ANSI/NISO

Information Retrieval (Z39.50):

Application Service Definition and

Protocol Specification

Appendicies

Table of Contents

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 Z39.50-2001.

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

ANSI/NISO

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 Z39.50-2001, 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.

ANSI/NISO

OID.7 Experimental Objects

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

ANSI/NISO

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/bib1diag.html

Code	Meaning	Addinfo
1	permanent system error	(unspecified)
2	temporary system error	(unspecified)
3	unsupported search	(unspecified)
4	Terms only exclusion (stop) words	(unspecified)
5	Too many argument words	(unspecified)

ANSI/NISO

Code	Meaning	Addinfo
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)
104	Challenge failed - record not included	(unspecified)
105	Terminated at client request	(unspecified)
106	No abstract syntaxes agreed to for this record	(unspecified)

ANSI/NISO

Code	Meaning	Addinfo
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
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

ANSI/NISO

Code	Meaning	Addinfo
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)
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)

ANSI/NISO

Code	Meaning	Addinfo
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 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	

ANSI/NISO

Code	Meaning	Addinfo
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 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	

ANSI/NISO

Code	Meaning	Addinfo
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 proposed (on behalf of the CIMI project) 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.

ANSI/NISO

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

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.5 Generic Record Syntax 1

See ASN1.6

REC5.1 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).

ANSI/NISO

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.6 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

Common Task Package Parameter	Client supplied	Server supplied	Reference
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.

ANSI/NISO

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

ANSI/NISO

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		

Specific Task Parameter	Client Supplied	Server Supplied	Task Package Parameter
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.6.

'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.

ANSI/NISO

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 is as thought 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
6	is variant supported?	NULL
7	document descriptor	NULL
8	surrogate information	NULL
998	all meta-data	NULL

Type	Meaning	Datatype/value
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 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 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.6)

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.5, 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.

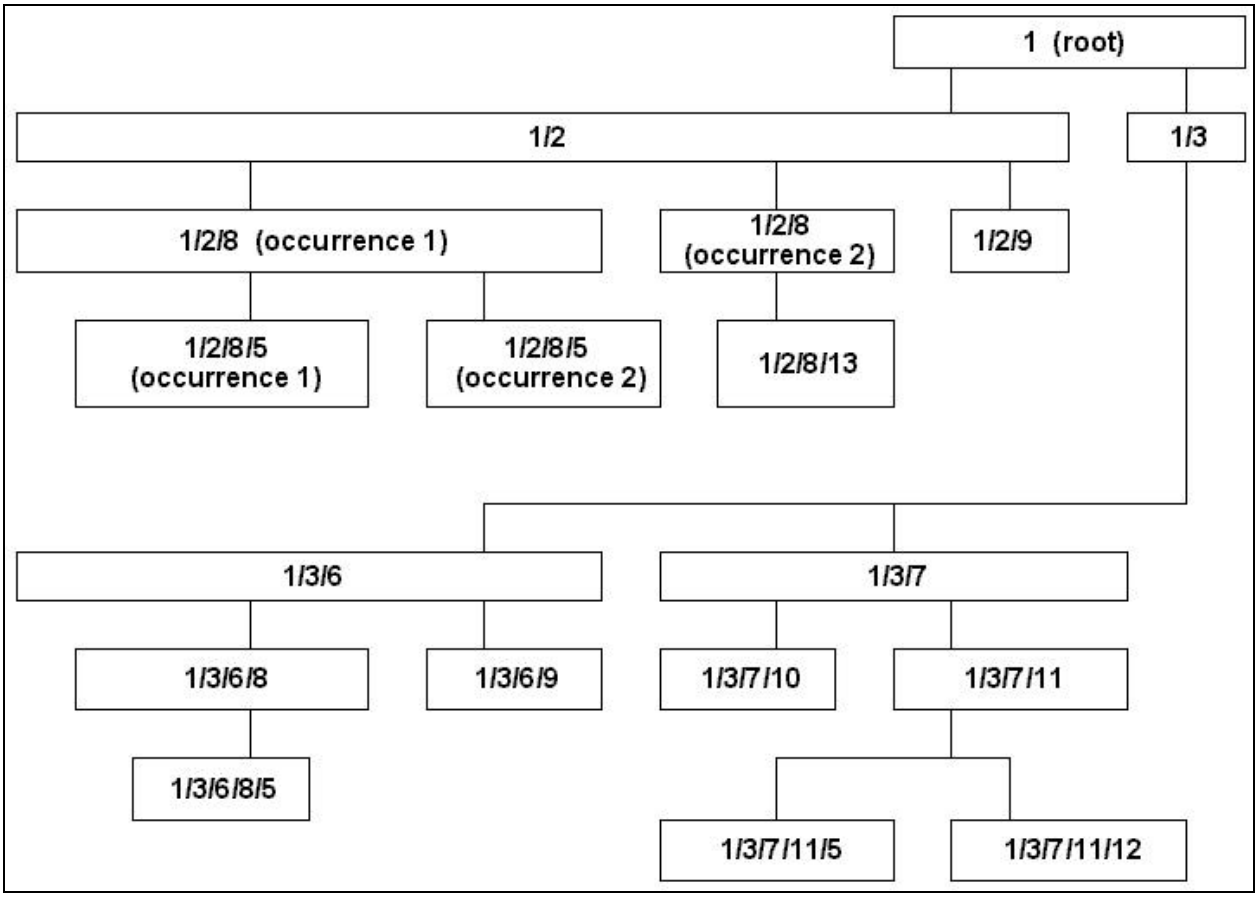
RET.3.1.1.4 Wild-cards

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.isi.edu/in-notes/iana/assignments/media-types/media-types)

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)

NEGO 2.1 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 Z39.50-2001 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 Z39.50-2001 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 it contains, profiles provide a necessary level of specification to achieve interoperability.

Profile developers are encouraged to take advantage of the39.50 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 t ype 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)
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 atttribute set will be developed and 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.