Appendix 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}
PeriodicQuerySchedule	{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 which are specific to the task type. Among the common parameters, some are supplied by the origin as parameters in the ES request, and others are supplied by the target. The specific parameters are derived from the ES request parameter Task-specific-parameters. Table EXT-1 provides a summary of common parameters. Their descriptions are included in 3.2.9.1. For parameters listed as both "origin supplied" and "target supplied", when both origin and target supply a value, the target supplied value overrides the origin supplied value.

Common Task Package Parameter	Origin Supplied	Target Supplied
packageType	x	
packageName	x (opt)	
userId	x (opt)	
retentionTime	x (opt)	x (opt)
permissionsList	x (opt)	x (opt)
description	x (opt)	
targetReference		x (opt)
creationDateTime		x (opt)
taskStatus		x
packageDiagnostics		x (opt)

Table EXT-1: parameters common to all Extended Service.

EXT.1.1 Persistent Result Set Extended Service

The Persistent Result Set Extended Service allows an origin to request that the target 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, either during the current or a different Z-association. It may subsequently be deleted, by deletion of the task package.

> Note: the origin may thus cause deletion of the persistent result set, by deleting the task package, if the origin 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 which contains a target-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).

The parameters of the Persistent Result Set Extended Service are those shown in table EXT-1 as well as those in table EXT-2.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
originSuppliedResultSet	x (if appl)		
replaceOrAppend	x (if appl)		
targetSuppliedResultSet		x (if appl)	x (if appl)
numberOfRecords		x (opt)	x (opt)

Table EXT-2: Specific Parameters for Persistent Result Set

originSuppliedResultSet The origin supplies the name of a transient result set belonging to the Z-association. If function is 'create', the target is to create a persistent result set from this transient result set. If function is 'modify' the target 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 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 origin 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.

targetSuppliedResultSet When the origin retrieves the task package, the target supplies the name of a transient result set, which then belong to the Z-association. The result set is a copy of the persistent result set represented by the package. (The target supplies this parameter only when the task package is retrieved, not on an ES response.)

numberOfRecords The target indicates the total number of records contained in the persistent result set.

EXT.1.2 Persistent Query Extended Service

The Persistent Query Extended Service allows an origin to request that the target save a Z39.50 Query for later reference, during the same or a subsequent Z-association.

The parameters of the Persistent Query Extended Service are those shown in table EXT-1 as well as those in table EXT-3.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
querySpec	x		
actualQuery		x	x
databaseNames	x (opt)		x (opt)
additionalSearchInformation	x (opt)		x (opt)

Table EXT-3: Specific Parameters for Persistent Query

querySpec and ActualQuery The origin supplies either the query to be saved or the name of another persistent query to be copied into this package. The target supplies the actualQuery: if the origin supplied a query, the target uses that query; if the origin supplies a task package name, the target copies the corresponding query.

databaseNames The origin 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 an origin to request that the target establish a Periodic Query Schedule. The origin 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 EXT-1 as well as those in table EXT-4.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
activeFlag	x		x
querySpec	x		
actualQuery		x	x
databaseNames	x (if appl)		x (if appl)
period	x	x (opt)	x
expiration	x (opt)	x (opt))	x (opt)
resultSetPackageName	x (opt)	x (if appl)	x (if appl)
resultSetDisposition	x (if appl)		x (if appl)
alertDestination	x (opt)		x (opt)
exportParameters	x (opt)		x (opt)
lastQueryTime		x	x
lastResultNumber		x	x
numberSinceModify		x (opt)	x (opt)

Table EXT-4: Specific Parameters for Periodic Query Schedule

activeFlag On a Create request, if this flag 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 origin may activate or deactivate the schedule. In the parameter package, this parameter indicates whether the schedule is active.

querySpec and ActualQuery The origin supplies either a query or the name of a Persistent Query Package. (If the origin supplies a query, or if the specified query package does not include a list of databases, then the databaseNames parameter is required.) The target supplies the actualQuery: if the origin supplied a query, the target uses that query; if the origin supplies a task package name, the target copies the corresponding query.

databaseNames The origin may supply a list of databases; the list is required if the origin supplied a query rather than a query package name for querySpec, or if the specified query package does not include a list of databases.

period The time period between invocations of the query. The target may override the period specified by the origin. 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 target's discretion).

expiration The origin may optionally supply a time/date for the target to discontinue execution of this Periodic Query. If the origin does not supply a value, the origin is proposing "no expiration". The target may override the origin supplied value. If the origin supplies a value and the target does not support expiration, the target should reject the ES request.

resultSetPackageName The origin may optionally supply the name of an existing Persistent Result Set package. If the origin omits this parameter, the target 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 target 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 origin and target 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 target to continually extend the result set by appending new records.

alertDestination The origin may optionally supply a destination address for Alerts triggered by receipt of new Periodic Query results (e.g, fax number, X.400 address, pager number).

exportParameters The origin may optionally sypply the name, or actual contents, of an Export Parameter Package to be used with this Periodic Query. It is included only if the origin wants newly posted results to be exported; if so new results may also be posted to ResultSetName if also specified.

lastQueryTime The target indicates the last time this Periodic Query was invoked.

lastResultNumber The target indicates the number of new records obtained last time query was invoked.

numberSinceModify The target 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 an origin to submit an item order request to the target. The parameters of the Item Order Extended Service are those shown in table EXT-1 as well as those in table EXT-5.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
requestedItem	x		
itemRequest		x (if appl)	x (if appl)
supplementalDescription	x (opt)		x (opt)
contactInformation	x (opt)		x (opt)
additionalBillingInfo	x (opt)		x (opt)
statusOrErrorReport		x	x
auxiliaryStatus		x (opt)	x (opt)

Table EXT-5: Task-Specific Parameters for Item Order

requestedItem The origin identifies the requested item, either by:

- 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
- an item request, whose definition is external to this standard, and which may be an Interlibrary Loan Request PDU of ISO 10161; or
- both.

itemRequest If requestedItem is an item request (e.g. an interlibrary loan request), the target copies it into the task package (although the target might first modify the request). If requestedItem is a result set item, the target may supply a corresponding item request; if it does not, then the requested item will not be identified within the task package.

supplementalDescription The origin may supply additional descriptive information pertaining to the requested item, as a supplement to requestedItem.

contactInformation The origin may optionally supply a name, phone number, and electronic mail address of a contact-person.

additionalBillingInfo The origin may optionally indicate payment method, credit card information, customer reference, and customer purchase order number.

statusOrErrorReport The target 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 target may provide an auxiliary status as a supplement to the status information which might be provided by the statusOrErrorReport.

EXT 1.5 DatabaseUpdate Extended Service

The databaseUpdate Extended Service allows an origin to request that the target 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 target 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 EXT-1 as well as those in table EXT-6.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
action	x		x
databaseName	x		x
schema	x (opt)		x (opt)
suppliedRecords	x		
recordIds	x (opt)		
supplementalIds	x (opt)		
correlationInfo	x (opt)		x (opt)
elementSetName	x (opt)		x (opt)
updateStatus		x (if appl)	x (if appl)
globalDiagnostics		x (if appl)	x (if appl)
taskPackageRecords		x (if appl)	x (if appl)
recordStatuses		x (if appl)	x (if appl)

Table EXT-6: Task-Specific Parameters for databaseUpdate

action The origin indicates recordInsert, recordReplace, recordDelete, or elementUpdate.

databaseName The origin indicates the database to which the action pertains.

schema The origin indicates the database schema which applies for this update.

Note: The action, databaseName, and schema are specified once, and apply to all of the included records.

suppliedRecords The origin supplies one or more records. (Along with each the origin may also supply a recordId, supplemental identification, and correlation information; see following three parameters.) For recordInsert or recordReplace, the origin supplies whole records. For recordReplace or recordDelete, each supplied record (or corresponding supplemental identification or recordId) must include sufficient information for the target 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, 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 origin may optionally supply a record Id.

supplementalIds Corresponding to each supplied record the origin may supply supplemental identification to allow the target 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 origin may include one or both of the following:

- a correlationNote,
- a correlationIdentifier.

The correlationIdentifier may be used to identify the record only within the context of this update task, for correlation purposes only (i.e to correlate a task package record with its corresponding supplied record). It may be used in the task package in lieu of a record id, for a record that might not have an unambiguous record id.

ElementSetName The origin 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:

Success	-	Update performed successfully.
Partial	-	Update failed for one or more records.
Failure	-	Target rejected execution of the task.

globalDiagnostics The target should supply one or more non-surrogate diagnostics 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, 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 record for which action is not complete, the structure includes the correplationInfo and status.

recordStatuses Corresponding to each task package record, the task package includes a record status:

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 target 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.

EXT 1.6 Export Specification Extended Service

The Export Specification Extended Service allows an origin to request that the target establish an export specification. Once established, the export specification may be subsequently invoked (repeatedly) by an Export Invocation Extended Services task; there may in fact be multiple invocations running simultaneously.

An Export Specification includes a delivery destination as well as other information which 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, X.400, FTAM, UUCP, FTP, or a target-supported print device. The parameters of the Export Specification Extended Service are those shown in table EXT-1 as well as those in table EXT-7.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
composition	x		x
exportDestination	x		x

Table EXT-7: Task-Specific Parameters for Export Specification

composition This parameter consists of a record syntax, element specification, variants, etc. of the records to be Exported.

exportDestination The origin 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 an origin to invoke an export specification. The origin may supply a export specification, or the name of an export specificiation 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 EXT-1 as well as those in table EXT-8.

Specific Task Parameter	Origin Supplied	Target Supplied	Task Package Parameter
exportSpecification	x		x
resultSetId	x		
resultSetRecords	x		
numberOfCopies	x		x
estimatedQuantity		x (opt)	x (opt)
quantitySoFar		x (opt)	x (opt)
estimatedCost		x (opt)	x (opt)
costSoFar		x (opt)	x (opt)

Table EXT-8: Task-Specific Parameters for Export Invocation

exportSpecification The origin supplies the packageName, or actual contents, of an export specification.

resultSetId The origin supplies the name of a transient result set, from which records are selected for export.

resultSetRecords The origin 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 origin indicates the number of copies requested.

estimatedQuantity and quantitySoFar The target 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 target 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 origin in the ES request; these are divided into those which are and which are not to be retained in the task package; 'toKeep' and 'notToKeep'. 'taskPackage' consists of all specific task parameters; these are divided into those supplied by the origin and those supplied by the target, 'originPart' and 'targetPart'. Note that 'toKeep' (from 'esRequest') is always the same sub-structure as 'originPart' (from taskPackage), so that structure is shared, in OriginPartToKeep.

Each definition may define one or more of OriginPartToKeep, OriginPartNotToKeep, and TargetPart. In EXT.1, in the parameter table in the service definition for a specific ES, for each parameter:

- If the parameter is marked "origin 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 OriginPartNotToKeep.
- If the parameter is marked "origin supplied", and also marked in the right column, then that parameter is represented in OriginPartToKeep.
- If the parameter is marked "target supplied" (in which case it will always also be marked in the right column), and not also marked "origin supplied" then that parameter is represented in TargetPart.
- If the parameter is marked "origin supplied", and also marked "target supplied" (in which case it will be marked in the right column), then it is a parameter for which the origin may suggest a value and the target may overide that value. In this case the origin suggested value is represented in OriginPartNotToKeep and the target value (which may be the same) is represented in "targetPart".

ESFormat-PersistentResultSet

```
{Z39-50-extendedService PersistentResultSet (1)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString FROM Z39-50-APDU;
PersistentResultSet ::= CHOICE{
    esRequest           [1] IMPLICIT SEQUENCE{
                                toKeep          [1] IMPLICIT NULL,
                                notToKeep       [2] OriginPartNotToKeep OPTIONAL},
    taskPackage         [2] IMPLICIT SEQUENCE{
                                originPart      [1] IMPLICIT NULL,
                                targetPart      [2] TargetPart OPTIONAL}}
OriginPartNotToKeep ::= SEQUENCE{
        originSuppliedResultSet     [1] IMPLICIT InternationalString OPTIONAL,
                                        -- name of transient result set, supplied on request.
                                        -- Mandatory unless function is 'delete'.
        replaceOrAppend             [2] IMPLICIT INTEGER{  -- only if function is "modify"
                                    replace         (1),
                                    append          (2)} OPTIONAL}
```

```
TargetPart ::= SEQUENCE{
        targetSuppliedResultSet          [1] IMPLICIT InternationalString OPTIONAL,
                                                 -- name of transient result set, supplied by target,
                                                 -- representing the persistent result set to which package
                                                 -- pertains. Meaningful only when package is presented
                                                 -- (i.e. not on ES response)
        numberOfRecords                  [2] IMPLICIT INTEGER OPTIONAL}
END
```

ESFormat-PersistentQuery

```
{Z39-50-extendedService PersistentQuery (2)} DEFINITIONS ::=
BEGIN
IMPORTS Query, InternationalString, OtherInformation FROM Z39-50-APDU;
PersistentQuery ::= CHOICE{
    esRequest             [1] IMPLICIT SEQUENCE{
                                   toKeep          [1] OriginPartToKeep OPTIONAL,
                                   notToKeep       [2] OriginPartNotToKeep},
    taskPackage           [2] IMPLICIT SEQUENCE{
                                   originPart      [1] OriginPartToKeep OPTIONAL,
                                   targetPart      [2] TargetPart}}
OriginPartToKeep ::= SEQUENCE{
        dbNames                    [2] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
        additionalSearchInfo       [3] OtherInformation OPTIONAL}
OriginPartNotToKeep ::= CHOICE{
        package            [1] IMPLICIT InternationalString,
        query              [2] Query}
TargetPart ::= Query
END
```

ESFormat-PeriodicQuerySchedule

```
{Z39-50-extendedService PeriodicQuerySchedule (3)} DEFINITIONS ::=
BEGIN
IMPORTS Query, InternationalString, IntUnit FROM Z39-50-APDU;
ExportSpecification FROM ESFormat-ExportSpecification;
EXPORTS Destination;

PeriodicQuerySchedule ::= ::= CHOICE{
    esRequest          [1] IMPLICIT SEQUENCE{
                                 toKeep       [1] OriginPartToKeep,
                                 notToKeep  [2] OriginPartNotToKeep},
    taskPackage       [2] IMPLICIT SEQUENCE{
                                 originPart   [1] OriginPartToKeep,
                                 targetPart   [2] TargetPart}}

OriginPartToKeep ::=SEQUENCE{
     activeFlag                    [1] IMPLICIT BOOLEAN,
     databaseNames                 [2] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
```

```
    resultSetDisposition        [3] IMPLICIT INTEGER{
                                        replace     (1),
                                        append      (2),
                                        createNew   (3)  -- Only if origin and target have agreement about
                                                         -- naming convention for the resulting package,
                                                         -- and only if no resut set is specified.
                                } OPTIONAL,         -- Mandatory on 'create' if result set is specified, in
                                                    -- which case it must be 'replace' or 'append.
    alertDestination            [4] Destination OPTIONAL,
    exportParameters            [5] CHOICE{
                                        packageName   [1]  IMPLICIT InternationalString,
                                        exportPackage  [2] ExportSpecification} OPTIONAL}

OriginPartNotToKeep ::= SEQUENCE{
    querySpec                   [1] CHOICE{
                                        actualQuery     [1] Query,
                                        packageName   [2] IMPLICIT InternationalString} OPTIONAL,
                                    -- mandatory for 'create'
    originSuggestedPeriod       [2] Period OPTIONAL, -- mandatory for 'create'
    expiration                  [3] IMPLICIT GeneralizedTime OPTIONAL,
    resultSetPackage            [4] IMPLICIT InternationalString OPTIONAL}

TargetPart ::= SEQUENCE{
    actualQuery             [1] Query,
    targetStatedPeriod      [2] Period,
                                    -- target supplies the period, which may be same as origin proposed.
    expiration              [3] IMPLICIT GeneralizedTime OPTIONAL,
                                    -- Target supplies value for task package. It may be the same as origin
                                    -- proposed or different from (and overides) origin proposal, but if
                                    -- omitted, there is no expiration.
    resultSetPackage        [4] IMPLICIT InternationalString OPTIONAL,
                                    -- May be omitted only if exportParameters was supplied. Target
                                    -- supplies same name as origin supplied, if origin did supply a name.
    lastQueryTime           [5] IMPLICIT GeneralizedTime,
    lastResultNumber        [6] IMPLICIT INTEGER,
    numberSinceModify       [7] IMPLICIT INTEGER OPTIONAL}

    Period ::= CHOICE{
                    unit                [1] IMPLICIT IntUnit,
                    businessDaily       [2] IMPLICIT NULL,
                    continuous          [3] IMPLICIT NULL,
                    other               [4] IMPLICIT InternationalString}
  Destination ::= CHOICE{
    phoneNumber       [1]     IMPLICIT InternationalString,
    faxNumber         [2]     IMPLICIT InternationalString,
    x400address       [3]     IMPLICIT InternationalString,
    emaIlAddress      [4]     IMPLICIT InternationalString,
    pagerNumber       [5]     IMPLICIT InternationalString,
    ftpAddress        [6]     IMPLICIT InternationalString,
```

```
     ftamAddress          [7]      IMPLICIT InternationalString,
     printerAddress       [8]      IMPLICIT InternationalString,
     other                [100]    IMPLICIT SEQUENCE{
                                         vehicle          [1]  IMPLICIT InternationalString  OPTIONAL,
                                         destination      [2] IMPLICIT InternationalString}}
END
```

ESFormat-ItemOrder

```
{Z39-50-extendedService ItemOrder (4)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString FROM Z39-50-APDU;
ItemOrder ::= ::= CHOICE{
    esRequest         [1] IMPLICIT SEQUENCE{
                                 toKeep       [1] OriginPartToKeep OPTIONAL,
                                 notToKeep  [2] OriginPartNotToKeep}
    taskPackage       [2] IMPLICIT SEQUENCE{
                                 originPart   [1] OriginPartToKeep OPTIONAL,
                                 targetPart   [2] TargetPart}}
OriginPartToKeep ::= SEQUENCE{
     supplDescription          [1] IMPLICIT EXTERNAL OPTIONAL,
     contact                   [1] IMPLICIT SEQUENCE{
                                         name    [1]   IMPLICIT InternationalString OPTIONAL,
                                         phone   [2]   IMPLICIT InternationalString OPTIONAL,
                                         email   [3]  IMPLICIT InternationalString OPTIONAL} OPTIONAL,
     addlBilling               [3] IMPLICIT SEQUENCE{
                                    paymentMethod           [1]   CHOICE{
                                         billInvoice                      [0] IMPLICIT NULL,
                                         prepay                           [1] IMPLICIT NULL,
                                         depositAccount                   [2] IMPLICIT NULL,
                                         creditCard                       [3] IMPLICIT CreditCardInfo,
                                         cardInfoPreviouslySupplied       [4] IMPLICIT NULL,
                                         privateKnown                     [5] IMPLICIT NULL,
                                         privateNotKnown                  [6] IMPLICIT EXTERNAL},
                                    customerReference       [2] IMPLICIT InternationalString OPTIONAL,
                                    customerPONumber        [3] IMPLICIT InternationalString OPTIONAL}
                                                                      OPTIONAL}
CreditCardInfo   ::= SEQUENCE{
          nameOnCard          [1] IMPLICIT InternationalString,
          expirationDate      [2] IMPLICIT InternationalString,
          cardNumber          [3] IMPLICIT InternationalString}

OriginPartNotToKeep ::= SEQUENCE{
                    Must supply at least one, and may supply both.
     resultSetItem          [1]   IMPLICIT SEQUENCE{
                                    resultSetId       [1]   IMPLICIT InternationalString,
                                    item              [2]   IMPLICIT INTEGER} OPTIONAL,
     itemRequest            [2] IMPLICIT EXTERNAL OPTIONAL}
```

```
TargetPart ::= SEQUENCE{
     itemRequest              [1]  IMPLICIT EXTERNAL OPTIONAL,
     statusOrErrorReport      [2] IMPLICIT EXTERNAL,
     auxiliaryStatus          [3]  IMPLICIT INTEGER{
                                        notReceived          (1),
                                        loanQueue            (2),
                                        forwarded            (3),
                                        unfilledCopyright    (4),
                                        filledCopyright      (5)} OPTIONAL}
END
```

ESFormat-Update

```
{Z39-50-extendedService Update (5)} DEFINITIONS ::=
BEGIN
IMPORTS DiagRec, InternationalString FROM Z39-50-APDU;
Update ::= CHOICE{
    esRequest          [1] IMPLICIT SEQUENCE{
                                toKeep       [1] OriginPartToKeep,
                                notToKeep [2] OriginPartNotToKeep},
    taskPackage        [2] IMPLICIT SEQUENCE{
                                originPart   [1] OriginPartToKeep,
                                targetPart   [2] TargetPart}}
OriginPartToKeep ::= SEQUENCE{
     action                        [1] IMPLICIT INTEGER{
                                        recordInsert          (1),
                                        recordReplace         (2),
                                        recordDelete          (3),
                                        elementUpdate         (4)},
     databaseName                  [2] IMPLICIT InternationalString,
     schema                        [3] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
     elementSetName                [4] IMPLICIT InternationalString OPTIONAL}
OriginPartNotToKeep ::= SuppliedRecords

TargetPart ::= SEQUENCE{
     updateStatus             [1] IMPLICIT INTEGER{
                                        success      (1),
                                        partial      (2),
                                        failure      (3)},
     globalDiagnostics        [2] IMPLICIT SEQUENCE OF DiagRec OPTIONAL,
                                        -- These are non-surrogate diagnostics relating to the task,
                                        -- not to individual records.
     taskPackageRecords       [3] IMPLICIT SEQUENCE OF TaskPackageRecordStructure
                                        -- There should be a TaskPackageRecordStructure for every record
                                        -- supplied. The target should create such a structure (for every
                                        -- record) immediately upon creating the task package, to include
                                        -- correlation information and status. The record itself would not
                                        -- be included until processing for that record is complete.
                         }
```

```
-- Auxiliary definitions for Update
SuppliedRecords ::= SEQUENCE OF SEQUENCE{
    recordId            [1] CHOICE{
                                number      [1] IMPLICIT INTEGER,
                                string      [2] IMPLICIT InternationalString,
                                opaque      [3] IMPLICIT OCTET STRING} OPTIONAL,
    supplementalId      [2] CHOICE{
                                timeStamp           [1] IMPLICIT GeneralizedTime,
                                versionNumber       [2] IMPLICIT InternationalString,
                                previousVersion     [3] IMPLICIT EXTERNAL} OPTIONAL,
    correlationInfo     [3] IMPLICIT CorrelationInfo OPTIONAL,
    record              [4] IMPLICIT EXTERNAL}

CorrelationInfo ::= SEQUENCE{
                    -- origin may supply one or both for any record:
    note    [1] IMPLICIT InternationalString OPTIONAL,
    id      [2] IMPLICIT Integer OPTIONAL}

TaskPackageRecordStructure ::= SEQUENCE{
        recordOrSurDiag     [1] CHOICE {
                                record      [1] IMPLICIT EXTERNAL,
                                                -- Choose 'record' if recordStatus is 'success', and
                                                -- elementSetName was supplied.
                                diagnostic  [2] DiagRec
                                                -- Choose 'diagnostic', if RecordStatus is failure.
                                                            } OPTIONAL,
                                    -- The parameter recordOrSurDiag will thus be omitted only if
                                    -- 'elementSetName' was omitted and recordStatus is
                                    -- 'success'; or if record status is 'queued' or in 'process'.
        correlationInfo     [2] IMPLICIT CorrelationInfo OPTIONAL,
                                -- This should be included if it was supplied by the origin.
        recordStatus        [3] IMPLICIT INTEGER{
                                success     (1),
                                queued      (2),
                                inProcess   (3),
                                failure     (4)}}
END
```

ESFormat-ExportSpecification

```
{Z39-50-extendedService ExportSpecification (6)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString, CompSpec FROM Z39-50-APDU;
Destination FROM ESFormat-PeriodicQueryService; EXPORTS ExportSpecification;
ExportSpecification ::= CHOICE{
    esRequest       [1] IMPLICIT SEQUENCE{
                toKeep      [1] OriginPartToKeep,
                notToKeep   [2] IMPLICIT NULL,
```

```
    taskPackage       [2] IMPLICIT SEQUENCE{
                  originPart   [1] OriginPartToKeep,
                  targetPart   [2] IMPLICIT NULL}}

OriginPartToKeep ::= SEQUENCE{
     composition              [1] IMPLICIT CompSpec,
     exportDestination        [2] Destination}
END
```

ESFormat-ExportInvocation

```
{Z39-50-extendedService ExportInvocation (7)} DEFINITIONS ::=
BEGIN
IMPORTS InternationalString FROM Z39-50-APDU;
ExportSpecification FROM ESFormat-ExportSpecification;
ExportInvocation ::= CHOICE{
    esRequest         [1] IMPLICIT SEQUENCE{
                  toKeep       [1] OriginPartToKeep,
                  notToKeep  [2] OriginPartNotToKeep},
    taskPackage       [2] IMPLICIT SEQUENCE{
                  originPart         [1] OriginPartToKeep,
                  targetPart         [2] TargetPart OPTIONAL}}

OriginPartToKeep ::= SEQUENCE{
     exportSpec            [1] CHOICE{
                                    packageName   [1]   IMPLICIT InternationalString,
                                    packageSpec     [2] ExportSpecification},
     numberOfCopies    [2] IMPLICIT INTEGER}

OriginPartNotToKeep   ::= SEQUENCE{
     resultSetId            [1] IMPLICIT InternationalString,
     records                 [2] CHOICE{
                                    all        [1] IMPLICIT NULL,
                                    ranges  [2]   IMPLICIT SEQUENCE OF SEQUENCE{
                                                  start      [1] IMPLICIT INTEGER,
                                                  count    [2] IMPLICIT INTEGER OPTIONAL
                                                       -- Count may be omitted only on last range, to indicate
                                                       -- "all remaining records beginning with 'start'".
                                           }}

TargetPart   ::= SEQUENCE{
          estimatedQuantity        [1] IMPLICIT IntUnit OPTIONAL,
          quantitySoFar             [2] IMPLICIT IntUnit OPTIONAL,
          estimatedCost              [3] IMPLICIT Intunit OPTIONAL,
          costSoFar                   [4] IMPLICIT IntUnit OPTIONAL}
END
```

Appendix 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.

This standard defines and registers the userInformation format SearchResult-1, defined for use within a SearchResponse APDU. The following object identifier is assigned:

SearchResult-1 {Z39-50-userInfoFormat 1} (See USR.1)

UserInformation formats may include negotiation records, defined for the parameters userInformationField and otherInfo in the Init and InitResponse APDUs. These are described in USR.2.

USR.1 User Information Format SearchResult-1

SearchResult-1 is for use primarily within the AdditionalSearchInformation parameter in the Search Response. The format allows the target to provide information per query component (the whole query or a sub-query, possibly restricted to a subset of the specified databases). The target 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 target to report on the progress of the search. However, when used in this manner, the target should not create a result set for a query component unless processing for that component is complete.

UserInfoFormat-searchResult-1

```
{Z39-50-userInfoFormat searchResult-1 (1)} DEFINITIONS ::=
BEGIN
IMPORTS DatabaseName, Term, Query, IntUnit, InternationalString FROM Z39-50-APDU;
SearchInfoReport ::= SEQUENCE OF SEQUENCE{
    subqueryId                  [1]  IMPLICIT InternationalString OPTIONAL,
                                          -- Shorthand identifier of subquery.
    fullQuery                   [2]  IMPLICIT BOOLEAN,
                                          -- 'true' means this is the full query; 'false', a sub-query.
    subqueryExpression          [3]  QueryExpression OPTIONAL,
                                          -- A subquery of the query as submitted. May be whole
                                          -- query; if so, "fullQuery" should be 'true'.
    subqueryInterpretation      [4]  QueryExpression OPTIONAL,
                                          -- How target interpreted subquery.
    subqueryRecommendation      [5]  QueryExpression OPTIONAL,
                                          -- Target-recommended alternative.
    SubqueryCount               [6]  IMPLICIT INTEGER OPTIONAL,
                                          -- Number of records for this subQuery, across all of the
                                          -- specified databases. (If during search, via resource control,
                                          -- number of records so far).
    subqueryWeight              [7] IMPLICIT Intunit OPTIONAL,
                                          -- Relative weight of this subquery.
    resultsByDB                 [8]  IMPLICIT ResultsByDB OPTIONAL}
```

```
ResultsByDB ::= SEQUENCE OF SEQUENCE{
      databases            [1] CHOICE{
                                 all        [1] IMPLICIT NULL,
                                                    -- applies across all of the databases in Search PDU
                                 list       [2]   IMPLICIT SEQUENCE OF DatabaseName
                                                    -- applies across all databases in this list
                                       }
      count                [2] IMPLICIT INTEGER OPTIONAL,
                                 -- Number of records for query component (and, as above, if during search,
                                 -- via resource control, number of records so far).
      resultSetName        [3] IMPLICIT InternationalString OPTIONAL
                                 -- Target-assigned result set by which subQuery is available. Should not
                                 -- be provided unless processing for this query component is concluded (i.e.
                                 -- when this report comes during search, via resource control, as opposed
                                 -- to after search, via additionalSearchInfo).
                                       }

QueryExpression ::=  CHOICE {
                    term       [1] IMPLICIT SEQUENCE{
                                     queryTerm      [1]   Term,
                                     termComment  [2] IMPLICIT InternationalString OPTIONAL}
                    query      [2] Query}
END
```

USR.2 Negotiation Records

Negotiation records are defined for use within the parameters otherInfo (version 3 only) and userInformationField in the Init and InitResponse APDUs. There are no negotiation records defined by this standard. Publicly defined negotiation record definitions are available from the Z39.50 Maintenance Agency.

In general, a negotiation record is defined for use as follows: the origin includes the negotiation record within the Init APDU (identified by its OID) to propose that some condition be in effect for the Z-association. The target may (but is not obligated to) respond to the proposal, using the same negotiation record format, and the target's response, if any, indicates whether the proposal is accepted, or may indicate a counter-proposal, which will then be in effect for the Z-association. Thus a negotiation record definition should include the format of both the origin proposal and the target response.

The following rules and guidelines apply to the definition and use of negotiation records:

- A negotiation record should be defined for the purpose of negotiating a single item of information, except in the following case: negotiation of related items may be defined for the same negotiated record, where it is not practical to separate their negotiation, for example, because of interdependence among the negotiation of these items.
- If the origin does not propose negotiation (i.e. does not submit a negotiation record) for a given item, then it is considered that "no negotiation takes place" for that item. If the origin does propose negotiation for an item, but the target does not respond (i.e. does not include a corresponding negotiation record), similarly, no negotiation takes place for that item.
- A negotiation record definition must not specify behavior governing the condition where no negotiation takes place. (If no negotiation takes place, neither origin nor target can be assumed to know any rules associated with the negotiation record definition.)

- If the target does not recognize the oid for a negotiation record submitted by the origin, it should ignore it (and not return a negotiation record of that type).

  Note: If the target does not recognize the oid of a negotiation record, then it cannot be certain that it is indeed a negotiation record. Therefore, care should be taken in general in defining user information formats, to ensure that if the target does not recognize an oid it may ignore it with impunity.
- If the origin does not submit a negotiation record of a particular type in the Init request, then the target is not to include a negotiation record of that type in the response.
- If multiple negotiation records are included in an Init request, there is no significance to their order, and there is no relationship between them: for example, if the origin includes two negotiation records, and the target does not recognize the first (in which case negotiation fails for the first) negotiation may still succeed for the second.

Appendix ESP

Element Specification Formats

(Normative)

This Standard defines and registers the element specification format eSpec-1.

The object identifier for eSpec-1 is {Z39-50-elementSpec 1}

ElementSpecificationFormat-eSpec-1

```
-- For detailed semantics, see Appendix RET.

{Z39-50-elementSpec eSpec-1 (1)} DEFINITIONS ::=
BEGIN
IMPORTS Variant FROM RecordSyntax-generic, StringOrNumeric, InternationalString FROM
Z39-50-APDU;
--
Espec-1 ::= SEQUENCE{
    elementSetNames          [1] IMPLICIT SEQUENCE OF InternationalString OPTIONAL,
                                   -- Origin may include one or more element set names, each
                                   -- specifying a set of elements. Each of the elements is to be
                                   -- treated as an elementRequest in the form of simpleElement,
                                   -- where occurrence is 1.
    defaultVariantSetId      [2] IMPLICIT OBJECT IDENTIFIER OPTIONAL,
                                   -- If supplied, applies whenever variantRequest
                                   -- does not include variantSetId.
    defaultVariantRequest    [3] IMPLICIT Variant OPTIONAL,
                                   -- If supplied, then for each simple elementRequest that does not
                                   -- include a variantRequest, the defaultVariantRequest applies.
                                   -- (defaultVariantRequest does not apply to a compositeRequest.)
    elements                 [4] IMPLICIT SEQUENCE OF ElementRequest OPTIONAL}


ElementRequest::= CHOICE{
    simpleElement       [1]  IMPLICIT SimpleElement,
    compositeElement    [2] IMPLICIT SEQUENCE{
                        elementList       [1] CHOICE{
                                  primitives     [1]  IMPLICIT SEQUENCE OF InternationalString,
                                                        -- Origin may specify one or more element
                                                        -- set names, each identifying a set of elements,
                                                        -- and the composite element is the union.
                                  specs          [2] IMPLICIT SEQUENCE OF SimpleElement},
                        deliveryTag       [2] IMPLICIT TagPath,
                                             -- deliveryTag tagPath for compositeElement may not
                                             -- include wildThing or wildPath.
                        variantRequest    [3] IMPLICIT Variant OPTIONAL}}
```

```
SimpleElement ::= SEQUENCE{
            path                [1] IMPLICIT TagPath,
            variantRequest      [2] IMPLICIT Variant OPTIONAL}



TagPath ::= SEQUENCE OF CHOICE{
    tag         [1] IMPLICIT SEQUENCE{
                        tagType         [1] IMPLICIT INTEGER OPTIONAL,
                                                -- if omitted, default listed in schema applies
                        tagValue        [2] StringOrNumeric,
                        occurrence      [3] Occurrences OPTIONAL
                                                -- default is "first occurrence"
                                        },
    wildThing   [2] Occurrences,
                        -- Get Nth "thing" at this level, regardless of tag, for each N specified by
                        -- "Occurrences" (which may be 'all' meaning match every element at this level).
                        -- E.g. if "Occurrences" is 3, get third element regardless of its tag or the tag of
                        -- the first two elements.
    wildPath    [3] IMPLICIT NULL
                        -- Match any tag at this level or below that is on a path for which next tag in this
                        -- TagPath sequence occurs. WildPath may not be last member of the TagPath
                        -- sequence.
                                            }
--

Occurrences ::= CHOICE{
    all     [1] IMPLICIT NULL,
    last    [2] IMPLICIT NULL,
    values  [3] IMPLICIT SEQUENCE{
                    start       [1] IMPLICIT INTEGER,
                                    -- if 'start' alone included, then single occurrence requested
                    howMany     [2] IMPLICIT INTEGER OPTIONAL
                                    -- For example, if 'start' is 5 and 'howMany' is 6, then request is for
                                    -- "occurrences 5 through 10".
                                    }
END
```

Appendix VAR

Variant Sets

(Normative)

This standard defines and registers the variant set variant-1.

The object identifier for variant-1 is {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.

For detailed semantics of the use of variant-1, see Appendix RET.

A variant specification is a sequence of triples; each triple is a variant specifier (as referenced by the identifier variantSpecifier in GRS-1 and ES-1). 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:

Class	Type	Value(s)
1 =	**variantId**	

Class 1 may be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Value(s)
1 = variantId	OCTET STRING

2 = BodyPartType

Class 2 may be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Value(s)
1 = ianaType/subType	InternationalString: "<ianaType>/<subType>". e.g. "application/postscript", where <ianaType> and <subType> are registered with IANA (Internet Assigned Numbers Authority)
2 = Z39.50Type/subType	InternationalString: e.g. 'sgml/'dtd
3 = otherType/subType	InternationalString; bilaterally agreed upon

3 = formatting/presentation

Class 3 may be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Value(s)
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")

Class	Type	Value(s)
3 = formatting/presentation *(continued)*		
	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

4 = Language/CharacterSet

Class 4 may be used within a supportedVariant, variantRequest, or appliedVariant.

Type	Value(s)
1 = language	InternationalString (from USMARC code list for languages)
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

5 = Piece

Class 5 may be used within a variantRequest or appliedVariant.

Type	Value(s)
1 = what fragment wanted	INTEGER *(variantRequest only)* 1 = start 2 = next 3 = previous 4 = current 5 = last
2= what fragment returned	INTEGER *(appliedVariant only)* 1 = start 2 = middle 3 = last 4 = end for now 5 = whole
3 = start	IntUnit
4 = end	IntUnit
5 = howMuch	IntUnit
6 = step	INTEGER
7= targetToken	OCTET STRING

6 = meta-data requested

Class 6 may be used within a variantRequest only.

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 = other meta-data	NULL

7 = meta-data returned

Class 7 may be used within a supportedVariant or appliedVariant.

1 = cost	IntUnit
2 = size	IntUnit
3 = integrity	INTEGER
4 = separability	INTEGER
5 = variant supported	BOOLEAN

8 = Highlighting

Class 8 may be used within a variantRequest or appliedVariant.

1 = prefix	OCTET STRING
2 = postfix	OCTET STRING
3 = server default	NULL *(variantRequest only)*

9 = miscellaneous

1 = NoData	NULL *(variantRequest only)*
2 = Unit	Unit *(variantRequest only -- origin requests element according to specific unit)*
3 = Version	InternationalString

Appendix TAG

TagSet Definitions and Schemas

(Normative)

A database schema represents a common understanding shared by the origin and target 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.2.1)
2	an element defined in tagSet-G	(see TAG.2.2)
3	a tag locally defined by the target (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.

TAG.1 Schema Definitions

This standard registers the following object identifiers for Schemas:

WAIS	{Z39-50-schema 1}
GILS	{Z39-50-schema 2}

TAG.2 TagSet Definitions

This standard defines and registers the tag set definitions tagSet-M and tagSet-G. TagSet-M includes elements intended for use as meta-data associated with 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.

TAG.2.1 Definition of tagSet-M

Element	tag	recommended ASN.1 datatype
schemaIdentifier	1	OBJECT IDENTIFIER
elementsOrdered	2	BOOLEAN
elementOrdering	3	INTEGER
defaultTagType	4	INTEGER
defaultVariantSetId	5	OBJECT IDENTIFIER
defaultVariantSpec	6	VariantSpec
processingInstructions	7	InternationalString
recordUsage	8	INTEGER
restriction	9	InternationalString
rank	10	INTEGER
userMessage	11	InternationalString
url	12	InternationalString
record	13	structured
local control number	14	InternationalString
creation date	15	GeneralizedTime
dateOfLastModification	16	GeneralizedTime
dateOfLastReview	17	GeneralizedTime
score	18	INTEGER
wellKnown	19	InternationalString
recordWrapper	20	structured

schemaIdentifier Identifies the schema in use. This element is available for cases where the origin does not specify a Schema in the request, or where the target uses a schema different than that requested by the origin.

elementsOrdered 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).

elementOrdering 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.

defaultTagType The tagType that applies for any element for which tagType is not included.

defaultVariantSetId The Variant set identifier that applies when the target returns a variant specification for an element, but does not include a variant set identifier.

defaultVariantSpec If this element is present, then the specified variant applies to all subsequent elements, when applicable, which do not include a variant specification.

processingInstructions Recommendation by the target on how to display this record to the user.

recordUsage
1 = redistributable.
2 = restricted, and the tagSet-M element 'restriction' (defined below) contains the restriction.
3 = restricted, and the 'restriction; contains a license pointer.

restriction 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).

rank The rank of this record within the result set. If there are N records in the result set, each record should have a unique rank from 1 to N.

userMessage A message pertaining to this record, that the target asks the origin to display to the user.

url Uniform resource locator. This is a URL for the record.

record 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.

localControlNumber An identifier of the record, unique within the database.

creationDate Date that the record was created.

dateOfLastModification Most recent date that this record was modified.

dateOfLastReview Most recent date that this record was verified.

score A normalized score assigned to the record by the target. Each record in the result set should 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.

wellKnown When an element is defined to be "structured into locally defined elements", the target 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 target might present the element structured into the following subelements: 'wellKnown', "spineTitle", and "variantTitle", where the latter two are string tags, target defined. In this case, 'wellKnown' is assumed to mean "title".

recordWrapper This element may be used to represent the root of the record, particularly when the record otherwise has no root. The origin may request the record skeleton by reference to this element.

TAG.2.2 Definition of tagSet-G

Element	tag	recommended ASN.1 datatype
title	1	InternationalString
author	2	InternationalString
publicationPlace	3	InternationalString
publicationDate	4	InternationalString
documentId	5	InternationalString
abstract	6	InternationalString
name	7	InternationalString
date	8	GeneralizedTime
bodyOfDisplay	9	InternationalString
organization	10	InternationalString
postalAddress	11	InternationalString
networkAddress	12	InternationalString
eMailAddress	13	InternationalString
phoneNumber	14	InternationalString
faxNumber	15	InternationalString
country	16	InternationalString
description	17	InternationalString
time	18	IntUnit

These elements (with the exception of bodyOfDisplay) are for generic use and their definitions are not supplied.

BodyOfDisplay The target might combine several elements into this single element, into a display format, for display.

Appendix 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 origin-specified criteria, and the creation by the target of a result-set representing the selected records. *Retrieval*, idiomatically speaking, is the transfer of result set records from the target to the origin.

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 an origin 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 target 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 <u>retrieval record</u>. (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 origin may request specific logical information <u>element</u>s from a record (via an element specification, described below).
- The origin and target may share a name space for tagging elements (via a schema and tagsets, described below), so that elements will be properly identified: by the origin, within an element specification, and by the target, within a retrieval record.
- The origin may request an individual element according to a specific representation or format (via variants, described below).
- The origin 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 target, when that element is included in a retrieval record.
- The target 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 target 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 origin 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 origin 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 origin and target are assumed to share a common definition, a <u>tagSet</u>, which assigns integers to elements. The integer is the name, or <u>tag</u>, 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. <u>String</u> 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 <u>tagType</u>. 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. An origin 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 OPTIONAL,
        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). An origin 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 origin may wish to qualify an elementRequest with a variantRequest, to specify a particular composition (e.g. PostScript), 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 origin and target 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.

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 target system.
objectElement	yes	no	Contains object information for the record. It may be text, image, etc.

In this example, for objectElement, the schema would indicate that the target 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 2, 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 origin will recognize it as the objectElement.

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 origin 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 origin would recognize the type of information, e.g. fingerprint, from its string tag; but the origin 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>)	ObjectElement	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:

Tag	Element Name	Recommended DataType
25	linkage	InternationalString
27	recordSource	InternationalString
51	purpose	InternationalString
52	originator	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)	originator
(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, originator, 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 origin may request an element (in a Present request) according to a specific variant;
- the target may present an element (in a Present response) according to a specific variant;
- the target 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 target 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);
- OPAC and Summary (REC.3 and REC.4); and
- 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-1, 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-1 it is assumed that GRS-1 is to be used as the record syntax. In general, however, no such restriction applies; eSpec-1 may be used as an element specification in conjunction with SUTRS for example.

RET.3.1 Element Specification Format eSpec-1

The element specification format eSpec-1 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, and 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 origin 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 origin 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:

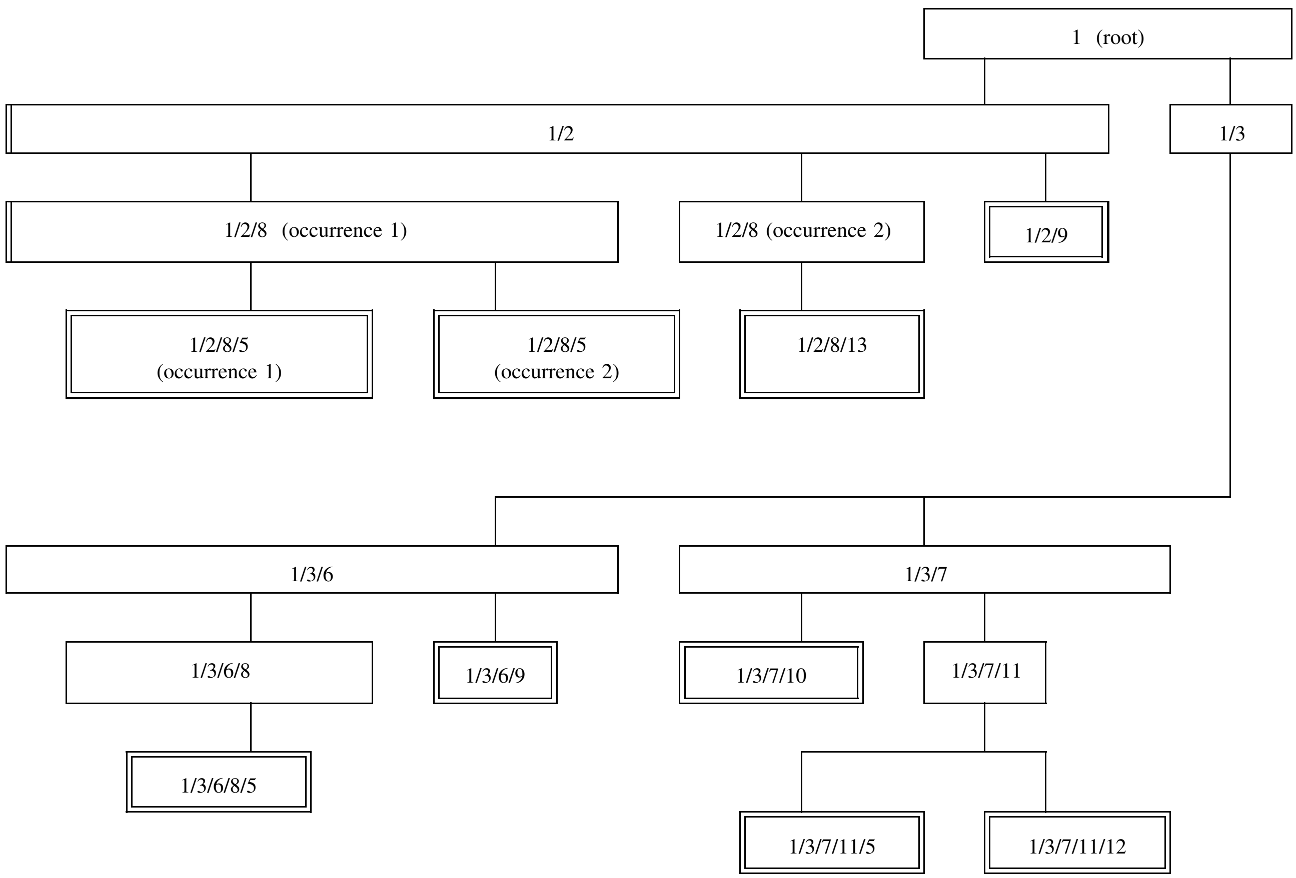
- If a tagPath identifies a non-leaf node, the request represents the entire subtree (it is logically equivalent to individual simple requests for each subordinate leaf-node).
- 'occurrence' may be specified as 'all, meaning "all nodes with a given tag".
- 'occurrence' may be specified in the form of 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.

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.

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-1 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 occurs 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 include a variantSetId, then 'defaultVariantSet' applies. (If the element request includes a variantRequest which does 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, and an optional variantRequest. The simple elements are to be combined by the target to form a single (logical) element, to which the (optional) composite variant is to be applied, and the target 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-1 element specification. A non-leaf node may correspond to an elementRequest; if an eSpec-1 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 for 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, and 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:

- o *usageRight*: the target may declare that the element is freely distributable, or that restrictions apply. In the latter case, the target supplies either a restriction in the form of a text message, or a license pointer.
- o *hits*; see RET.3.2.3.1.
- o *displayName*: A name for the element, suggested by the target, for the origin to display.
- o *supportedVariants*; see RET.4.
- o *message*: A message for the origin 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:

- o *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 target might return hit vectors along with an element, so that the origin 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 target 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 origin 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 target 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 target may assign a token to the hit vector, which points to the fragment associated with the hit. The origin 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 target might provide location information, or a token, which may be used subsequently to retrieve the specific fragment. The target might provide both location information and a token: for example, the location information might indicate "page 10"; the origin may subsequently retrieve the pages before and after, inclusive (i.e pages 9-11). If the target also supplies a token, the origin 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 origin 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 origin could subsequently use a token in an elementRequest to retrieve the corresponding fragment, independent of what variantRequest was included in the elementRequest.

The origin might request 'hits: variant-specific' for an element, for a particular variant. The target might return location information or tokens, or both, but in any case, the hit vectors would apply only for that variant. The origin 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 target 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 target (see 3.3.3.7.1). It consists of a list of supportedVariants for the element. Each may include a variantId, which may be used subsequently by the origin 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 target in case the origin 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.

Type	Subtype
text	plain
	richtext
	tab-separated-values
application	octet-stream
	postscript
	oda
dx	wordperfect5.1
	pdf
	zip
	macwriteii
	msword
image	jpeg
	gif
	ief
	tiff
audio	basic
video	mpeg

PostScript, for example, would be indicated by the triple (2,1, 'application/postscript'). SGML is not registered yet by IANA, so it is registered as a Z39.50 body part type. It may be indicated by (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 origin 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 targetToken, see type 7);
- the 'previous' fragment;
- the 'current' fragment (the fragment indicated by targetToken);
- the 'last' fragment (within the element).

The target 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 target 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 target 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 origin 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 targetToken, 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 target token.

The target may use type 7 in an appliedVariant to supply a token as an identifier of the supplied fragment, and the origin 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 origin might want to know, for example, the cost to retrieve a particular element in PostScript, as well as the page count (of the PostScript form of the element). The following variant specifiers would be included within the variantRequest for that element:

```
(2,1, 'application/postscript')      -- PostScript
(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 postScript specifier, in this example) if the origin knows the variantId of a variant for which it wants cost or size information. (Although if the origin 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 target may have included the cost and page information within the supportedVariant.)

The origin 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. postScript), using (6,3, NULL) or non-variant-specific hits, using (6,4, NULL). In either case, the request is for the target to return hit vectors within the retrieved GRS record.

The origin may request that the target supply the variant list for an element via the specifier (6,5, NULL). The target 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 origin 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 supportedVariant. 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 target 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 origin may include 'prefix' and 'postfix' in a variantRequest to request that the target insert the specified strings into the actual data, surrounding hits, so that the origin, upon retrieving the data, may simply locate the strings, for fast access to the hits. The origin may use 'server default' in lieu of 'prefix' and 'postfix' to indicate that it the target should select the strings for highlighting.

The target 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 target may depend on the implementation. For example, for an element (representing a document) which the target provides in PostScript, consider the following cases:

- The document might already exist in print format, and the target might support only that single postScript variant.
- The target might support a few variants forms, varying by language.
- The target might support many variant forms; varying by language.
- The target might support many variant forms, varying by language, and also varying by formatting/presentation parameters, including lines per page, font style, etc.

The target might list a single supportedVariant in the variant list for the element, indicating that the element is available in postScript. In that case the origin cannot necessarily conclude which of the above cases applies. The target might instead list three supportedVariants, each indicating postScript and a language. In that case, it may be reasonable for the origin to surmise that the element is available in those three languages only, but the origin probably cannot deduce which formatting parameters apply. The target might further indicate one or more formatting parameters within each supportedVariant. Again, the extend to which the origin may deduce what other variations are supported will depend on the implementation.

The origin 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 target provide cost (6,1, NULL) and size (6,2, NULL) information if the variant does exist. The target 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 target indicates TRUE (that the variant is supported) it may also supply a variantId that the origin may then use to request the variant.

The origin may construct a variantRequest that includes a variantId along with additional variant specifiers.

Suppose the target lists the following supportedVariant:

(1,1, <variantId>) -- identifies this variant
(2,1, 'application/postscript') -- in postScript
(4,1, 'por') -- language: Portuguese

The element is thus available in PostScript, in Portuguese. The origin may submit a variantRequest consisting of only:

(1,1, <variantId>)

to request the element in postScript, in Portuguese.

Suppose, instead, the target lists the following supportedVariant:

(1,1, <variantId>) -- identifies this variant
(2,1, 'application/postscript') -- in postScript

Thus the target indicates that the element is available in PostScript, but no other variant information is provided.

The origin may submit a variantRequest consisting of only:

(1,1, <variantId>
(4,1 'por')

Again, this is to request the element in postScript, in Portuguese.

Or, the origin 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 postScript, 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) 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 target 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. Within a GRS-1 record, the target 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*; they do not pertain to elements within the database record which may in fact be meta-information about some object other than the record itself.

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 meaningful only within the context of a schema definition. In many (perhaps most) cases the target may reasonably expect that the origin knows which schema definition applies to a particular retrieval record. In those cases the target 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, should normally 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 target wishes to present elements in numerical order (see RET.3.4.1.2.2).

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 an origin 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 target 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 tagType 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 target 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 defaultVariantSetId is not provided, then every appliedVariant within the retrieval record must include a variantId. If the schema does not list a default appliedVariant 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 targets 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 target simply return a single value, what the target 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 target wishes to return the element 'title' encompassing several forms of the title, including one which the origin 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 origin does not). The origin 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 origin 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 bodyOfDisplay.

RET.3.4.2.1 bodyOfDisplay

The target might combine several elements of a record into this single element, into a display format, for the origin to display to the user.

For a given schema, perhaps for a particular application, some origins may need the target to distinguish all elements in a retrieval record, perhaps because the origin is going to replicate the record. In other cases, the origin is satisfied for the target to package all elements into display format for direct display to the end-user. In either of these cases, bodyOfDisplay is not applicable (in the latter case the target may use the SUTRS record syntax instead of GRS-1).

In some cases though, the origin may need some of the elements distinguished, but is satisfied to have the target package the remaining elements into a single retrieval element for display. In these cases bodyOfDisplay may be useful.

Suppose the target wishes to present 20 elements of a record, but only the first three elements are intended for origin use, and the remaining elements are intended to be transparently passed to the user. Rather than packaging all 20 elements, the target instead may send 4 elements, where the 4th delivery element packages the latter 17 original elements, in a display format.

The bodyOfDisplay 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 bodyOfDisplay differs from a composite element in three respects:

- The target, not the origin, selects the subset of elements for packaging.
- In a composite element there may be semantics conveyed by the tag that the origin 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 bodyOfDisplay element has no semantics other than telling the origin "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 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 target 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 target might maintain to perform proximity operations requiring the extended model, or to evaluate retstriction operands.

1. Extended Result Set Model for Proximity

In the extended result set model for proximity, the target 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 target 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 target 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 target 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 target maintains (associated with result set entries to be used in the proximity operations) nor the manner in which the target maintains this information, is prescribed by the standard. The above is supplied as an example only.

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. 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, 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 target maintains information that allows this type of search to be performed. In this example, the target 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, there are two databases, R and C. 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 which 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 which 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 target maintains information that allows this type of search to be performed. In this example, the target 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 target maintains (associated with result set entries to be used in the evaluation of a Restriction operand), nor the manner in which the target maintains this information, is prescribed by the standard. The above are supplied as an example only.