant to be human-readable, and not variable by language.

Comment 6
Values of searchCost are:
Optimized: The attribute (or combination) associated with this list will do fast searches.
Normal: The attribute (combination) will work as expected. So there's probably an index for the attribute (combination) or some similar mechanism.
Expensive: Can use the attribute (combination), but it might not provide satisfactory results. Probably there is no index, or post-processing of records is required.
Filter: Can't search with this attribute (combination) alone.

Comment 7
A term list might exist to optimize searching (i.e. the so-called term list is actually an index) even though it is not scanable. For example, a server maintaining an index of social security numbers might not support scanning that index.

Comment 8
The detail records describe relationships among entities supported by the server. RetrievalRecordDetails describes the way that schema elements are mapped into record elements. This mapping may be different for each combination of database, schema, record syntax. The per-element details describe the default mapping.
Client-request re-tagging can change that mapping. When multiple databases are listed in a databaseNames element, the record applies equally to all of the listed databases. This is unrelated to searching the databases together. AttributeDetails describes how databases can be searched. Each supported attribute is listed, and the allowable combinations can be described.

Comment 9
The human-readable description should generally be provided
It is legal for both default elements to be missing, which means that the server will allow the attribute type to be omitted, but isn't saying what it will do

Comment 10
partialSupport indicates that an attributeValue is accepted, but may not be processed in the "expected" way. One important reason for this is composite databases: in this case partialSupport may indicate that only some of the subDbs support the attribute, and others ignore it.

Comment 11
ElementSetDetails describes the way that database records are mapped to record elements. This mapping may be different for each combination of database name and element set. The database record description is a schema, which may be private to the server. The schema's abstract record structure and tag sets provide the vocabulary for discussing record content; their presence in the Explain database does not imply support for complex retrieval specification.

Comment 12
A record in the VariantSetInfo category describes a variant set definition, i.e. classes, types, and values, for a specific variant set definition supported by the server. Support by the server of a particular variant set definition does not imply that the definition is supported for any specific database or element.

Comment 13
AccessInfo contains the fundamental information about what facilities are required to use this server or server. For example, if a client can handle none of the record syntaxes a database can provide, it might choose not to access the database.

ASN1.5 Z39.50 ASN.1 Definition for SUTRS

```
RecordSyntax-SUTRS
{Z39-50-recordSyntax sutrs (101)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString  FROM Z39-50-APDU-2001;
SutrsRecord ::=  InternationalString
-- See Comment 1.
END
```

Comment 1
This use of InternationalString should be interpreted to mean GeneralString
Line terminator is ASCII LF (X'0A')
Recommended maximum line length is 72 characters

ASN1.6 Z39.50 ASN.1 Definition for GRS-1

RecordSyntax-generic

```
-- For detailed semantics, see Appendix RET.
{Z39-50-recordSyntax grs-1 (105)} DEFINITIONS ::=
BEGIN
EXPORTS Variant;
IMPORTS IntUnit, Unit, InternationalString, StringOrNumeric, Term FROM Z39-50-APDU-2001;
GenericRecord ::= SEQUENCE OF TaggedElement
TaggedElement ::= SEQUENCE {
        tagType                 [1] IMPLICIT INTEGER OPTIONAL,
--If omitted, default should be supplied dynamically by tagSet-M;
--otherwise it should be statically specified by the schema.
        tagValue                [2] StringOrNumeric,
        tagOccurrence           [3] IMPLICIT INTEGER OPTIONAL,
--Occurrence within the database record, and relative to the parent.
--No default; if omitted, server not telling or it is irrelevant.
-- 1-based. Tags are numbered beginning with 1.
        content                 [4] ElementData,
        metaData                [5] IMPLICIT ElementMetaData OPTIONAL,
        appliedVariant          [6] IMPLICIT Variant OPTIONAL}
ElementData ::= CHOICE{
        octets          OCTET STRING,
        numeric         INTEGER,
        date            GeneralizedTime,
        ext             EXTERNAL,
        string          InternationalString,
--Note: VisibleString (ASN.1 tag 26) is not supported by GRS-1.
--The tag for 'string' must always be 2, even when
--version 2 is in effect and the character repertoire collapses to that of VisibleString.
        trueOrFalse                     BOOLEAN,
```

```
        oid                             OBJECT IDENTIFIER,
        intUnit                 [1]     IMPLICIT IntUnit,
        elementNotThere         [2] IMPLICIT NULL,
                                        -- Element requested but not there
        elementEmpty            [3] IMPLICIT NULL,
                                        --element there, but empty
        noDataRequested         [4] IMPLICIT NULL,
                                        --Variant request said 'no data'
        diagnostic              [5] IMPLICIT EXTERNAL,
        subtree                 [6] SEQUENCE OF TaggedElement
                                        -- Recursive, for nested tags
        }
ElementMetaData ::= SEQUENCE{
        seriesOrder                     [1]     IMPLICIT Order OPTIONAL,
                                                        --Only for a non-leaf node
        usageRight                      [2]     IMPLICIT Usage OPTIONAL,
        hits                            [3]     IMPLICIT SEQUENCE OF HitVector OPTIONAL,
        displayName                     [4]     IMPLICIT InternationalString OPTIONAL,
            --Name for element that client can use for display
        supportedVariants               [5]     IMPLICIT SEQUENCE OF Variant OPTIONAL,
        message                         [6]     IMPLICIT InternationalString OPTIONAL,
        elementDescriptor               [7]     IMPLICIT OCTET STRING OPTIONAL,
            --For example, a DTD
        surrogateFor                    [8]     IMPLICIT TagPath OPTIONAL,
            --The retrieved element is a surrogate for the element given by this path
        surrogateElement        [9]     IMPLICIT TagPath OPTIONAL,
            --The element given by this path is a surrogate for the retrieved element
            -- See comment 1.
        other                           [99]    IMPLICIT EXTERNAL OPTIONAL}


TagPath ::= SEQUENCE  OF SEQUENCE{
        tagType                 [1]     IMPLICIT INTEGER OPTIONAL,
        tagValue                [2]     StringOrNumeric,
        tagOccurrence           [3]     IMPLICIT INTEGER OPTIONAL}
Order ::= SEQUENCE{
        ascending               [1] IMPLICIT BOOLEAN,
            --"true" means monotonically increasing (i.e. non-decreasing);
            --"false" means monotonically decreasing (i.e. non-increasing).
        order                           [2] IMPLICIT INTEGER
            --Same as defined by 'elementOrdering' in tagSet-M, though this may be
            --overridden by schema.
        }
Usage ::= SEQUENCE      {
        type            [1] IMPLICIT INTEGER{
                        redistributable     (1), --Element is freely redistributable
                        restricted          (2), --Restriction contains statement
                        licensePointer      (3) --Restriction contains license pointer
                        },
        restriction     [2] IMPLICIT InternationalString OPTIONAL}
HitVector ::= SEQUENCE{
--Each hit vector points to a fragment within the element, via location and/or token.
        satisfier               Term OPTIONAL,  -- sourceword, etc.
        offsetIntoElement       [1] IMPLICIT IntUnit OPTIONAL,
        length                  [2] IMPLICIT IntUnit OPTIONAL,
        hitRank                 [3] IMPLICIT INTEGER OPTIONAL,
        serverToken             [4] IMPLICIT OCTET STRING OPTIONAL
```

```
            --Client may use token subsequently within a variantRequest
            --(in an elementRequest) to retrieve (or to refer to) the fragment.
        }
Variant ::= SEQUENCE{
        globalVariantSetId      [1]     IMPLICIT OBJECT IDENTIFIER OPTIONAL,
            --Applies to the triples below, when variantSetId omitted.
            --If globalVariantSetId omitted, default applies.
            --Default may be provided by the tagSet-M element defaultVariantSetId.
        triples                 [2]     IMPLICIT SEQUENCE OF SEQUENCE{
        variantSetId            [0]     IMPLICIT OBJECT IDENTIFIER OPTIONAL,
                --If omitted, globalVariantSetId (above) applies,
                --unless that too is omitted, in which case, default used.
                    class       [1]     IMPLICIT INTEGER,
                    type        [2]     IMPLICIT INTEGER,
                    value       [3]     CHOICE{
                                                integer INTEGER,
                                                string  InternationalString,
                                                octets  OCTET STRING,
                                                oid     OBJECT IDENTIFIER,
                                                bool    BOOLEAN,
                                                null    NULL,
                --Following need context tags:
                    unit        [1] IMPLICIT Unit,
                valueAndUnit    [2] IMPLICIT IntUnit}}}
                    END
```

Comment 1

surrogateFor and surrogateElement are for use when a record contains a number of elements. For example, suppose the elements are objects and one particular object is a surrogate (e.g. thumbnail) for another element within that record. Suppose for element A surrogateFor is supplied; it is supplied in the form of a tagPath, suppose for element B. This means that element A is a surrogate for element B. Or suppose for element A, surrogateElement is supplied, in the form of a tagPath for element B. This means that element B is a surrogate for element A. (Note that these two are useful only when both the element and its surrogate are in the same record.)

ASN1.7 Z39.50 ASN.1 Definition for the Extended Services Task Package Record Syntax

```
RecordSyntax-ESTaskPackage
{Z39-50-recordSyntax esTaskPackage (106)} DEFINITIONS ::=
BEGIN
IMPORTS Permissions, InternationalString, IntUnit, DiagRec FROM Z39-50-APDU-2001;
TaskPackage ::= SEQUENCE{
        packageType             [1]     IMPLICIT OBJECT IDENTIFIER,
            --oid of specific ES definition
        packageName             [2]     IMPLICIT InternationalString OPTIONAL,
        userId                  [3]     IMPLICIT InternationalString OPTIONAL,
        retentionTime           [4]     IMPLICIT IntUnit OPTIONAL,
        permissions             [5]     IMPLICIT Permissions OPTIONAL,
        description             [6]     IMPLICIT InternationalString OPTIONAL,
        serverReference         [7]     IMPLICIT OCTET STRING OPTIONAL,
        creationDateTime        [8]     IMPLICIT GeneralizedTime OPTIONAL,
        taskStatus              [9]     IMPLICIT INTEGER{
                                                pending         (0),
```

```
                                        active          (1),
                                        complete        (2),
                                        aborted         (3)},
        packageDiagnostics      [10]    IMPLICIT SEQUENCE OF DiagRec OPTIONAL,
        taskSpecificParameters  [11]    IMPLICIT EXTERNAL
            --Use oid for specific ES definition
            --(same oid as packageType above) and select [2] "taskPackage."
        }
        END
```

ASN1.8 Z39.50 ASN.1 Definition for Resource Report Format Resource-2

```
ResourceReport-Format-Resource-2
{Z39-50-resourceReport resource-2 (2)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString, StringOrNumeric, IntUnit FROM Z39-50-APDU-2001;
ResourceReport ::= SEQUENCE{
        estimates       [1]     IMPLICIT SEQUENCE OF Estimate OPTIONAL,
        message         [2]     IMPLICIT InternationalString OPTIONAL}
Estimate ::= SEQUENCE{
        type            [1]     StringOrNumeric,
            --Numeric values of 1-16 are the same as used in Resource-1.
            --See Z39.50-1995 Appendix 6, RSC.1
        value           [2]     IMPLICIT IntUnit
-- When expressing currency: unitSystem (of Unit) is 'z3950'
-- (case insensitive), and unitType is 'iso4217-1990' (case insensitive).
-- Unit is currency code from ISO 4217-1990
        }
        END
```

ASN1.9 Z39.50 ASN.1 Definition for Access Control Formats

ASN1.9.1 Prompt-1

```
AccessControlFormat-prompt-1
{Z39-50-accessControl prompt-1 (1)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString, DiagRec FROM Z39-50-APDU-2001;
PromptObject ::= CHOICE{
        challenge               [1] IMPLICIT Challenge,
        response                [2] IMPLICIT Response}
Challenge ::= SEQUENCE OF SEQUENCE {
        promptId                [1] PromptId,
            -- See comment 1
        defaultResponse         [2] IMPLICIT InternationalString OPTIONAL,
        promptInfo              [3] CHOICE{
        character               [1] IMPLICIT InternationalString,
        encrypted               [2] IMPLICIT Encryption} OPTIONAL,

                -- See comment 2
        regExpr                 [4] IMPLICIT InternationalString OPTIONAL,
                -- See comment 3
        responseRequired        [5] IMPLICIT NULL OPTIONAL,
        allowedValues           [6] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
            --e.g. promptId="Desired color"; allowed = 'red', 'blue','Green'
```

```
        shouldSave                      [7] IMPLICIT NULL OPTIONAL,
                -- See comment 4
        dataType                [8] IMPLICIT INTEGER{
                                        integer         (1),
                                        date            (2),
                                        float           (3),
                                        alphaNumeric    (4),
                                        url-urn         (5),
                                        boolean         (6)} OPTIONAL,
                -- See comment  5
        diagnostic                      [9] IMPLICIT EXTERNAL OPTIONAL
                --Intended for repeat requests when there is an error
                --the client should report to the user from previous attempt.
        }
Response ::= SEQUENCE OF SEQUENCE {
        promptId                [1] PromptId,
                -- See comment 6
        promptResponse          [2] CHOICE{
        string                  [1] IMPLICIT InternationalString,
        accept                  [2] IMPLICIT BOOLEAN,
        acknowledge             [3] IMPLICIT NULL,
        diagnostic              [4] DiagRec,
        encrypted               [5] IMPLICIT Encryption}}
PromptId ::= CHOICE{
        enummeratedPrompt               [1] IMPLICIT SEQUENCE{
                        type                    [1] IMPLICIT INTEGER{
                                                        groupId         (0),
                                                        userId          (1),
                                                        password        (2),
                                                        newPassword     (3),
                                                        copyright       (4),
                                                                -- See comment 7
                                                        sessionId       (5)},
                        suggestedString         [2] IMPLICIT InternationalString OPTIONAL},
        nonEnumeratedPrompt     [2] IMPLICIT InternationalString}
Encryption ::= SEQUENCE{
        cryptType       [1] IMPLICIT OCTET STRING OPTIONAL,
        credential      [2] IMPLICIT OCTET STRING OPTIONAL,
             --Random number, SALT, or other factor
        data            [3] IMPLICIT OCTET STRING}
        END
```

Comment 1
Server supplies a number (for an enumerated prompt) or string (for a non-enumerated prompt), for each prompt, and the client returns it in response, for this prompt, so server may correlate the prompt response with the prompt.

Comment 2
Information corresponding to an enumerated prompt. For example if 'type', within PromptId, is 'copyright', then promptInfo may contain a copyright statement

Comment 3
A regular expression that promptResponse should match. See IEEE 1003.2 Volume 1, Section 2.8 "Regular Expression Notation." For example if promptId is "Year of publication," regExpr might be "19[89][0-9]|20[0-9][0-9]".

Comment 4
Server recommends that client save the data that it prompts from the user corresponding to this prompt, because it is likely to be requested again (so client might not have to prompt the user next time).

Comment 5
Server telling client type of data it wants. E.g., if "date" is specified, presumably the client will try to prompt something "date-like" from the user.

Comment 6
Corresponds to a prompt in the challenge, or may be unprompted. For example "newPassword," if unprompted, should be "enumerated." If this responds to a non-enumerated prompt, then nonEnumeratedPrompt should contain the prompt string from the challenge.

Comment 7
When type on Challenge is 'copyright', promptInfo has text of copyright message to be displayed verbatim to the user. If promptResponse indicates 'acceptance', this indicates the user has been shown, and accepted, the terms of the copyright. This is not intended to be legally binding, but provides a good-faith attempt on the part of the server to inform the user of the copyright.

ASN1.9.2 Des-1

```
AccessControlFormat-des-1
{Z39-50-accessControlFormat  des-1 (2)} DEFINITIONS ::=
BEGIN
DES-RN-Object ::= CHOICE {
        challenge               [1] IMPLICIT DRNType,
        response                [2] IMPLICIT DRNType}
DRNType ::= SEQUENCE{
        userId                  [1] IMPLICIT OCTET STRING OPTIONAL,
        salt                    [2] IMPLICIT OCTET STRING OPTIONAL,
        randomNumber            [3] IMPLICIT OCTET STRING}
        END
```

ASN1.9.3 Krb-1

```
AccessControlFormat-krb-1
{Z39-50-accessControlFormat  krb-1 (3)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString FROM Z39-50-APDU-2001;
KRBObject ::= CHOICE {
        challenge       [1] IMPLICIT KRBRequest,
        response        [2] IMPLICIT KRBResponse}
KRBRequest ::= SEQUENCE{
        service         [1] IMPLICIT InternationalString,
        instance        [2] IMPLICIT InternationalString OPTIONAL,
        realm           [3] IMPLICIT InternationalString OPTIONAL}
            --Server requests a ticket for the given service, instance, and realm
KRBResponse ::= SEQUENCE{
        userid          [1] IMPLICIT InternationalString OPTIONAL,
        ticket          [2] IMPLICIT OCTET STRING}
            -- Client responds with a ticket for the requested service
END
```

ASN1.10 Z39.50 ASN.1 Definitions for Extended Services

```
ESFormat-PersistentResultSet
{Z39-50-extendedService persistentResultSet (1)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString FROM Z39-50-APDU-2001;
PersistentResultSet ::= CHOICE{
        esRequest       [1] IMPLICIT SEQUENCE{
                                toKeep          [1] IMPLICIT NULL,
                                notToKeep       [2] ClientPartNotToKeep OPTIONAL},
        taskPackage     [2] IMPLICIT SEQUENCE{
                                clientPart      [1] IMPLICIT NULL,
                                serverPart      [2] ServerPart OPTIONAL}}
ClientPartNotToKeep ::= SEQUENCE{
        clientSuppliedResultSet [1] IMPLICIT InternationalString OPTIONAL,
            --Name of transient result set, supplied on request,
            -- mandatory unless function is 'delete'
        replaceOrAppend         [2] IMPLICIT INTEGER{ -- Only if function is "modify"
                        replace         (1),
                        append          (2)} OPTIONAL}
ServerPart ::= SEQUENCE{
        serverSuppliedResultSet [1] IMPLICIT InternationalString OPTIONAL,
                --Name of transient result set, supplied by server,
                 --representing the persistent result set to which
                --package pertains. Meaningful only when package is
                --presented. (i.e. not on ES response).
        NumberOfRecords         [2] IMPLICIT INTEGER OPTIONAL}
        END


ESFormat-PersistentQuery
{Z39-50-extendedService persistentQuery (2)} DEFINITIONS ::=
BEGIN
IMPORTS Query, InternationalString, OtherInformation FROM Z39-50-APDU-2001;
PersistentQuery ::= CHOICE{
        esRequest       [1] IMPLICIT SEQUENCE{
                        toKeep          [1] ClientPartToKeep OPTIONAL,
                        notToKeep       [2] ClientPartNotToKeep},
        taskPackage     [2] IMPLICIT SEQUENCE{
                        clientPart      [1] ClientPartToKeep OPTIONAL,
                        serverPart      [2] ServerPart}}
ClientPartToKeep ::= SEQUENCE{
                dbNames         [2] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        additionalSearchInfo    [3] OtherInformation OPTIONAL}
ClientPartNotToKeep ::= CHOICE{
                package         [1] IMPLICIT InternationalString,
                query           [2] Query}
ServerPart ::= Query
END


ESFormat-PeriodicQuerySchedule
{Z39-50-extendedService periodicQuerySchedule (3)} DEFINITIONS ::=
BEGIN
IMPORTS Query, InternationalString, IntUnit FROM Z39-50-APDU-2001
ExportSpecification, Destination FROM ESFormat-ExportSpecification;
PeriodicQuerySchedule ::= CHOICE{
```

```
        esRequest       [1] IMPLICIT SEQUENCE{
                        toKeep          [1] ClientPartToKeep,
                        notToKeep       [2] ClientPartNotToKeep},
        taskPackage     [2] IMPLICIT SEQUENCE{
                        clientPart      [1] ClientPartToKeep,
                        serverPart      [2] ServerPart}}
ClientPartToKeep ::=SEQUENCE{
        activeFlag              [1] IMPLICIT BOOLEAN,
        databaseNames           [2] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
                --databaseNames must not occur if option bit 20 is set
        resultSetDisposition    [3] IMPLICIT INTEGER{
                                replace         (1),
                                append          (2),
                                createNew       (3)
                                --Only if client and server have agreement about naming
                                --convention for the resulting package,
                                --and only if no result set is specified
                        } OPTIONAL,
                        --Mandatory on 'create' if result set is specified,
                        --in which case it must be 'replace' or 'append
        alertDestination        [4] Destination OPTIONAL,
        exportParameters        [5] CHOICE{
                                packageName     [1]  IMPLICIT InternationalString,
                                exportPackage   [2] ExportSpecification} OPTIONAL}
ClientPartNotToKeep ::= SEQUENCE{
                databaseNames   [0] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
                        --Must not occur unless option bit 20 is set
        querySpec                       [1] CHOICE{
                        actualQuery     [1] Query,
                        packageName     [2] IMPLICIT InternationalString} OPTIONAL,
                                --Mandatory for 'create'
                clientSuggestedPeriod   [2] Period OPTIONAL,
                                -- mandatory for 'create'
                expiration              [3] IMPLICIT GeneralizedTime OPTIONAL,
                resultSetPackage        [4] IMPLICIT InternationalString OPTIONAL,
                additionalSearchInfo    [5] OtherInformation OPTIONAL
                        --Must not occur unless option bit 20 is set
        }
ServerPart ::= SEQUENCE{
                databaseNames   [0] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
                        --databaseNames must occur if bit 20 is set
                        -- and must not occur if option bit 20 is not set
        actualQuery             [1] Query,
        serverStatedPeriod      [2] Period,
                --Server supplies the period, which may be same as client proposed
        expiration              [3] IMPLICIT GeneralizedTime OPTIONAL,
                --Server supplies value for task package.
                --It may be the same as client proposed or different from
                --(and overrides) client proposal, but if omitted, there is no expiration.
        resultSetPackage        [4] IMPLICIT InternationalString OPTIONAL,
                --May be omitted only if exportParameters was supplied.
                --Server supplies same name as client supplied, if client did supply a name.
        lastQueryTime           [5] IMPLICIT GeneralizedTime OPTIONAL,
        lastResultNumber        [6] IMPLICIT INTEGER OPTIONAL,
                --Above two were made optional in 2001 version,
                --because there won't be any value for these between
                --the time the package is created and the first query is executed.
```

```
        numberSinceModify       [7] IMPLICIT INTEGER OPTIONAL,
        additionalSearchInfo    [8] OtherInformation OPTIONAL
                --Must not occur unless option bit 20 is set
        }
Period ::= CHOICE{
        unit                    [1] IMPLICIT IntUnit,
        businessDaily           [2] IMPLICIT NULL,
        continuous              [3] IMPLICIT NULL,
        other                   [4] IMPLICIT InternationalString}
        END


ESFormat-ItemOrder
{Z39-50-extendedService itemOrder (4)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString FROM Z39-50-APDU-2001;
ItemOrder ::= CHOICE{
        esRequest       [1] IMPLICIT SEQUENCE{
                        toKeep  [1] ClientPartToKeep OPTIONAL,
                        notToKeep       [2] ClientPartNotToKeep},
        taskPackage     [2] IMPLICIT SEQUENCE{
                        clientPart      [1] ClientPartToKeep OPTIONAL,
                        serverPart      [2] ServerPart}}
ClientPartToKeep ::= SEQUENCE{
        supplDescription [1] IMPLICIT EXTERNAL OPTIONAL,
        contact         [2] IMPLICIT SEQUENCE{
                                name    [1] IMPLICIT InternationalString OPTIONAL,
                                phone   [2] IMPLICIT InternationalString OPTIONAL,
                                email   [3] IMPLICIT InternationalString OPTIONAL} OPTIONAL,
        addlBilling     [3] IMPLICIT SEQUENCE{
                        paymentMethod           [1]   CHOICE{
                                billInvoice                     [0] IMPLICIT NULL,
                                prepay                          [1] IMPLICIT NULL,
                                depositAccount                  [2] IMPLICIT NULL,
                                creditCard                      [3] IMPLICIT CreditCardInfo,
                                cardInfoPreviouslySupplied      [4] IMPLICIT NULL,
                                privateKnown                    [5] IMPLICIT NULL,
                                privateNotKnown                 [6] IMPLICIT EXTERNAL},
                        customerReference       [2] IMPLICIT InternationalString OPTIONAL,
                                --An identifier assigned by the client
                                --to identify the customer.
                                --It could be used when the client want
                                --to search for Item Order task packages
                                --for a specific customer.
                        customerPONumber        [3] IMPLICIT InternationalString OPTIONAL
                        --A purchase order number assigned by the
                        --customer (as opposed to one that might be
                        --assigned by the supplier). Similarly, a client
                        --may search for a task package knowing
                        --only the customer reference.
                    }
                    OPTIONAL}
CreditCardInfo    ::= SEQUENCE{
        nameOnCard              [1] IMPLICIT InternationalString,
        expirationDate          [2] IMPLICIT InternationalString,
        cardNumber              [3] IMPLICIT InternationalString}
ClientPartNotToKeep ::= SEQUENCE{
```

```
--Corresponds to 'requestedItem' in service definition
--Must supply at least one, and may supply both.
        resultSetItem       [1] IMPLICIT SEQUENCE{
                            resultSetId         [1] IMPLICIT InternationalString,
                            item                [2] IMPLICIT INTEGER} OPTIONAL,
        itemRequest         [2] IMPLICIT EXTERNAL OPTIONAL
        -- See comment  1.
        }
ServerPart ::= SEQUENCE{
        itemRequest                 [1]     IMPLICIT EXTERNAL OPTIONAL,
            --When itemRequest is an ILL-Request APDU,
            --use OID 1.0.10161.2.1 (as above)
        statusOrErrorReport         [2] IMPLICIT EXTERNAL OPTIONAL,
            --When statusOrErrorReport is an ILL Status-Or-Error-Report
            --APDU, use OID 1.0.10161.2.1 (as above)
        auxiliaryStatus             [3] IMPLICIT INTEGER{
                                    notReceived             (1),
                                    loanQueue               (2),
                                    forwarded               (3),
                                    unfilledCopyright       (4),
                                    filledCopyright         (5)} OPTIONAL}
        END
```

Comment 1

When itemRequest is an ILL-Request APDU, use OID {iso standard 10161 abstract-syntax (2) ill-APDUs (1)}. Note that the 'itemRequest', as EXTERNAL, is not constrained (by Z39.50) to be ASN.1 BER. If the itemRequest is an ILL APDU (and the external uses OID 1.0.10161.2.1) then it is constrained to be ASN.1, but not constrained to be BER; it can be EDIFACT, as provided by the ILL standard. If BER is used, the BER encoding may be wrapped in Base64 (in store and forward mode).

```
ESFormat-Update
{Z39-50-extendedService update (5)} DEFINITIONS ::=
BEGIN
IMPORTS DiagRec, InternationalString FROM Z39-50-APDU-2001;
Update ::= CHOICE{
        esRequest       [1] IMPLICIT SEQUENCE{
                        toKeep  [1] ClientPartToKeep,
                        notToKeep       [2] ClientPartNotToKeep},
        taskPackage     [2] IMPLICIT SEQUENCE{
                        clientPart      [1] ClientPartToKeep,
                        serverPart      [2] ServerPart}}
ClientPartToKeep ::= SEQUENCE{
        action          [1] IMPLICIT INTEGER{
                                    recordInsert    (1),
                                    recordReplace   (2),
                                    recordDelete    (3),
                                    elementUpdate   (4)},
        databaseName    [2] IMPLICIT InternationalString,
        schema          [3] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
        elementSetName  [4] IMPLICIT InternationalString OPTIONAL}
ClientPartNotToKeep ::= SuppliedRecords
ServerPart ::= SEQUENCE{
        updateStatus            [1] IMPLICIT INTEGER{
                                    success  (1),
                                    partial  (2),
                                    failure  (3)},
```

```
        globalDiagnostics       [2] IMPLICIT SEQUENCE OF DiagRec OPTIONAL,
                --These are non-surrogate diagnostics relating
                --to the task, not to individual records.
                taskPackageRecords      [3] IMPLICIT SEQUENCE OF TaskPackageRecordStructure
        --See comment 1.
        }
--Auxiliary definitions for Update
SuppliedRecords ::= SEQUENCE OF SEQUENCE{
        recordId        [1] CHOICE{
                                number  [1] IMPLICIT INTEGER,
                                string  [2] IMPLICIT InternationalString,
                                opaque  [3] IMPLICIT OCTET STRING} OPTIONAL,
supplementalId          [2] CHOICE{
                                timeStamp       [1] IMPLICIT GeneralizedTime,
                                versionNumber   [2] IMPLICIT InternationalString,
                                previousVersion [3] IMPLICIT EXTERNAL} OPTIONAL,
correlationInfo         [3] IMPLICIT CorrelationInfo OPTIONAL,
record                  [4] IMPLICIT EXTERNAL}
CorrelationInfo ::= SEQUENCE{
--Client may supply one or both for any record:
        note    [1] IMPLICIT InternationalString OPTIONAL,
        id      [2] IMPLICIT INTEGER OPTIONAL}
TaskPackageRecordStructure ::= SEQUENCE{
        recordOrSurDiag         [1] CHOICE {
                                record  [1] IMPLICIT EXTERNAL,
--Choose 'record' if recordStatus is 'success', and elementSetName was supplied.
                                diagnostic      [2] DiagRec
-- Choose 'diagnostic', if RecordStatus is failure
                        } OPTIONAL,
        --See comment 2
        correlationInfo         [2] IMPLICIT CorrelationInfo OPTIONAL,
-- This should be included if it was supplied by the client
        recordStatus            [3] IMPLICIT INTEGER{
                                success         (1),
                                queued          (2),
                                inProcess       (3),
                                failure         (4)}}
        END
```

Comment 1
There should be a TaskPackageRecordStructure for every record supplied. The server should create such a structure for every record immediately upon creating the task package to include correlation information and status. The record itself would not be included until processing for that record is complete.

Comment 2
The parameter recordOrSurDiag will thus be omitted only if 'elementSetName' was omitted and recordStatus is 'success'; or if record status is 'queued' or in 'process'

```
ESFormat-ExportSpecification
{Z39-50-extendedService exportSpecification (6)} DEFINITIONS ::=
BEGIN
EXPORTS ExportSpecification, Destination; IMPORTS CompSpec, InternationalString FROM
Z39-50-APDU-2001;
ExportSpecification ::= CHOICE{
        esRequest       [1] IMPLICIT SEQUENCE{
                        toKeep          [1] ClientPartToKeep,
```

```
                        notToKeep       [2] IMPLICIT NULL},
            taskPackage [2] IMPLICIT SEQUENCE{
                        clientPart      [1] ClientPartToKeep,
                        serverPart      [2] IMPLICIT NULL}}
    ClientPartToKeep ::= SEQUENCE{
            composition             [1] IMPLICIT CompSpec,
            exportDestination       [2] Destination}
    Destination ::= CHOICE{
            phoneNumber             [1]  IMPLICIT InternationalString,
            faxNumber               [2]  IMPLICIT InternationalString,
            x400address             [3]  IMPLICIT InternationalString,
            emailAddress            [4]  IMPLICIT InternationalString,
            pagerNumber             [5]  IMPLICIT InternationalString,
            ftpAddress              [6]  IMPLICIT InternationalString,
            ftamAddress             [7]  IMPLICIT InternationalString,
            printerAddress          [8]  IMPLICIT InternationalString,
            other                   [100]  IMPLICIT SEQUENCE{
                                    vehicle         [1] IMPLICIT InternationalString   OPTIONAL,
                                    destination     [2] IMPLICIT InternationalString}}
    END


    ESFormat-ExportInvocation
    {Z39-50-extendedService exportInvocation (7)} DEFINITIONS ::=
    BEGIN
    IMPORTS InternationalString, IntUnit FROM Z39-50-APDU-2001
    ExportSpecification FROM ESFormat-ExportSpecification;
    ExportInvocation ::= CHOICE{
            esRequest       [1] IMPLICIT SEQUENCE{
                            toKeep          [1] ClientPartToKeep,
                            notToKeep       [2] ClientPartNotToKeep},
            taskPackage     [2] IMPLICIT SEQUENCE{
                            clientPart      [1] ClientPartToKeep,
                            serverPart      [2] ServerPart OPTIONAL}}
    ClientPartToKeep ::= SEQUENCE{
            exportSpec      [1] CHOICE{
                            packageName     [1] IMPLICIT InternationalString,
                            packageSpec     [2] ExportSpecification},
            numberOfCopies [2] IMPLICIT INTEGER}
    ClientPartNotToKeep     ::= SEQUENCE{
            resultSetId             [1] IMPLICIT InternationalString,
            records                 [2] CHOICE{
                                    all     [1] IMPLICIT NULL,
                                    ranges  [2] IMPLICIT SEQUENCE OF SEQUENCE{
                                            start   [1] IMPLICIT INTEGER,
                                            count   [2] IMPLICIT INTEGER OPTIONAL
                                    --Count may be omitted only on last range,
                                    -- to indicate "all remaining records beginning with 'start'."
                                }}}
    ServerPart      ::= SEQUENCE{
            estimatedQuantity       [1] IMPLICIT IntUnit OPTIONAL,
            quantitySoFar           [2] IMPLICIT IntUnit OPTIONAL,
            estimatedCost           [3] IMPLICIT IntUnit OPTIONAL,
            costSoFar               [4] IMPLICIT IntUnit OPTIONAL}
END
```

ASN1.11. Z39.50 ASN.1 Definition for SearchResult-1

```
UserInfoFormat-searchResult-1
{Z39-50-userInfoFormat searchResult-1 (1)} DEFINITIONS ::=
BEGIN
IMPORTS DatabaseName, Term, Query, IntUnit, InternationalString FROM Z39-50-APDU-2001;
SearchInfoReport ::= SEQUENCE OF SEQUENCE{
        subqueryId          [1] IMPLICIT InternationalString OPTIONAL,
                --Shorthand identifier of subquery
        fullQuery           [2] IMPLICIT BOOLEAN,
                --'true' means this is the full query; 'false', a sub-query
subqueryExpression          [3] QueryExpression OPTIONAL,
        --A subquery of the query as submitted.
        --May be whole query; if so, "fullQuery" should be 'true'
subqueryInterpretation      [4] QueryExpression OPTIONAL,
        --How server interpreted subquery
subqueryRecommendation [5] QueryExpression OPTIONAL,
        -- Server-recommended alternative
subqueryCount               [6] IMPLICIT INTEGER OPTIONAL,
        --Number of records for thissubQuery, across
        --all of the specified databases. (If during search,
        --via resource control, number of records so far)
subqueryWeight              [7] IMPLICIT IntUnit OPTIONAL,
        --Relative weight of this subquery
resultsByDB                 [8] IMPLICIT ResultsByDB OPTIONAL}

ResultsByDB ::= SEQUENCE OF SEQUENCE{
        databases           [1] CHOICE{
                            all         [1] IMPLICIT NULL,
                            --Applies across all of the databases in Search APDU
                            list                [2] IMPLICIT SEQUENCE OF DatabaseName
                            --Applies across all databases in this list
                            },
        count               [2] IMPLICIT INTEGER OPTIONAL,
                --Number of records for query component
                --(and, as above, if during search, via resource control,
                -- number of records so far)
        resultSetName               [3] IMPLICIT InternationalString OPTIONAL
        --See comment 1.
        }
QueryExpression ::=  CHOICE {
        term    [1] IMPLICIT SEQUENCE{
                queryTerm       [1]     Term,
                termComment     [2] IMPLICIT InternationalString OPTIONAL},
        query   [2] Query}
END
```

Comment 1

Server-assigned result set by which subQuery is available. Should not be provided unless processing for this query component is concluded (i.e., when this report comes during search, via resource control, as opposed to after search, via additionalSearchInfo)

ASN1.12. Z39.50 ASN.1 Definition for UserInfo-1

```
UserInfoFormat-userInfo-1
{Z39-50-userInfoFormat userInfo-1 (3)} DEFINITIONS ::=
BEGIN
IMPORTS OtherInformation FROM Z39-50-APDU-2001;
UserInfo-1 ::= OtherInformation
END
```

ASN1.13. Z39.50 ASN.1 Definitions for eSpec-2

```
{Z39-50-elementSpec eSpec-2 (2)} DEFINITIONS ::=
--For detailed semantics, see Appendix RET.
BEGIN
IMPORTS Variant FROM RecordSyntax-generic
StringOrNumeric, InternationalString FROM Z39-50-APDU-2001;
Espec-2 ::= SEQUENCE{
elementSetNames          [1] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        -- See comment 1
        defaultVariantSetId      [2] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
                --If supplied, applies whenever variantRequest does not include variantSetId
        defaultVariantRequest    [3] IMPLICIT Variant OPTIONAL,
                --See comment 2.
        defaultTagType           [4] IMPLICIT INTEGER OPTIONAL,
                 --If supplied, applies whenever 'tagType'
                 --(within 'tag' within TagPath) is omitted
        elements                 [5] IMPLICIT SEQUENCE OF ElementRequest OPTIONAL}
ElementRequest::= CHOICE{
        simpleElement            [1] IMPLICIT SimpleElement,
        compositeElement         [2] IMPLICIT SEQUENCE{
                                     elementList      [1] CHOICE{
                                             primitives       [1] IMPLICIT
                                             SEQUENCE OF InternationalString,
                        --Client may specify one or more element set names,
                        --each identifying a set of elements, and the composite element is the union
                                             specs            [2] IMPLICIT
                                             SEQUENCE OF SimpleElement},
                                     deliveryTag      [2] IMPLICIT TagPath,
                                             --DeliveryTag tagPath for compositeElement
                                             --may not include wildThing or wildPath
                                     variantRequest   [3] IMPLICIT Variant OPTIONAL}}
SimpleElement ::= SEQUENCE{
        path                     [1] IMPLICIT TagPath,
        variantRequest           [2] IMPLICIT Variant OPTIONAL}
TagPath ::= SEQUENCE OF CHOICE{
        specificTag              [1] IMPLICIT SEQUENCE{
                                     --The following line, schemaId is the
                                     --only difference in this definition from that of eSpec-1.
                        schemaId         [0] IMPLICIT OBJECT  IDENTIFIER OPTIONAL,
                                     --see comment 3
                        TagType          [1] IMPLICIT INTEGER OPTIONAL,
                        --If omitted, then 'defaultTagType' (above) applies,
                        --if supplied, and if not supplied, then default
                        --listed in schema applies
                        TagValue         [2] StringOrNumeric,
                        occurrence       [3] Occurrences OPTIONAL
```

```
                                --default is "first occurrence"
                },
        wildThing               [2] Occurrences,
        -- See comment 4
        wildPath                [3] IMPLICIT NULL
        -- See comment 5.
}
Occurrences ::= CHOICE{
        all                     [1] IMPLICIT NULL,
        last                    [2] IMPLICIT NULL,
        values                  [3] IMPLICIT SEQUENCE{
                                start      [1] IMPLICIT INTEGER,
                                --If 'start' alone is included, then
                                --single occurrence is requested
                        howMany         [2] IMPLICIT INTEGER OPTIONAL
                        --For example, if 'start' is 5 and 'howMany' is 6,
                        --then request is for "occurrences 5 through 10."
                        }}
END
```

Comment 1
Client may include one or more element set names, each specifying a set of elements. Each of the elements is to be treated as an elementRequest in the form of simpleElement, where occurrence is 1.

Comment 2
If defaultVariantRequest is supplied, then for each simple elementRequest that does not include a variantRequest, the defaultVariantRequest applies. (defaultVariantRequest does not apply to a compositeRequest.)

Comment 3
SchemaId occurs only if the tagType in specificTag is to be interpreted according to some schema other than that which was specified in CompSpec (which is what references eSpec). The optional schema id is attached at the tag level. Its purpose is to qualify the tagType only.

Comment 4
Wildthing: Get Nth "thing" at this level, regardless of tag, for each N specified by "Occurrences" (which may be 'all' meaning match every element at this level). E.g., if "Occurrences" is 3, get third element regardless of its tag or the tag of the first two elements.

Comment 5
Wildpath: Match any tag, at this level or below, that is on a path for which next tag in this TagPath sequence occurs. WildPath may not be last member of the TagPath sequence.

ASN1.14. Z39.50 ASN.1 Definitions for eSpec-q

```
Element Specification eSpec-q
{Z39-50-elementSpec eSpec-q (3)} DEFINITIONS ::=
BEGIN
IMPORTS Term,  AttributeList, AttributeElement
FROM Z39-50-APDU-2001
Espec-q ::= SEQUENCE{
        valueRestrictor         [1] IMPLICIT ValueRestrictor,
        elementSelector         [2] IMPLICIT EXTERNAL OPTIONAL}
```

```
ValueRestrictor ::= SEQUENCE{
        attributeSetId              OBJECT IDENTIFIER,
        nodeSelectionCriteria       RPNStructure}
RPNStructure ::= CHOICE{
        op                                  [0] AttributesPlusTerm,
        rpnRpnOp                            [1] IMPLICIT SEQUENCE{
                        rpn1        RPNStructure,
                        rpn2        RPNStructure,
        op                                  [46] CHOICE{
                        and                 [0]  IMPLICIT NULL,
                        or                  [1]  IMPLICIT NULL,
                        and-not     [2]  IMPLICIT NULL} }}
AttributesPlusTerm ::= [102] IMPLICIT SEQUENCE{
                attributes          AttributeList,
        term    Term}
        END
```

Appendix 19: Maintenance Agency, ZIG, and Historical Background

Non-normative

The Library of Congress is Maintenance Agency/Registration Authority for this standard (this version, Z39.50-2003, and all earlier versions, as well as its ISO equivalent, ISO 23950). The Maintenance Agency web page is http://www.loc.gov/z3950/agency/. The Z39.50 Implementors Group is the advisory body to the Maintenance Agency.

This standard, NISO Z39.50-2003, is a maintenance revision of Z39.50-1995. The remainder of this appendix describes how this version differs from Z39.50-1995, and the reasons for these changes.

Historical Background

Two application protocols, Record Transfer and Information Retrieval, were developed in the early 1980s to support the two intersystem applications of the Linked Systems Project, record exchange and search. In 1983 the LSP participants submitted both protocols to ANSI/NISO for consideration as American National Standards. Record Transfer was withdrawn in favor of FTP, however there remained considerable U.S. interest in standardizing an information retrieval protocol, and the LSP Information Retrieval protocol was assigned to the NISO committee on application protocols who prepared it for ballot, in 1984. It was designated by NISO as "Z39.50", as it is known today. (NISO was formerly named Z39, and continues to use that designation for its standards.)

The 1984 ballot failed within NISO, primarily because it was not yet sufficiently well-developed. There was significant further development over the next three years; Z39.50 was re-balloted in 1987, this time successfully, and it was approved by ANSI and published in 1988. Thus the first published version was Z39.50-1988.

Independently, in 1984, a work item was approved in ISO for a "Search and Retrieve" protocol, SR. There were several drafts of the SR standard between 1984 and 1991 when it was finally approved. The two standards, Z39.50-1988 and SR, shared considerable common functionality, but they were not bit-compatible.

The ZIG and Maintenance Agency

In 1990 the ***Z39.50 Implementors Group*** (ZIG) was established, initially to share Z39.50 implementation resources. Over the past 12 years the ZIG's role has evolved, and during much of that period its primary activity has been to develop and recommend enhancements to the standard.

Also in 1990 a ***Maintenance Agency*** for Z39.50 was established, at the Library of Congress. In late 1991 the Maintenance Agency put forth version 2 for ballot; it was approved in 1992. Version 2, developed by the Maintenance Agency in collaboration with the ZIG, replaced and superseded the 1988 version.

There were two broad categories of change in version 2: changes necessary for alignment with SR, and the addition of features deemed necessary by implementors, to provide sufficient functionality so that implementation would be economically justified. Thus Z39.50-1992 was not identical to but was a compatible superset of SR.

Development of Z39.50-1995 (Version 3)

Many additional enhancements had been proposed by ZIG members for the 1992 version, beyond those that were actually adopted. Several features not yet fully developed or which had not achieved consensus were deferred. Their adoption would have caused significant delay, which was unacceptable because the Maintenance Agency had been assigned as its top priority to revise Z39.50-1988 to achieve bit-compatibility with SR, and to do so quickly, for the credibility of both efforts. The ZIG agreed to defer the proposed new features in return for a commitment from the Maintenance Agency that their development would proceed immediately, and that the resultant subsequent version would be a compatible superset of the 1992 standard.

Thus began in late 1991 an intensive collaboration between the ZIG and Maintenance Agency towards development of (what eventually became) Z39.50-1995. The ZIG met roughly four times per year during that period; for each meeting the Maintenance Agency prepared a new iterative draft (there were a total of ten or so). Each was carefully scrutinized by implementors and discussed at length both over the ZIG listserv and at the ZIG meetings. In April 1994, the ZIG recommended that the draft be finalized. It was balloted successfully in 1995.

The 1992 version came to be known as ***version 2***, and the 1995 version, ***version 3***. Z39.50-1992 replaced and superseded Z39.50-1988, which became obsolete. Z39.50-1995 is a compatible superset of the 1992 version.

Disposition of SR

Z39.50-1992 had been a compatible superset of SR (it included features not in SR such as resource control, access control, and proximity searching). In addition, widening the gap between SR and Z39.50 were the new features developed for Z39.50-1995 (e.g. Sort, Scan, Explain, Segmentation, Concurrent Operations, and Extended Services).

Beginning in 1992, ISO TC 46/SC 4 advanced several proposals intended to narrow or eliminate that gap, by process of alignment by individual amendment. In 1994, however, ISO TC 46/SC 4 decided that that process was excessively burdensome, and instead adopted (by fast-track ballot) the text of Z39.50 verbatim as an ISO standard, ISO 23950-1998.

Collaborative Work of The ZIG and Maintenance Agency since 1995

The Library of Congress maintains a Web page (http://lcweb.loc.gov/z3950/agency) for matters pertaining to the maintenance, ongoing development, and implementation of Z39.50. Available via this page are various categories of information, including the text of Z39.50, definitions of registered objects, and register of implementors. Of particular relevance pertaining to the 2003 revision are: clarifications, commentaries, defect reports, amendments, and implementor agreements. These latter categories of information represent much of the collaborative work of the ZIG and Maintenance Agency since publication of Z39.50-1995, and it is this work primarily that is incorporated into the proposed revision.

Purpose and Scope of Revision

The historical background provided above describes the circumstances and reasons for the revisions in 1992 and 1995. Those circumstances and reasons do not apply to this revision.

There are two primary purposes for this revision. First, ANSI standards are required to undergo periodic review/reaffirmation every five years. Thus reaffirmation of Z39.50 is appropriate in the 2000-2002 timeframe.

The second objective is to integrate the above mentioned information into the standard's text. These clarifications, amendments, etc., have been approved by the ZIG, but have not undergone formal approval beyond that; they have never been approved by a standards body. Many of the ZIG participants feel that for the credibility of the standard, the Maintenance Agency, and the ZIG, this body of work needs formal approval, and that these individual elements need to be integrated into the standard's text.

A reaffirmation is usually very limited in scope; references may be updated, errors and defects corrected, etc. This revision therefore constitutes more than a reaffirmation. On the other hand it does not represent a new version of the standard. Consequently, it is considered a maintenance revision.

Revision in Perspective

This effort to revise Z39.50 has caused some confusion, given the wide discussion of various other aspects of Z39.50's future.

There has been ongoing discussion of a new protocol version, "version 4", not compatible with version 3 (or with Z39.50-1995). Version 4 might incorporate new models and mechanics, include new functionality, and attempt to address intersystem-search requirements for the next decade or so. Discussion has included decoupling the mechanics (syntax and underlying protocol) from the topological, data, and semantics/service models, thus allowing the possibilities of rendering the protocol for instance into XML rather than ASN.1.

In addition, in recent years there has been substantial interest among implementors in evolving Z39.50 to a more mainstream protocol, lowering the barriers to implementation while preserving the existing intellectual capital of Z39.50. This interest has resulted in a new initiative known as ZING (Z39.50-International: Next Generation), described at http://www.loc.gov/z3950/agency/zing/zing.html.

This revision, Z39.50-2003, is not related to either of these two efforts, neither of which would be compatible with Z39.50-1995 (i.e. version 3). Z39.50-2003 is a compatible superset of Z39.50-1995.

The three efforts:

- version 4
- ZING
- 2003 revision

can be seen as complementary, independent initiatives which collectively address the near-to-mid-term future of Z39.50.

Summary of Changes in Z39.50-2003

The following web pages describe much of the changes that have been integrated into the revision:

- **Implementor agreements.** See: http://lcweb.loc.gov/z3950/agency/agree/agree.html
- **Amendments**. See http://lcweb.loc.gov/z3950/agency/amend/amend.html including:
  - Duplicate Detection Service: See http://lcweb.loc.gov/z3950/agency/amend/am2.html, and
  - Encapsulation. See: http://lcweb.loc.gov/z3950/agency/amend/am3.html
- **Defect reports**. See http://lcweb.loc.gov/z3950/agency/defects/defects.html
- **Commentaries**. See http://lcweb.loc.gov/z3950/agency/wisdom/wisdom.html
- **Clarifications**. See http://lcweb.loc.gov/z3950/agency/clarify/clarify.html

- **Version 3 Baseline Requirements**. See http://lcweb.loc.gov/z3950/agency/v3base.html
- **Z39.50 Attribute Architecture**. See http://lcweb.loc.gov/z3950/agency/attrarch/attrarch.html
- **Negotiation Model** (Model for Z39.50 Negotiation During Initialization). See http://lcweb.loc.gov/z3950/agency/nego.html

Miscellaneous changes

OSI.

All references to OSI (Open System Interconnection) are purged. Purging began in the 1995 version, and is now complete. Thus such concepts as Application Contexts and Presentation Contexts are deleted; 'origin' and 'target' are replaced by 'client' and 'server'.

ASN.1

All of the ASN.1 has been consolidated into a single appendix. This includes APDU as well as object definitions.

State Tables

State tables have been removed, along with accompanying prose and supporting definitions.

Appendices

The following table summarizes changes in the appendices.

Appendix in Z39.50-1995	Corresponding Appendix in Z39.50-2003	Changes
Appendix 1: OID (object Identifiers)	Appendix 1	No additional substantial changes. See note 1.
Appendix 2: CTX (application contexts)	none	removed entirely
Appendix 3:ATR (attributes)	Appendix 2	Bib-1 removed. See note 2.
Appendix 4: ERR (diagnostics)	Appendix 3: DIAG (name change)	bib-1 renamed 'general diagnostic set'. Diag-1 removed. 'General Diagnostic Container' added. 'Returning Diagnostics in an Init Response' added.
Appendix 5: REC (record syntaxes)	Appendix 4.	The list of syntaxes and their oids is removed. OPAC and Summary removed.

Appendix 6: RSC (resource reports	Appendix 5	Resource-1 removed. See note 3.
Appendix 7:ACC (access control)	Appendix 6	No additional substantial changes. See note 1.
Appendix 8: EXT (extended services)	Appendix 7	No additional substantial changes. See note 1.
Appendix 9: USR (user information formats)	Appendix 8	• Old USR.2 (negotiation records) has been removed, superceded by new appendix 14. • New section 'use of Init Parameters for User Information' • New section 'General User Information format, User-1".
Appendix 10: ESP (especs)	Appendix 9	• eSpec-2 replaces eSpec-1. • eSpec-q added.
Appendix 11:VAR (variant sets)	Appendix 10	No additional substantial changes. See note 1.
Appendix 12:TAG (tagSets & schemas)	Appendix 11	No additional substantial changes. See note 1.
Appendix 13:ERS (result set model)	Appendix 12	Unchanged
Appendix 14:RET (retrieval)	Appendix 13	No additional substantial changes. See note 1.
Appendix 15:PRO (profiles)	Appendix 16	completely re-written
Appendix 16 (maintenance agency)	None.	
	Appendix 14: NEGO (Negotiation model)	New in Z39.50-2003
	Appendix 15: NEGO2 (negotiation records)	New in Z39.50-2003
	Appendix 17 (attribute architecture)	New in Z39.50-2003

Notes:
1. 'No substantial changes' above mean none other than (perhaps) changes made in accordance with changes listed above before the table (which themselves may be substantial).
2. Bib-1 attributes are no longer maintained within the text of Z39.50. They will continue to be maintained on the web site.
3. Originally there was resource-1, which listed 16 categories of resources, with no provision of extensibility. Resource-2 was added in 1995, inheriting the original 16 categories, with provision for extensibility. Thus Resource-2 is a compatible superset of resource-1. Thus resource-1 is dropped resource-2 retained (with a comment to the effect that a client might want to be prepared to recognize the resource-1 object identifier.)