RETS 1.7 Specification . . . 3
List of Tables & Figures . . . 4
SECTION 1 - Introduction . . . 6
1.1 Purpose . . . 6
1.2 Scope . . . 7
1.3 Requirements . . . 7
1.4 Terminology . . . 7
SECTION 2 - Notational Conventions . . . 8
2.1 Augmented BNF . . . 8
2.2 Typographic Conventions . . . 9
2.3 Rules . . . 9
2.4 Atoms and Primitive Entities . . . 9
SECTION 3 - Message Format . . . 10
3.1 General Message Format . . . 11
3.2 Request Format . . . 11
3.3 Required Client Request Header Fields . . . 11
3.4 Optional Client Request Header Fields . . . 12
3.5 Response Format . . . 13
3.6 Required Server Response Header Fields . . . 14
3.7 Optional Server Response Header Fields . . . 15
3.8 Data Compression in RETS Transactions . . . 16
3.9 General Status Codes . . . 16
3.10 Computation of the RETS User Agent Authorization value . . . 17
SECTION 4 - Login Transaction . . . 18
4.1 Security . . . 18
4.2 Authorization Example . . . 19
4.3 Required Request Arguments . . . 19
4.4 Optional Request Arguments . . . 19
4.5 Optional Response Header Fields . . . 20
4.6 Login Response Body Format . . . 20
4.7 Required Response Arguments . . . 20
4.8 Optional Response Arguments . . . 22
4.9 Well-Known Names . . . 23
4.10 Capability URL List . . . 23
4.11 Reply Codes . . . 24
SECTION 5 - GetObject Transaction . . . 25
5.1 Required Client Request Header Fields . . . 25
5.2 Optional Client Request Header Fields . . . 26
5.3 Required Request Arguments . . . 26
5.4 Optional Request Arguments . . . 26
5.5 Required Server Response Header Fields . . . 27
5.6 Optional Server Response Header Fields . . . 27
5.7 Required Response Arguments . . . 28
5.8 Optional Response Arguments . . . 28
5.9 Metadata . . . 28
5.10 Resources . . . 28
5.11 Multipart Responses . . . 28
5.12 Reply Codes . . . 29
SECTION 6 - Logout Transaction . . . 30
6.1 Required Request Arguments . . . 30
6.2 Optional Request Arguments . . . 30
6.3 Required Response Arguments . . . 30
6.4 Optional Response Arguments . . . 31
6.5 Logout Response Body Format . . . 31
6.6 Reply Codes . . . 31
SECTION 7 - Search Transaction . . . 32
7.1 Search Types . . . 32
7.2 Search Terminology . . . 32
7.3 Required Request Arguments . . . 33
7.4 Optional Request Arguments . . . 33
7.5 Required Response Arguments . . . 35
7.6 Search Response Body Format . . . 35
7.7 Query language . . . 37
7.8 Reply Codes . . . 39
SECTION 8 - Get Transaction . . . 40
8.1 Required Request Arguments . . . 40
8.2 Optional Request Arguments . . . 40
8.3 Required Response Arguments . . . 40
8.4 Optional Response Arguments . . . 40
8.5 Status Conditions . . . 41
SECTION 9 - Change Password Transaction . . . 41
9.1 Required Request Arguments . . . 41

9.2 Optional Request Arguments . . . 41
9.3 Required Response Arguments . . . 41
9.4 Optional Response Arguments . . . 41
9.5 Reply Codes . . . 41
9.6 Encryption Key Construction . . . 41
9.7 ECB Padding . . . 42
9.8 Effect of change . . . 42
SECTION 10 - Update Transaction . . . 42
10.1 Required Request Arguments . . . 42
10.2 Optional Request Arguments . . . 43
10.3 Required Response Arguments . . . 43
10.4 Optional Response Arguments . . . 43
10.5 Update Response Body Format . . . 43
10.6 Validation . . . 45
10.7 Reply Codes . . . 46
SECTION 11 - Metadata Format . . . 46
11.1 Organization and Retrieval . . . 46
11.2 System-Level Metadata . . . 48
11.3 Metadata Format for Class Elements . . . 54
11.4 Metadata Format for Shared Elements . . . 62
SECTION 12 - GetMetaData Transaction . . . 75
12.1 Required Client Request Header Fields . . . 75
12.2 Required Request Arguments . . . 75
12.3 Optional Request Arguments . . . 75
12.4 Required Server Response Header Fields . . . 76
12.5 Required Response Arguments . . . 76
12.6 Optional Response Arguments . . . 76
12.7 Metadata Response Body Format . . . 76
12.8 Reply Codes . . . 77
SECTION 13 - Compact Data Format . . . 78
13.1 Overall format . . . 78
13.2 Decoded Format . . . 78
13.3 Transmission standards . . . 78
SECTION 14 - Session Protocol . . . 79
14.1 Connection Establishment . . . 79
14.2 Authorization . . . 79
14.3 Session . . . 79
14.4 Termination . . . 80
SECTION 15 - ServerInformation Transaction . . . 80
15.1 Required Request Arguments . . . 80
15.2 Optional Request Arguments . . . 80
15.3 Response Format . . . 80
15.4 Well-known names . . . 81
15.5 Reply Codes . . . 82
SECTION 16 - Acknowledgements . . . 82
SECTION 17 - Authors . . . 83
SECTION 18 - References . . . 84
Appendix A - DTD References . . . 85
Appendix B - Sample Compact Metadata Response . . . 86
Appendix C - Summary of RETS Reply Codes . . . 92
Appendix D - Interpretation of Maximum Field Length and Display Information . . . 96

RETS 1.7 Specification

Chapters

SECTION 1 - Introduction

SECTION 2 - Notational Conventions

SECTION 3 - Message Format

SECTION 4 - Login Transaction

SECTION 5 - GetObject Transaction

SECTION 6 - Logout Transaction

SECTION 7 - Search Transaction

SECTION 8 - Get Transaction

SECTION 9 - Change Password

SECTION 10 - Update Transaction

SECTION 11 - Metadata Format

SECTION 12 - Get Metadata Format

SECTION 13 - Compact Data Format

SECTION 14 - Session Protocol

SECTION 15 - ServerInformation Transaction

SECTION 16 - Acknowledgements

SECTION 17 - Authors

SECTION 18 - References

Additional Sections

List of Tables & Figures

Appendix A - DTD References

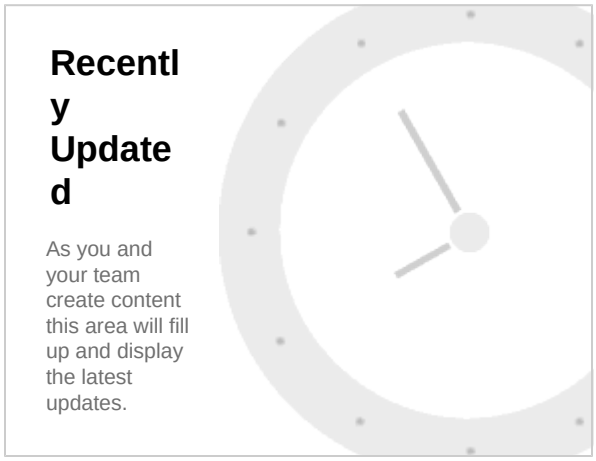
Appendix B - Sample Compact Metadata Response

Appendix C - Summary of RETS Reply Codes

Appendix D - Interpretation of

Maximum Field Length and
Display Information

RETS 1.7 Recent News & Updates



List of Tables & Figures

List of Tables

3-1 General Status Codes

4-1 Well-Known Names for Input Fields

4-2 Capability URL Descriptions

4-3 Valid Reply Codes for Login Transaction

5-1 GetObject Reply Codes

6-1 Logout Reply Codes

7-1 Search Transaction Reply Codes

9-1 Change Password Reply Codes

10-1 Update Transaction Reply Codes

11-1 Well-Known Resource Names

11-2 Resource Metadata Compact Header Attributes

11-3 Metadata: Resource Description Fields

11-4 ForeignKeys Metadata Compact Header Attributes

11-5 Metadata Content: Foreign Keys

11-6 Class Metadata Compact Header Attributes

11-7 Metadata Content: Resource Class

11-8 Table Metadata Compact Header Attributes

11-9 Metadata Content - Tables

11-10 Update Metadata Compact Header Attributes

11-11 Metadata Content – Update

11-12 UpdateType Metadata Compact Header Attributes

11-13 Metadata Content – Update Type

11-14 Well-known Object Types

11-15 Object Metadata Compact Header Attributes

11-16 Metadata Content: Resource Object

11-17 Lookup Metadata Compact Header Attributes

11-18 Metadata Content: Lookup

11-19 Lookup Type Metadata Compact Header Attributes

11-20 Metadata Content: Lookup Type

11-21 Search Help Metadata Compact Header Attributes

11-22 Metadata Content: Search Help

11-23 EditMask Metadata Compact Header Attributes

11-24 Metadata Content: Edit Mask

11-25 RETS Regular Expression Metacharacters

11-26 Update Help Metadata Compact Header Attributes

11-27 Metadata Content: Update Help

11-28 ValidationLookup Metadata Compact Header Attributes

11-29 Metadata Content: Validation Lookup

11-30 Validation Lookup Type Metadata Compact Header Attributes

11-31 Metadata Content: Validation Lookup Type

11-32 Validation Expression Types

11-33 Validation Expression Operators

11-34 Validation Expression Special Operand Tokens

11-35 Validation Expression Metadata Compact Header Attributes

11-36 Metadata Content: Validation Expression

11-37 Validation External Metadata Compact Header Attributes

11-38 Metadata Content: Validation External

11-39 Validation External Type Metadata Compact Header Attributes

11-40 Metadata Content: Validation External Type

12-1 GetMetadata Reply Codes

13-1 Compact Data Format Representation

15-1 Well-Known Parameter Names

15-2 ServerInformation Reply Codes

A-1 DTD References

C-1 Consolidated list of RETS reply codes

List of Figures

11.1 Metadata Structure

SECTION 1 - Introduction

Section 1 contains the following:

- 1.1 Purpose
- 1.2 Scope
- 1.3 Requirements
- 1.4 Terminology

1.1 Purpose

The Real Estate Transaction Standard (RETS) is a specification for a standard communication method between computer systems exchanging real estate information. It defines a standard interface for use by applications such as agent desktop software, IDX (Internet Data Exchange) systems, data aggregation systems, and many other systems that store, display or operate on real estate listing, sales and other data.

This specification describes the Real Estate Transaction Standard communication protocol. Together with the companion XML DTDs (Document Type Definitions) listed in Appendix A, it constitutes the specification for the standard.

1.2 Scope

This specification is intended to define only the minimum a product or service must do in order to be considered "compliant". This specification is extensible and nothing in the specification precludes a vendor from adding data or functionality over and above that detailed here. However, when a function is provided or a data element is stored by a compliant system, it must offer access to the function or mechanism in a way that complies with the specification in order to be considered compliant.

1.3 Requirements

1.3.1 Required Features

This specification uses the same words as RFC 1123 [1] for defining the significance of each particular requirement. These words are:

MUST	This word or the adjective "required" means that the item is an absolute requirement of the specification. A feature that the specification states MUST be implemented is required in an implementation in order to be considered compliant.
SHOULD	This word or the adjective "recommended" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course. A feature that the specification states SHOULD be implemented is treated for compliance purposes as a feature that may be implemented.
MAY	This word or the adjective "optional" means that this item is truly optional. A feature that the specification states MAY be implemented need not be implemented in order to be considered compliant. However, if it is implemented, the feature MUST be implemented in accordance with the specification.

An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be "conditionally compliant."

Client and server implementations should generally follow the Internet protocol convention of being strict in what they generate, but tolerant in what they accept. However, in cases where tolerance of deviations from the specification could result in an incorrect interpretation of user data or intentions, implementors are urged to reject transactions rather than supplying possibly-incorrect defaults.

1.3.2 Compatibility with Prior Versions

The RETS 1.7 specification supersedes previous versions of the RETS specification. There is no *requirement* for a client or server that advertises itself as "compliant with RETS 1.7" to interoperate with earlier versions. However, client and server implementers are urged to support the prior version, RETS 1.5, in order to insure a smooth transition.

1.4 Terminology

Class	A subset of data elements within a Resource that share common metadata elements.
Client	The system requesting data. This may well be a server seeking to update itself from another server. The specification does not assume any particular kind of client.
Endpoint	Either a server or client.
Metadata	The set of data that describes data fields in detail.
Metadata Dictionary	The set of data that describes the available metadata. It is used to determine the different classes of accessible data on the server and does not describe the fields within the those classes. It also defines what different types of searches are available (tax, open house, etc.)
Object	For purposes of RETS and its GetObject transaction, a collection of octets treated as a unit and associated with a unique resource element.
Optional	A field or feature described by this specification but not required for an endpoint to be considered compliant. The specification states the action to be taken by a compliant system in the absence of an optional field. The fact that the specification designates a field as optional does not mean that the recipient of a transaction that is missing optional fields is required to provide all services that could be required if the field were present.
Required	A compliant server or client MUST include any field designated as required. A transaction that does not include every required field MUST be rejected by the recipient.
Resource	A collection of data having the external appearance of belonging to a single database and being accessible for search or update via RETS transactions.
Resource Element	An individual record from a resource identified by a Resource Key.
Resource Key	The unique key that identifies a resource element.
Server	The system providing data (also referred to as the "host").
Request ID	A client-provided character string of up to 64 printable characters which uniquely identifies a request to a client. The contents are implementation-defined. Defined in Section 3.4, "Optional Client Request Header Fields".
Standard-Name	The name of a data field as it is known in the Real Estate Transaction Standard Data Dictionary.
System-Name	The name of a data field as it is known in the metadata.

SECTION 2 - Notational Conventions

- 2.1 Augmented BNF
- 2.2 Typographic Conventions
- 2.3 Rules
- 2.4 Atoms and Primitive Entities

2.1 Augmented BNF

This document expresses message layouts and character sequences in an augmented Backus-Naur Form (BNF) similar to

that used by RFC 2822 [4].

2.2 Typographic Conventions

Parsing constructs and examples are set in a `monospaced` font:
`Server: Microsoft-IIS/4.0`

In parsing constructs, textual elements that are required exactly as shown are indicated by **`boldface type`**., while textual elements that represent placeholders for actual data are indicated by a *`slanted font:`*
`Server:`*`server identifier`*

Entities designated by a textual definition contain that definition enclosed in angle brackets:
<any 8-bit sequence of data>

Atoms and primitive entities are indicated by *`ITALIC CAPS`*
`1*64ALPHANUM`

Three nonprinting characters also have significance in some RETS constructs. These may be represented by special printing graphics:

	Tab character, ASCII HT, an octet with a value of 9
.	Space character, ASCII SP, an octet with a value of 32. The symbol is used where needed for clarity.

2.3 Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986 [5].

Parsed entities are constructed combinations of atoms or other entities as defined below. Atoms may be combined and repeated to form longer constructs. When there are constraints on the repetition of atoms, the constraints are expressed by a notation of the form:
*m * n*

where both *m* and *n* are integers. *m* represents the minimum allowed number of repetitions, and *n* represents the maximum. If *m* is omitted, it is presumed to be zero; if *n* is omitted, it is presumed to be infinite. For example, the syntactic construct

`1*64ALPHANUM`

means a string of *`ALPHANUM`* s containing at least 1 and at most 64.

When a parsing construct is represented by a string of entities, some of which are optional, the optional entities are enclosed in square brackets. For example, in the string

*`error-number [error-code]`*the `error-number` entity is required, while the `error-code` entity is optional.

Alternation is indicated by the vertical bar. The entity description
`ALPHA | DIGIT`

means “either an *`ALPHA`* or a *`DIGIT`* ”.

2.4 Atoms and Primitive Entities

`OCTET`	: : = <any 8-bit sequence of data>
`CHAR`	: : = <any US-ASCII character (octets 0 - 127)>

UPALPHA	::= <any US-ASCII uppercase letter "A".."Z">
LOALPHA	::= <any US-ASCII lowercase letter "a".."z">
ALPHA	::= *UPALPHA* | *LOALPHA*
DIGIT	::= <any US-ASCII digit "0".."9">
ALPHANUM	::= *ALPHA* | *DIGIT*
SQLFIELDNAME	::= *ALPHA *31ALPHANUM* <except ANSI SQL 92 reserved words>
IDALPHA	::= *ALPHA* | _
CTL	::= <any US-ASCII control character (octets 0 - 31) and DEL (127)>
CR	::= <US-ASCII CR, carriage return (13)>
LF	::= <US-ASCII LF, linefeed (10)>
SP	::= <US-ASCII SP, space (32)>
HT	::= <US-ASCII HT, horizontal-tab (9)>
<"> or "	::= <US-ASCII double-quote mark (34)>
NULL	::= <no character>
CRLF or	::= *CR LF*
LWS	::= *[CRLF] 1*(SP | HT)*
HEX	::= "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f" | *DIGIT*
LHEX	::= "a" | "b" | "c" | "d" | "e" | "f" | DIGIT
TEXT	::= <any *OCTET* except *CTL*s, but including *LWS*>
PLAINTEXT	::= <any *OCTET* except *CTL*s
TSPECIALS	::= "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | <"> | "/" | "[" | "]" | "?" | "=" | "}" | "{" | SP | HT
TOKENCHAR	::= <any *CHAR* except *CTL*s or *TSPECIALS* >
TOKEN	::= 1**TOKENCHAR*
QUOTED-STRING	::= (<"> *(*QDTEXT*) <">)
QDTEXT	::= <any *TEXT* except <">>
DATE	::= Date using the format defined in RFC 1123.
RETSNAME	::= *1*32IDALPHA*

SECTION 3 - Message Format

RETS uses HTTP version 1.1 [2] for sending messages between clients and servers. It defines three additional HTTP headers, and some RETS transactions constrain the values of certain headers defined by HTTP 1.1 and/or make certain

headers designated as optional in HTTP 1.1 mandatory when used for RETS. In addition, RETS requests use HTML 4.01 [16] form encoding to encapsulate request parameters. In addition, a compliant RETS client MUST implement cookie handling as specified in RFC 2109 [15].

The information below summarizes some of the requirements of HTTP 1.1 and HTML 4.01 for ease of reference. However, in all cases, the underlying standards are the normative references for message formats.

- 3.1 General Message Format
- 3.2 Request Format
- 3.3 Required Client Request Header Fields
- 3.4 Optional Client Request Header Fields
- 3.5 Response Format
- 3.6 Required Server Response Header Fields
- 3.7 Optional Server Response Header Fields
- 3.8 Data Compression in RETS Transactions
- 3.9 General Status Codes
- 3.10 Computation of the RETS User Agent Authorization value

3.1 General Message Format

3.1.1 RETS HTTP/1.1 Encapsulation

RETS messages are encapsulated as the bodies of HTTP/1.1 requests and responses. The request body may be null, depending on the request. The response body is never null

Note that, per RFC 2822, keywords in header key-value pairs are not case-sensitive. The values, however, may be case-sensitive depending on context.

3.1.2 Request Arguments

RETS requests are HTML 4.01-compliant form submissions, following all of the specifications in the HTML 4.01 recommendation. Note that the HTML 4.01 specification provides that:

- Key names in key/value pairs are not case-sensitive.
- Both key names and key values MUST be encoded as specified in HTML 4.01 section 17.13.4, with + characters replacing spaces, and then reserved characters being escaped per RFC 2396 [13], unless the client uses a content-type of multipart/form-data.

3.1.3 Response Bodies

The body of a response to most RETS requests is a well-formed XML document; the exceptions are the Get transaction (section 8) and the GetObject transaction (section 5) . This means that servers must construct the body in accordance with the XML specification [17], and that clients must parse the body in accordance with that specification.

3.2 Request Format

A RETS request is either an HTTP GET request or an HTTP POST request. In the case of the GET-request the Argument-List is appended to the Request-URI after a delimiting question mark ("?"). For the post-request the Argument-List is sent as the first entity body for the POST method.

get-request	::= **GET** *Request-URI* [**?** *Argument-List*] *HTTP-VersionCRLF* {{}} **message-header* *CRLF*
post-request	::= **POST** *Request-URI HTTP-VersionCRLF* **message-header* *CRLF* [*Argument-List*]

The *Request-URI*, *HTTP-Version* and *message-header* are defined in RFC 2616. The detailed construction of the *Argument-List* is defined in HTML 4.01.

3.3 Required Client Request Header Fields

The HTTP header of any messages sent from the client MUST contain the following header fields:

`User-Agent`	This header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations, as well as providing enhanced capabilities to some user-agents. All client requests MUST include this field. This is a standard HTTP header field as defined in RFC 2616.
`User-Agent`	`::= User-Agent: 1* product`
`product`	`::= token [/ product-version]`
`product-version`	`::= token`

Example: `User-Agent: CMAZilla/4.00`

Product tokens should be short and to the point: use of them for advertising or other non-essential information is explicitly forbidden. Although any token character may appear in a product-version, this token SHOULD only be used for a version identifier (i.e., successive versions of the same product SHOULD only differ in the product-version portion of the product value). For more information about User-Agent see RFC 2616.

A server MAY advertise additional capabilities based on the client's User-Agent, and MAY refuse to proceed with the authorization if an acceptable User-Agent has not been supplied. A server MAY also choose to authenticate the client's identity cryptographically using the RETS-UA-Authorization header; see section 3.4 for additional information.

`RETS-Version`	The client MUST send the RETS-Version. The convention used is a "<major>.<minor>" numbering scheme similar to the HTTP Version in Section 3.1 of RFC 2616. The version of a RETS message is indicated by a RETS-Version field in the header of the message.
`Cookie`	The client MUST implement cookie handling as specified in RFC 2109. If any server response has included a valid Set-Cookie header, and the cookie in that header has not expired, the client MUST return the corresponding Cookie header. See RFC 2109 for the full specification.

3.4 Optional Client Request Header Fields

`Authorization`	Authorization header field as defined in RFC 2617. See 4.1, "Security", as well as RFC 2617, for additional information.
`RETS-Request-ID`	A character string of printable characters which the client can use to identify this request. The contents are implementation-defined. If this field is included in a request from the client then the server MUST return it in the response.
`RETS-Request-ID`	`::= 1*64ALPHNUM`
`Accept-Encoding`	A comma-separated list of MIME types indicating the content encoding schemes that the client is willing to accept. This is intended to support the use of compression in data returns; see section 3.8 for additional information.

Accept-Encoding	::= *1*64ALPHNUM/1*64ALPHNUM *[,1*64ALPHNUM/1*64ALPHNUM…]*
RETS-UA-Authorization	A client MAY support authentication of its User-Agent value by including the RETS-UA-Authorization header. Servers MAY require this header with a valid value before providing services.
RETS-UA-Authorization	::= ua-method ua-digest-response
ua-method	::= **Digest**
ua-digest-response	::= "*LHEX " See section 3.10 for the method of computing the ua-digest-response value. The client MAY send this header under any circumstances. It need not send this header if the server has not indicated that it requires user-agent authentication by responding to a transaction with a RETS error code of 20037.

In addition to the header fields listed here, the client may send any header compliant with HTTP 1.1.

3.5 Response Format

The general server response to a request is either a well-formed XML document returning RETS-encapsulated data or error information, or, for the Get transaction and for successful GetObject transactions, the content of the requested object in the format given in the response's HTTP Content-Type header. Note that this is an ordinary HTTP response per RFC 2616.

The more common HTTP *Status-Code*s are provided in Section 3.9 , though any status code defined in RFC 2616 is permissible. Servers MUST use appropriate predefined status codes when communicating with the client.

The *Status-Code* is intended to provide HTTP level errors to the client (Authorization, URI, etc.). Software level errors (search queries, invalid argument values, etc.) should be returned in the reply-code. If the server is unable to determine that a particular request is in fact a RETS request, it MUST return an HTTP status code indicating the type of error.
Except in those transactions specifically stating otherwise, a RETS response body is a well-formed XML document with the following general form:

response-body	::=*RETS-response*
RETS-response	::=*bodystartline* *response* *[retsstatus]* *bodyendline]*
body-start-line	::= **<RETS** *1*SP* {{**ReplyCode=** *quoted-reply-code 1*SP* **ReplyText=** *quoted-string*SP***>**
response	::= {*key-value-body* | *data*}
key-value-body	::= **<RETS-RESPONSE>***CRLF* **(key= value CRLF) </RETS-RESPONSE>**
rets-status	::= **<RETS-STATUS** [*1*SP* **ReplyCode=***quotedendreplycode* *1*SP* **ReplyText=***quotedstring* *SP]**/>**

The *rets-status* MAY be included in the response if the ReplyCode or ReplyText given in the *body-start-line*{} becomes invalid during the creation of the response. If the server includes a *rets-status* in its reply, the client MUST use the ReplyCode and ReplyText from the rets-status rather than from the body-start-line.

body-end-line	::= **</RETS>**

If a *body-start-line* is returned in the response then the *body-end-line* MUST also be returned.

quoted-reply-code	::="*1*5DIGITS*"

The reply-code is included to provide a mechanism to pass additional information to the client in the event that the request is processed OK (Status-Code = 200) but some condition still exist that may require an action by the client. A value of '0' indicates success. Applicable reply-codes can be found under specific transactions.

quoted-end-reply-code	::= "*1*5DIGITS*"

The *end-reply-code* is included to provide a mechanism to pass additional information to the client in the event that the request being processed by the server errors before the request has been completed. This allows the server to start streaming out data before it has completed processing the request. A value of **0** indicates success, however the server SHOULD only send an *end-reply-code* if there is an error.
The valid <*key*>, <*value*> and <*data*> elements are defined in the Response Arguments section for each transaction.

NOTE

> RETS 1.7 requires all server responses to be well-formed XML, In addition, this specification requires that clients parse RETS responses as XML, not as simple text streams. The response formats shown here are normative with respect to content, but not normative with respect to form. That is, servers are free to produce response XML in any format that complies with the W3C XML 1.0 recommendation. XML escaping of content is implied, as is XML processing of line endings and whitespace. See the W3C *XML Recommendation 1.0, Third Edition*, for full information on XML.

An example server-reply where the reply body consists of key-value pairs:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Sun, 20 Mar 2005 12:03:38 GMT
Content-Type: text/xml
Cache-Control: private
RETS-Version: RETS/1.7
<RETS ReplyCode="0" ReplyText="SUCCESS">
<RETS-RESPONSE>
Key1=Value1
Key2=Value2
</RETS-RESPONSE>
</RETS>
```

3.6 Required Server Response Header Fields

The HTTP header of any messages sent from the server MUST contain the following header fields:

`Date`	The server MUST send the date using the format defined in RFC 1123. This is a standard HTTP header field as defined in RFC 2616.

Example: `Date: Sun, 20 Mar 2005 12:03:38 GMT`

The date/time stamp MUST be represented in Greenwich Mean Time (GMT), without exception.

`Cache-Control`	The RFC 2616 standard general-header field is used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. This field SHOULD be set to "private" for all transaction in this specification.

Example: `Cache-Control: private`

`Content-Type`	This is a standard HTTP header field as defined in RFC 2616. It specifies the media type of the underlying data. The server MUST return this field in all replies. For most replies this will be set to "**`text/xml`**". See Section 5.5 in the GetObject Transaction for exceptions and more information on this field.

Example: `Content-Type: text/xml`

`RETS-Version`	The server MUST send the RETS-Version. The convention used is a "<major>.<minor>" numbering scheme similar to the HTTP Version in Section 3.1 of RFC 2616. The version of a RETS message is indicated by a RETS-Version field in header of the message.
`RETS-Version`	`::=` **`RETS-Version:`** *`version-info`*
`version-info`	`::=` **`RETS/`** `1*`*`DIGIT`* **`.`** `1*`*`DIGIT`*

Example: `RETS-Version: RETS/1.7`

Applications sending request or response messages, as defined by this specification, MUST include a `RETS-Version` of "RETS/1.7". Use of this version number indicates that the sending application is compliant with this specification.

3.7 Optional Server Response Header Fields

`Content-Length`	The Content-Length entity-header field indicates the size of the message-body, in decimal number of octets. This is a standard header field defined in RFC 2616 and is required for all requests containing a message-body not using Chunked transfer encoding.
`Transfer-Encoding`	The Transfer-Encoding entity-header field when set to the Chunked value, indicates the size of the message-body is in the chunk stream. This is a standard header field defined in RFC 2616 and is required for all responses with a body not using `Content-Length` or a `Content-Type: Multipart` response.
`Content-Encoding`	The Content Encoding entity-header field MAY be returned by the server if the client has included an AcceptEncoding header in its request () indicating that it can accept one or more compression types supported by the server. It is recommended that servers accept at least **`application/gzip`** (see 3.8, "Data Compression in RETS Transactions ").
`Content-Encoding`	`::=` *`1*64ALPHANUM / 1*64ALPHANUM`*
`RETS-Request-ID`	The contents of the `RETS-Request-ID` header, if any, sent by the client in the request. If a `RETS-Request-ID` is included in a request from the client then the server MUST return it in the response.

RETS-Request-ID	*::= 1*64ALPHNUM*
Server	The server standard response-header field contains information about the software used to handle the request. The format of this field specified in RFC 2616 Section 3.8.

Example: Server: Microsoft-IIS/4.0

RETS-Server	The RETS server vendor and server-controlled version number. This is not necessarily the same as the Server response-header field; it will be different if the HTTP server is separate from the RETS server. The format of this field is specified in RFC 2616 Section 3.8.

Example: RETS-Server: AcmeRETS/1.0

Set-Cookie	The server MAY use HTTP cookies to maintain state information. See RFC 2109 for the format of the Set-Cookie header. A cookie having a name of RETS-Session-ID defines the RETS session ID, which is used in calculating the RETS User-Agent Authentication (section 3.10). Cookies with other names have no special meaning in RETS but MAY be used when necessary.

In addition to the header fields listed here, the server may send any header compliant with HTTP 1.1.

3.8 Data Compression in RETS Transactions

Clients and servers may choose to support data compression in data returned from the server. To indicate its willingness to accept compressed data, a client includes an AcceptEncoding header in its request. If the server supports one of the compression methods accepted by the client, it can include a Content-Encoding header in its response indicating the compression method it has chose.

Clients and servers choosing to implement compression SHOULD at least support GZip compression. This method is implemented by freely-available source code in a number of languages, as well as in several proprietary software development environments. A second freely-available alternative is BZIP. Clients and servers are free to choose other encoding methods as well.

3.9 General Status Codes

Any of the following status codes (in addition to the others provided in RFC 2616) may be returned by a server in response to any request:

Table 3-1 General Status Codes

Status	Meaning
200	Operation successful.
400	Bad Request The request could not be understood by the server due to malformed syntax.
401	Not Authorized Either the header did not contain an acceptable Authorization or the username/password was invalid. The server response MUST include a WWW-Authenticate header field.

402	Payment Required The requested transaction requires a payment which could not be authorized.
403	Forbidden The server understood the request, but is refusing to fulfill it.
404	Not Found The server has not found anything matching the Request-URI.
405	Method Not Allowed The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
406	Not Acceptable The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.
408	Request Timeout The client did not produce a request within the time that the server was prepared to wait.
411	Length Required The server refuses to accept the request without a defined Content-Length.
412	Precondition Failed Transaction not permitted at this point in the session
413	Request Entity Too Large The server is refusing to process a request because the request entity is larger than the server is willing or able to process.
414	Request-URI Too Long The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This error usually only occurs for a GET method.
500	Internal server error. The server encountered an unexpected condition which prevented it from fulfilling the request.
501	Not Implemented The server does not support the functionality required to fulfill the request.
503	Service Unavailable The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
505	HTTP Version Not Supported The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

HTTP error status returns are only to be used for system level, transport syntax, and invalid transaction errors. RETS error status codes are used to indicate errors in the request arguments or the transaction processing.

3.10 Computation of the RETS User Agent Authorization value

The RETS User Agent Authorization digest response value is used in the RETS-UA-Authorization header specified in section

3.4 . It is computed as follows:

a1	::= MD5(*product* : *UserAgent-Password*)
ua-digest-response	::= HEX(MD5(HEX(a1): *RETS-Request-ID* : *session-id* : *version-info*))

where:

product	The first *product* value taken from the User-Agent header (section 3.3). Note that the *product* value consists of both the product token and version.
UserAgent-Password	::= *TOKEN* This value is a secret shared between the client and server.
RETS-Request-ID	::= *RETS-Request-ID* This value MUST be the same as that sent with the RETS-Request-ID header. If the client does not use the RETS-Request-ID header, this token is empty in the calculation.
session-id	::= If the server has sent a Set-Cookie header with a cookie name of RETS-Session-ID, session-id is the value of that cookie. If the server has not sent a cookie with that name, or if the cookie by that name has expired, this token is empty in the calculation.
version-info	::= The value of the RETS-Version header sent by the client with this transaction.

Each individual value in the concatenated string is included with whitespace removed from the beginning and end of that element, that is, there is no whitespace on either side of the delimiting colon characters.

The method of performing the MD5 calculation is given in RFC 1321.

SECTION 4 - Login Transaction

A client MUST issue a login request prior to proceeding with any other request. The Login transaction verifies all login information provided by the user and begins a RETS session. Subsequent session control may be mediated by HTTP cookies or any other method, though clients are required to support at least session control via HTTP cookies. Section 14 describes the session protocol in detail.

The server's response to the Login transaction contains the information necessary for a client to issue other requests. It includes URLs that may be used for other RETS requests, and may also contain identity and parameter information if required by the functions supported by the server.

- 4.1 Security
- 4.2 Authorization Example
- 4.3 Required Request Arguments
- 4.4 Optional Request Arguments
- 4.5 Optional Response Header Fields
- 4.6 Login Response Body Format
- 4.7 Required Response Arguments
- 4.8 Optional Response Arguments
- 4.9 Well-Known Names
- 4.10 Capability URL List
- 4.11 Reply Codes

4.1 Security

4.1.1 User Authentication

While this specification does not require the use of security — it is permissible, for example, to operate a publicly-accessible RETS server — most operators of RETS servers will wish to authenticate users. A server that requires that users be

authenticated MAY implement RFC 2617, HTTP Authentication. The use of at least digest authentication is strongly recommended.

4.1.2 Client Authentication

Client authentication may be performed through the use of the optional RETS-UA-Authorization header (section 3.4). Prior versions of this specification used a specially-calculated cnonce value in the Authorization header to implement this function. A server implementing this version of the RETS specification MUST accept the RETS-UA-Authorization header for client authentication. It MAY accept RFC 2617-style authentication as in prior versions of the RETS specification.

4.1.3 Data Security

Needs for secure HTTP transactions cannot be met by authentication schemes. For those needs, HTTP-over-TLS (commonly known as HTTPS) is a more appropriate protocol. A compliant server MAY support only HTTP-over-SSL. In this case, the server SHOULD listen on port 12109 rather than the standard RETS port, 6103.

4.2 Authorization Example

The following example assumes that a client application is trying to access the Login URI on the server using the POST method, and without using client authentication. The URI is `"http://www.example.com/login"`. Both client and server know that the username is "joesmith", and the password is "SuperAgent". The example also assumes the use of authentication using RFC 2617.

The first time the client requests the document, no Authorization header is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
```

```
WWW-Authenticate: Digest realm="Users@example.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c0"
opaque="5ccdef346870ab04ddfe0412367fccba"
```

The client may prompt the user for the username and password, after which it will respond with a new request, including the following Authorization header:

```
Authorization: Digest username="joesmith",
realm="Users@example.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c0",
opaque="5ccdef346870ab04ddfe0412367fccba",
uri="/login",
response="13258d9b0bc217c9502b47e32dff8ee9"
```

4.3 Required Request Arguments

There are no required request arguments.

4.4 Optional Request Arguments

4.4.1 BrokerCode Argument

brokerCodeArgument	::= **BrokerCode =** *broker-code* [, *broker-branch*]

Some servers may support the scenario where a user belongs to multiple brokerages. If this is the case then the broker information (broker-code and broker-branch) must be input during login. If they are not included then the list of broker codes/branches is passed back to the client application through the response along with a "20012 Broker Code Required" reply-code.

broker-code	::=*1*24ALPHANUM*

broker-branch	::=*1*24ALPHANUM*

4.4.2 SavedMetadataTimestamp Argument

savedMetadataTimestamp	::=**SavedMetadataTimestamp =** *saved-timestamp*

The client MAY inform the server of the timestamp associated with the version of metadata that it has currently saved. The server MAY use this to adapt to an earlier version of metadata than it chooses to advertise, or simply log the value to note out-of-date client metadata, or ignore the value entirely. In particular, the server is not required to alter its behavior in any way based on the value of this argument.

saved-timetamp	::= *DATE*

4.5 Optional Response Header Fields

There are no additional optional response header fields.

4.6 Login Response Body Format

The body of the login response has three basic formats when replying to a request. The simplest form is when there is an error:

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP />
```

The second case is where the user belongs to more than one broker and they have not provided broker information as part of the login. The reply contains a list of all brokerages the user belongs to.

```
<RETS ReplyCode = "20012" 1*SP ReplyText = quoted-string *SP >
<RETS-RESPONSE>CRLF
2*( Broker = broker-code [, broker-branch ] CRLF )
</RETS-RESPONSE></RETS>
```

The third case is the normal "OK" response. In this case several arguments are passed back to the client in the response.

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP >
<RETS-RESPONSE>
member-name-key
user-info-key
broker-key
metadata-ver-key
metadata-timestamp-key
min-metadata-timestamp-key
[office-list-key ]
[balance-key ]
[timeout-key ]
[pwd-expire-key ]
capability-url-list
</RETS-RESPONSE>
[<RETS-STATUS [1*SP ReplyCode= quoted-end-reply-code 1*SP }}{{{}ReplyText=
quoted-string *SP]/>
</RETS> CRLF
```

4.7 Required Response Arguments

4.7.1 Broker

broker-key	::= **Broker =** *broker-code* [**,** *broker-branch*] *CRLF*

Broker information for the logged in user is returned to the client.

broker-code	::= *1*24ALPHANUM*
broker-branch	::= *1*24ALPHANUM*

These parameters are used in the validation routines of the Update transaction (see Section 10 for more information).

4.7.2 Member Name

member-name-key	::= **MemberName =** *member-name CRLF*

The member's full name (display name) as it is to appear on any printed output, for example "Jane T. Row".

member-name	::= *1*48TEXT*

4.7.3 Metadata Version Information

The metadata version and timestamp keys indicate the current and minimum-acceptable versions of metadata.

metadata-ver-key{}	::= **MetadataVersion =** *metadata-version CRLF*

This is the most current version of the metadata that is available on the server.

metadata-version	::=*1*2DIGITS* **.** *1*2DIGITS* [**.** *1*5DIGITS*]

It uses a "<major>.<minor>.<release>" numbering scheme. The version is advisory and is not used by the metadata currency scheme.

metadata-timestamp-key	::= **MetadataTimestamp =** *DATE CRLF*

This is the timestamp associated with the current version of metadata on the host. If the client has cached an earlier version of metadata, it SHOULD take whatever action is necessary to load the current version of metadata.

min-metadata-timestamp-key	::= **MinMetadataTimestamp =** *DATE CRLF*

This is the earliest version of the metadata that the host will support. If the version of the metadata being used by the client has a timestamp earlier than this time the client SHOULD retrieve the newer metadata from the host. In any case, the client MUST NOT send transactions using metadata older than `MinMetadataTimestamp`.

The definition of the minimum version of the metadata is to permit clients to ignore non-essential changes to components such as help text and user-readable descriptions.

4.7.4 User information

user-info-key	::= **User =** *user-id* **,** *user-level* **,** *user-class* **,** *agent-code CRLF*

This key contains basic information about the user that is stored on the server. If a server does not support one of these fields then it MUST set the returned value to empty (a zero-length string).

user-id	::= *1*30ALPHANUM*

user-class	::= *1*30ALPHANUM*
user-level	::= *1*5DIGIT*
agent-code	::= *1*30ALPHANUM*

The agent-code is the code that is stored in the property records for the listing agent, selling agent, etc. In some implementations this may be the same as the user-id. The fields user-class and user-level are implementation dependent and may not exist on some systems, in which case, an empty string should be returned. These parameters are used in the validation routines of the Update transaction (see Section 10 for more information).

4.7.5 Capability URL List

capability-url-list	::= see Section 4.10 for format information

The server MUST return a capability list that includes at least Search, Login and GetMetadata. The server MAY in addition return any of the other types in Section 4.10 . If the server supports any of the additional functions (and the client is entitled to access the function by virtue of the supplied login information), it MUST provide URLs for those functions. The server MAY supply URLs in addition to those in Section 4.10 based on the user-agent. If it does, it MUST follow the format specified in Section 4.10.

4.8 Optional Response Arguments

4.8.1 Accounting Information

balance-key	::= **Balance =** *balance CRLF*

If the server supports an active billing account then this value SHOULD represent a user-readable indication of the money balance in the account.

balance	::= *1*32ALPHANUM*

4.8.2 Access Control Information

timeout-key	::= **TimeoutSeconds =** *1*5DIGIT CRLF*

The number of seconds after a transaction that a session will remain alive, after which the server will terminate the session automatically (e.g. invalidate the session-id). This is commonly referred to as the inactivity timeout. A server need not provide this capability; however, if it does use session timeouts in order to prevent monopolization of resources, it MUST inform the client of the timeout interval by returning this response field.

pwd-expire-key	::= **Expr =** *pwd-expr ,expr-warn-per CRLF*

Indicates when a users password will expire. The parameter pwd-expr is a date in RFC 1123 format. And expr-warn-per is the number of days (*1*3DIGIT*) prior to expiration that the user should be warned of the upcoming password expiration. A expr-warn-per value of (-1) indicates that the password expiration is disabled.

4.8.3 Office List Information

office-list-key	::= **OfficeList =** *broker-code* [**;** *broker-branch*] *(**,** *broker-code* [**;** *broker-branch*]) *CRLF*

If the logged in user is a company owner or manager they may have rights to login to multiple offices. The *office-list-key* is an enumeration of the offices to which the server will permit login.

broker-code	::=*1*24ALPHANUM*

broker-branch	*::=1*24ALPHANUM*

4.9 Well-Known Names

Some fields returned from the login are considered "Well-Known" and are used in the validation routines of the Update transaction. Those fields are as follows:

Table 4-1 Well-Known Names for Input Fields

Well-Known name	Input Return Field
USERID.	user-id
.USERCLASS.	user-class
.USERLEVEL.	user-level
.AGENTCODE.	agent-code
.BROKERCODE.	broker-code
.BROKERBRANCH.	broker-branch

The client MUST assume a blank value for any well-known name for which the server does not supply an input field. These values are used in Table 11-34, "Validation Expression Special Operand Tokens ".

4.10 Capability URL List

The capability-url-list is the set of functions or URLs to which the login grants access. A capability consists of a key and a URL. The list returned from the server in the login reply has the following format:

```
[Action = action-URL CRLF]
[ChangePassword = change-password-URL CRLF]
[GetObject = get-object-URL CRLF]
Login = login-URL CRLF
[LoginComplete = login-complete-URL CRLF]
[Logout = logout-URL CRLF]
Search = search-URL CRLF
GetMetadata = get-metadata-URL CRLF
[ServerInformation = server-information-URL CRLF]
[Update = update-URL CRLF]
```

Table 4-2 Capability URL Descriptions

Parameter	Purpose
action-URL	A URL on which the client MUST perform a GET immediately after login. This might include a bulletin or the notification of email. The client application SHOULD provide a means for the user to view the retrieved document. A server is not required to supply an Action URL.
change-password-URL	A URL for the ChangePassword transaction.
get-metadata-URL	A URL for the Get Metadata transaction.
get-object-URL	A URL for the Get Object transaction.

login-URL	A URL for the Login Transaction. The client software should use this URL the next time it performs a Login. If this URL is different from the one currently stored by the client the client, MUST update the stored one to the new one. This provides a mechanism to move the Login server.
login-complete-URL	RESERVED
logout-URL	A URL for the Logout transaction.
search-UR	A URL for the Search transaction.
update-URL	A URL for the Update transaction.
server-information-URL	A URL for the System Information transaction

The URLs in the capability-url-list may be specified in any order. In addition, the table is extensible; servers may define additional transactions for clients to access. If a transaction is offered only to particular user agents, the keys for those additional transactions MUST begin with the user-agent token, followed by a dash "-", followed by an implementation-defined function name.

additional-transaction	`:: =` *user-agent-token* `-` *function-name CRLF*
user-agent-token	`::= token portion of the user-agent` (Section 3.3)
function-name	`::=` *1*ALPHA*

Example: `MLSWindows-special = /special_function`

A compliant client need not recognize any transaction that is not included in this specification. If some extended transactions are offered to any user-agent, the keys for those transactions MUST begin with an "X" followed by a dash, followed by an implementation-defined function name. Server implementors who implement potentially-unrestricted extension transactions are urged to register their keys and service descriptions on the RETS web site to encourage wider adoption.

URLs other than the Login URL may be relative URLs. The Login URL MUST be an absolute URL. If a URL is not absolute, the client application should canonicalize it according to the rules in RFC 2396, section 5. The "base URL" (as defined in RFC 2396, section 5.1.1) for this operation is the URL used for the *current* login transaction, not the new Login URL.

URLs MUST be URL-encoded per RFC 2396.

4.11 Reply Codes

Table 4-3 Valid Reply Codes for Login Transaction

Reply Code	Meaning
0	Operation successful
20003	Zero Balance The user has zero balance left in their account.
20004 thru 20011	RESERVED
20012	Broker Code Required The user belongs to multiple broker codes and one must be supplied as part of the login. The broker list is sent back to the client as part of the login response (see section 4.6).
20013	Broker Code Invalid The Broker Code sent by the client is not valid or not valid for the user
20014 thru 20019	RESERVED

20022	Additional login not permitted There is already a user logged in with this user name, and this server does not permit multiple logins.
20036	Miscellaneous server login error The quoted-string of the body-start-line contains text that SHOULD be displayed to the user
20037	Client authentication failed. The server requires the use of a client password (section 4.1.2), and the client either did not supply the correct client password or did not properly compute its challenge response value.
20050	Server Temporarily Disabled The server is temporarily offline. The user should try again later

NOTE: RETS does not require that a server maintain user accounts.

SECTION 5 - GetObject Transaction

The GetObject transaction is used to retrieve structured information related to known system entities. It can be used to retrieve multimedia files and other key-related information. Objects requested and returned from this transaction are requested and returned as MIME media types. The message body for successful retrievals contains only the objects in the specified MIME media type. Error responses follow the normal response format (section 3.9).

- 5.1 Required Client Request Header Fields
- 5.2 Optional Client Request Header Fields
- 5.3 Required Request Arguments
- 5.4 Optional Request Arguments
- 5.5 Required Server Response Header Fields
- 5.6 Optional Server Response Header Fields
- 5.7 Required Response Arguments
- 5.8 Optional Response Arguments
- 5.9 Metadata
- 5.10 Resources
- 5.11 Multipart Responses
- 5.12 Reply Codes

5.1 Required Client Request Header Fields

In addition to the Required Client Request Header Fields specified in Section 3.3 , the header of any messages sent from the client MUST contain the following header fields:

`Accept`	The client MUST request a media type using the standard HTTP Accept header field. Media-type formats (subtypes) are registered with the Internet Assigned Number Authority (IANA) and use a format outlined in RFC 2045 [8]. When submitting a request the client MUST specify the desired type and format. If the server is unable to provide the desired format it SHOULD return a "406 Not Acceptable" status. However, if there are no objects of any subtype available for the requested object the server SHOULD return "404 Not Found." The format of the Accept field is as follows:

Accept	::= **Accept:** *type* / *subtype* [; *parameter*] *(, *SP type* / *subtype* [; *parameter*])
type	::= * | <a publicly-defined type>
ubtype	::= * | <A publicly-defined extension token that has been registered with IANA>

parameter	::= **q =** < qvalue scale from 0 to 1 >

A complete list of media types and subtypes is available at:

http://www.iana.org/assignments/media-types/

The qvalue is used to specify the desirability of a given media type/subtype, with "1" being the most desirable, "0" being the least desirable, and a range in between. The default qvalue is "1".

Example: `Accept: image/jpeg, image/tiff;q=0.5, image/gif;q=0.1`

Verbally, this would be interpreted as "image/jpeg is the preferred media type, but if that does not exist, then send the image/tiff entity, and if that does not exist, send the image/gif entity."

The types supported by the server are defined in the Metadata Dictionary as defined in section 11.4.1 .

5.2 Optional Client Request Header Fields

The GetObject transaction has no optional request header fields.

5.3 Required Request Arguments

`Resource`	A resource defined in the metadata dictionary (see Section 11.2.2)

The resource from which the object should be retrieved is specified by this entry. For more information see 5.9. The resource MUST be a resource defined in the metadata (section 11.4.1).

`Type`	The object type as defined in the metadata (see Section 11.4.1)

The grouping category to which the object belongs. Type MUST be an ObjectType defined in the Object metadata for this Resource. For more information see section 11.4.1 .

`ID`	A string identifying the object or objects being requested:

ID	::= *resource-set* *(, *resource-set*)
resource-set	::= *resource-entity* [: *object-id-list*]
resource-entity	::= *1*ALPHANUM*
object-id-list	::= * | *object-id* *(: *object-id*)
object-id	::= *1*DIGIT*

For objects, the *resource-entity* is a value (e.g., MLS number, AgentID) from the KeyField of the Resource for which the object is to be retrieved.

The `object-id` is the particular object to be retrieved. Objects are assumed to be stored sequentially on the host beginning with an `object-id` of "1". If the object-id is 0 (zero or not provided), the designated preferred object of the given type is returned. If the object-id is set to "*" then all objects corresponding to the `resource-entity` are returned. This parameter can be used to specify the photo number, e.g. a value of "3" would indicate photo number 3.

If multiple `resource-entity` or `object-id` values are sent, or if any object-id-list is "*", then the host MUST respond with a multipart MIME [8] response. See 5.11, "Multipart Responses" , for more detail.

5.4 Optional Request Arguments

5.4.1 Location

`Location`	**`0`** | **`1`**

This parameter indicates whether the object or a URL to the object should be returned. This is used to provide access to the semi-permanent storage location of information for access outside of the transaction (e.g. for use in email to a customer). The lifetime of this semi-permanent storage is not defined by this specification.

If Location is set to "1" the server MAY return a URL to the given object. The default is "0". The server MAY support this functionality (`Location`="1") but MUST support `Location`="0". In other words, some servers may store the objects in a database or generate them dynamically. Therefore, it may not be possible for those servers to return a URL to the requested object. In these cases the server MAY choose not to support `Location`="1". However, all servers MUST support a method to get the object and therefore, MUST support the case where `Location`="0".

5.5 Required Server Response Header Fields

In addition to the other Required Server Header Fields specified in Section 3.6 the following response header fields are required.

Content-Type	The media type of the underlying data. The server MUST return this field in all replies. Additionally, this field MUST be returned as part of the header for each body part. This field MUST be set to the type of media returned. See Section 5.1 for more information on <*`type`*> and <*`subtype`*>.

`Content-Type`	`::=` **`Content-Type:`** *`type /subtype`*

Example: `Content-Type: image/jpeg`

If the client has requested multiple IDs, the server MUST return a multipart message. If it does, it MUST return a Content-Type of "multipart/parallel" along with a boundary delimiter in the response header. See Section 5.11 for more information on multipart responses

Example: `Content-Type: multipart/parallel; boundary=AAABBBCCC`

Content-ID	An `ID` for the object. This field MUST be returned as part of the header for each body part in a multipart response.

`Content-ID`	`::=` **`Content-ID:`** *`*64<TEXT, EXCLUDING CR/LF>`*

Example: `Content-ID: 123456`

Object-ID	The object number being returned. This field MUST be returned as part of the header for each body part in a multipart response. *`Object-ID`* `::=` **`Object-ID:`** *`1*5DIGIT`*

Example: `Object-ID: 2`

MIME-Version	All responses MUST include a MIME-Version of "1.0" in the response header.

Example: `MIME-Version: 1.0`

5.6 Optional Server Response Header Fields

In addition to the other Optional Server Header Fields specified in Section 3.7 the following response header fields are also optional.

5.6.1 Location

Location	If the client has submitted a request with "`Location=1`" the header of the response MUST contain the Location header field. If the server does not support this functionality then "`Location:`" without a URI MUST be returned.

`Location`	`::=` **`Location:`** *`URL`*

Example: `Location: http://www.example.com/pic/123456.jpg`

If the server is returning a multipart response, then this header MUST be included in the MIME part headers for each object successfully requested.

5.6.2 Description

Description	A text description of the object.

`Description`	`::=` **`Content-Description:`** *`*64<TEXT, EXCLUDING CR/LF>`*

Example: `Content-Description: Front View`

If the server is returning a multipart response, then this header should be included in the MIME part headers for each object successfully requested.

5.7 Required Response Arguments

There are no required response arguments.

5.8 Optional Response Arguments

There are no optional response arguments.

5.9 Metadata

To retrieve objects the client MAY first retrieve the metadata that describes the Resources and Objects that are available with the GetMetadata transaction described in section 12. A full description of the Metadata Dictionary is provided in Section 11 .

5.10 Resources

RETS does not require that any particular type of object be made available by a server. However, a server MUST use a standard well-known name under which to make its data available if a suitable well-known name is defined in the standard.

5.11 Multipart Responses

In the case where the client has requested multiple IDs, the server MUST return a multipart response. In the case of multipart responses, in which one or more different sets of data are combined in a single body, a "multipart" media type field must appear in the entity's header.

5.11.1 General Construction

RFC 2045 describes the format of an Internet message body containing a MIME message. The body contains one or more body parts, each preceded by a boundary delimiter line, and the last one followed by a closing boundary delimiter line. After its boundary delimiter line, each body part then consists of a header area, a blank line, and a body area.

Example:

```
HTTP/1.1 200 OK
Server: Apache/2.0.13
Date: Fri, 22 OCT 2004 12:03:38 GMT
Cache-Control: private
```

```
RETS-Version: RETS/1.0
MIME-Version: 1.0
Content-type: multipart/parallel; boundary="simple boundary"

--simple boundary
Content-Type: image/jpeg
Content-ID: 123456
Object-ID: 1

<binary data>
--simple boundary
Content-Type: image/jpeg
Content-ID: 123457
Object-ID: 1

<binary data>
--simple boundary --
```

5.11.2 Error Handling

When a client requests multiple objects in a single transaction, one or more of those objects may be unavailable. In this case, the server communicates the failure by including a RETS return message in place of the unavailable object. In this case, the Content-Type will be text/xml, and the content will be a RETS response:

Example:

```
HTTP/1.1 200 OK
Server: Apache/2.0.13
Date: Fri, 22 OCT 2004 12:03:38 GMT
Cache-Control: private
RETS-Version: RETS/1.7
MIME-Version: 1.0

Content-type: multipart/parallel; boundary="simple boundary"{}

--simple boundary
Content-Type: image/jpeg
Content-ID: 123456
Object-ID: 1

<binary data>
--simple boundary
Content-Type: text/xml
Content-ID: 123457
Object-ID: 1

<RETS ReplyCode=”20403” ReplyText=”There is no listing with that ListingID”/>

--simple boundary --
```

If the server is supplying an error message for a wild-card object request (`Object-ID` of *), the `Object-ID` for the error part SHOULD be * as well.

5.12 Reply Codes

Table 5-1 GetObject Reply Codes

Reply Code	Meaning
20400	Invalid Resource The request could not be understood due to an unknown resource.

20401	Invalid Type The request could not be understood due to an unknown object type for the resource.
20402	Invalid Identifier The identifier does not match the KeyField of any data in the resource.
20403	No Object Found No matching object was found to satisfy the request.
20406	Unsupported MIME type The server cannot return the object in any of the requested MIME types.
20407	Unauthorized Retrieval The object could not be retrieved because it requests an object to which the supplied login does not grant access.
20408	Resource Unavailable The requested resource is currently unavailable.
20409	Object Unavailable The requested object is currently unavailable.
20410	Request Too Large No further objects will be retrieved because a system limit was exceeded.
20411	Timeout The request timed out while executing
20412	Too many outstanding requests The user has too many outstanding requests and new requests will not be accepted at this time.
20413	Miscellaneous error The server encountered an internal error.

SECTION 6 - Logout Transaction

The Logout transaction terminates a session. Except in cases where connection failure prevents it or the user has requested an immediate shutdown of the client, the client SHOULD send the Logout transaction. If the client sends a Logout transaction, the server MUST attempt to send a response before terminating the session.

The server MAY send accounting information back to the client in the response to this transaction. The client is not required to display or otherwise process the accounting information.

- 6.1 Required Request Arguments
- 6.2 Optional Request Arguments
- 6.3 Required Response Arguments
- 6.4 Optional Response Arguments
- 6.5 Logout Response Body Format
- 6.6 Reply Codes

6.1 Required Request Arguments

There are no required request arguments.

6.2 Optional Request Arguments

There are no optional request arguments.

6.3 Required Response Arguments

There are no required response arguments.

6.4 Optional Response Arguments

ConnectTime	The amount of time that the client spent connected to the server, specified in seconds.

connect-time	::= **ConnectTime=***1*9DIGITS CRLF*

Billing	If the server supports an active billing account, this is total amount billed for this session, specified as TEXT which is implementation-defined

billing	::= **Billing=****<TEXT, EXCLUDING CR/LF> CRLF*

SignOffMessage	Any text. The client MAY display this message, if the server includes it in the response. Servers should not expect, however, that users would read or see the message, since communication failure may make it impossible for the client to receive the Logoff response.

sign-off-message	::= **SignOffMessage=****<TEXT, EXCLUDING CR/LF> CRLF*

6.5 Logout Response Body Format

The Logout response body is a key/value response (see section 3.5, "Response Format").

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP >
[<RETS-RESPONSE>
[connect-time]
[billing]
[sign-off-message]
</RETS-RESPONSE>] [<RETS-STATUS [1*SP ReplyCode= quoted-end-reply-code 1*SP ReplyText=quot
ed-string *SP]/>]
</RETS>
```

6.6 Reply Codes

Table 6-1 Logout Reply Code

Reply Code	Meaning
0	Operation successful
20701	Not logged in The server did not detect an active login for the session in which the Logout transaction was submitted.

20702	Miscellaneous error. The transaction could not be completed. The ReplyText gives additional information.

SECTION 7 - Search Transaction

The Search transaction requests that the server search one or more searchable databases and return the list of qualifying records. The body of the response contains the records matching the query, presented in the requested format. The data can be returned in one of three formats: COMPACT, COMPACT-DECODED or STANDARD-XML.

- 7.1 Search Types
- 7.2 Search Terminology
- 7.3 Required Request Arguments
- 7.4 Optional Request Arguments
- 7.5 Required Response Arguments
- 7.6 Search Response Body Format
- 7.7 Query language
- 7.8 Reply Codes

7.1 Search Types

The server MUST support at least one type of search. The types of searches supported by the server are specified in the metadata. Each of these searches may by conducted against different databases or tables depending on the server implementation.

Some resources are specified by well-known names. If a server implementation supports searches of any of these resources, it SHOULD use the well-known name to describe that search. The list of well-known resource names is provided in Table 11-1, "Well-Known Resource Names" ; well-known classes for those resources are given in Table 11-7, "Metadata Content: Resource Class".

StandardNames for classes are given in Table 11-7, "Metadata Content: Resource Class" .

NOTE: RETS does not require that a server support any particular search. The user or maintainer of a server is responsible for deciding which resources should be made searchable.

7.2 Search Terminology

7.2.1 Field Delimiter

A server may designate a particular OCTET to be used as a delimiter for separating entries in both the COLUMNS list and the DATA returned using the COMPACT and COMPACT-DECODED formats. The octet should be chosen to avoid the need to escape data within a record

field-delimiter	::= *HEX HEX*

7.2.2 Field Name

A field is the keyword or code that the server uses to identify a particular column in the database table. Each field may be either a System-Name, as defined in the metadata, or a Standard-Name, as defined in the Real Estate Transaction XML DTD. The server MUST accept either set of names interchangeably.

7.2.3 Record Count

This value indicates the number of records on the server matching the search criteria sent in the search query.

record-count	::= *1*9DIGITS*

Note that this value may be greater than the number of records returned, if the server has limited the size of the return for any reason.

7.2.4 Other terms

XML-data-record	::= <A data record as defined by the RETS Data XML DTD>.

7.3 Required Request Arguments

7.3.1 Search Type and Class

The SearchType and Class arguments specify the data that the server is to search.

SearchType	::= *ResourceID*

The type of search to perform as discussed in Section 7.1 and defined in the Metadata (see section 11.2.2).

Class	:: = *1*32ALPHANUM*

This parameter is set to a value that represents the class of data within the SearchType, taken from the Class metadata (section 11.3.1). If the resource represented by the `SearchType` has no classes, the `Class` parameter will be ignored by the server and MAY be omitted by the client. If the client does include the `Class` parameter for a classless search, the value SHOULD be the same as the *ResourceID* in order to insure forward compatibility.

Note that if `StandardNames` (Section 7.4.7) is set to 1, then both the SearchType and Class are specified using StandardNames.

7.3.2 Query Specification

The specification consists of the query itself together with a designation of the query language.

Query	::= <The query to be executed by the server>

The query is specified in the language described in Section 7.7 .

QueryType	::= **DMQL2**

An enumeration giving the language in which the query is presented. The only valid value for RETS 1.7 is "DMQL2" which indicates the query language described in Section 7.7

7.4 Optional Request Arguments

7.4.1 Count

The Count argument controls whether the server's response includes a count.

Count	::= **0** | **1** | **2**

If this argument is set to one ("1"), then a record-count is returned in the response in addition to the data. Note that on some servers this will cause the search to take longer since the count must be returned before any records are received. If this entry is set to two ("2") then only a record-count is returned; no data is returned, but all matches are counted regardless of any `Offset` or `Limit` parameter. If the `Count` argument is not present or set to zero ("0") there is no record count returned.

Example: `Count=2`

Instructs the server to return only a count of the records matching the query.

7.4.2 Format

The Format argument selects one of the three supported data return formats for the query response.

Format	::= **COMPACT** | **COMPACT-DECODED** | **STANDARD-XML** | **STANDARD-XML:***dtd-version*

"COMPACT" means a field list <COLUMNS> followed by a delimited set of the data fields. "COMPACT-DECODED" is the same as COMPACT except the data is returned in a fully-decoded (people-readable) format. See Section 13 for more information on the COMPACT formats. "STANDARD-XML" means an XML presentation of the data in the format defined by the RETS Data XML DTD. Servers MUST support all formats. If the format is not specified, the server MUST return STANDARD-XML.

Example: `Format=COMPACT-DECODED`

If the client has requests STANDARD-XML, it may also append a preferred DTD version. The server SHOULD support at least the current version and the prior one.

Example: `Format=STANDARD-XML:1.0`

7.4.3 Limit

The Limit argument requests the server to apply or suspend a limit on the number of records returned in the search.

Limit	::= **NONE** | *1*9DIGIT*

If this entry is set to ("NONE") or is not present, the server SHOULD treat this as a request to suspend enforcement of the standard download limit. The use of "NONE" MAY disable both the <MAXROWS> tag and return-code "20208 Maximum Records Exceeded". Client implementors should be aware that some server implementations might not honor the request to disable the limit; the server operator's business rules take precedence over the request to waive the system download limit.

Alternatively, if the entry is set to a number greater than zero, the server MUST not return more than the specified number of records. If the server did not return all matching records then the <MAXROWS> tag MUST be sent at the end of the data stream.

7.4.4 Offset

The client may specify that a retrieval start at other than the first record in the set of records matching he query by specifying the Offset argument.

Offset	::= *1*9DIGIT*

This argument indicates to the server that it SHOULD start sending the data to the client beginning with the record number indicated, with a value of "1" indicating to start with the first record. This can be useful when requesting records in batches, however, client implementors should be aware that data on the server MAY change as they iterate through the batches and it is possible that some records may be missed or added. In other words, the server is not required to maintain a cursor to the data.

7.4.5 Select

By default, the server MUST return all fields accessible to the client. The client may select a subset of those fields by specifying the Select argument.

Select	::= *field *(, field)*

This parameter is used to set the fields that are returned by the query. If this entry is not present then all allowable fields for the search/class are returned. The server MAY return an error when there are unknown fields in the select list. The server MUST NOT return more fields than are specified in the Select argument when the client requests COMPACT or COMPACT-DECODED data. It MAY return fewer if some of the field names are invalid or if a requested field is unavailable to the user based on security or other restrictions.

7.4.6 Restricted Indicator

In some instances, the server may withhold the values of selected fields on selected records. In this case, the server SHOULD send back a null value, unless the client has specified a RestrictedIndicator.

RestrictedIndicator	*::= 1*9TOKENCHAR*

This entry indicates to the server that it should set the restriction indicator to the value specified by this tag. The default is that the server returns no restriction indicator.

Example: `RestrictedIndicator` = `####`

This would mean that all fields that the user is not allowed to see within a record (e.g. ExpirationDate) are returned with a value of ####.

Note that if the client requests fields that the server would withhold for every record, the server MAY choose to omit the field from the list returned rather than use the RestrictedIndicator for every record.

7.4.7 StandardNames

Queries may use either standard names or system names in the query (Section 7.7). If the client chooses to use standard names, it MUST indicate this using the StandardNames argument.

StandardNames	*::=* **0** | **1**

If this entry is set to ("0") or is not present the field names passed in the search are the `SystemNames`, as defined in the metadata. If this entry is set to ("1") then the StandardNames are used for the field names passed in the search. The StandardName designation applies to all names used in the query: `SearchType`, `Class`, `Query` and `Select`.

7.5 Required Response Arguments

There are no required response arguments.

7.6 Search Response Body Format

NOTE

> RETS 1.7 requires all server responses to be well-formed XML, and additionally requires search transaction responses to be valid XML. In addition, RETS requires that clients parse server responses as XML, not as simple text streams. The response formats shown here are normative with respect to content, but not normative with respect to form. That is, servers are free to produce response XML in any format that complies with the W3C XML 1.0 recommendation, so long as it is valid with respect to the appropriate DTD. So, for example, when the response format below calls out an empty XML tag, either the abbreviated tag format (**<MAXROWS/>**) or the full format (**<MAXROWS></MAXROWS>**) may be sent by the server and should be interpreted appropriately by the client. In addition, XML escaping of content is implied. See the W3C *XML Recommendation 1.0, Third Edition,* for full information on XML.

The body of the search response has the following format when replying to a request with the format set to "COMPACT" or "COMPACT-DECODED":

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP >
[count-tag ]
[delimiter-tag ]
[column-tag ]
*( compact-data )
[max-row-tag ]
[next-key-tag ]
[<RETS-STATUS [1*SP ReplyCode= quoted-end-reply-code 1*SP
ReplyText= quoted-string *SP]/>
</RETS> CRLF
```

The body of the search response has the following format when replying to a format request of "STANDARD-XML" data:

```
<?xml version="1.0" ?>
[doctype]
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP >
[ count-tag ]
( XML-data-record )
[ max-row-tag ]
[ next-key-tag ][<RETS-STATUS [1*SP ReplyCode= quoted-end-reply-code 1*SP
ReplyText= quoted-string *SP]/>]
</RETS> CRLF
```

doctype	::= **<!DOCTYPE RETS PUBLIC "-//RETS//DTD RETS DATA 1.7//EN>" "http://www.rets.org/dtd/2004/09/rets-data-1_7.dtd">**
dtd-version	::= <Name of the RETS DTD used to produce this document>

When the client requests the STANDARD-XML representation, it may also specify a DTD version. The server SHOULD be prepared to support at least the current version and the prior version. Data DTD versions are of the form

RETS-*yyyymmdd*.dtd

where *yyyymmdd* is the release date of the DTD.

compact-data	::= **<DATA>** *field-delimiter *(field-data field-delimiter*) **</DATA>**

If a "COMPACT" or "COMPACT-DECODED" format is specified in the request then a "<DATA>" tag, a delimited list of field-data and a "</DATA>" end tag are returned to the client for each record returned. The field-delimiter is determined by the delimiter-tag.

count-tag	::= **<COUNT** *1*SP* **Records="***record-count***"** *1*SP* **/>**

When the client application specifies that a count should be returned (count-type = "1" | "2") a count-tag MUST be sent by the server in the response. The "<COUNT>" tag MUST be on the first line following the reply-code line. The record-count value indicates the number of records on the server matching the search criteria sent in the search query.

column-tag	::= **<COLUMNS>** *field-delimiter* 1*(*field field-delimiter*) **</COLUMNS>**

If a "COMPACT" format is specified in the request then a "<COLUMNS>" tag, including a delimited list of the names of all the fields of data being returned, is sent back in the response. These names are the system-names unless standard-names were used in the query.

The field-delimiter is determined by the delimiter-tag.

delimiter-tag	::= **<DELIMITER value ="** *field-delimiter* **"/>**

This parameter tells the client which character (OCTET) to use as a delimiter for both the COLUMNS list and the DATA returned. The server MUST send this parameter for "COMPACT" or "COMPACT-DECODED" formats. The "<DELIMITER>" tag MUST precede column-tag.

max-row-tag	::= **<MAXROWS/>** *CRLF* **<MAXROWS></MAXROWS>**

A tag that indicates the maximum number of records allowed to be returned by the server has been exceeded, or alternatively, the Limit number passed by the client in the request has been exceeded.

7.7 Query language

The query takes the form indicated below. This is the actual search criteria passed to the server. The server parses this query and generates a server-compatible query based on the parameters passed in the query-list.

7.7.1 Query language BNF

search-condition	::= *query-clause* | (*search-condition or query-clause*)
query-clause	::= *boolean-element* | (*query-clause and boolean-element*)
boolean-element	:: = [*not*] *query-element*
query-element	::= *field-criteria*|(**(** search-condition **)**)
or	::= **OR** | **|**
and	::= **AND** | **,**
not	::= **NOT** | **~**
field-criteria	::= **(** *field* **=** *field-value* **)**
field-value	::= *lookup-list* | *string-list* | *range-list* | *period* | *number* | *string-literal* | **.EMPTY.**
lookup-list	::= *lookup-or* | *lookup-not* | *lookup-and* | **.ANY.**
lookup-or	::= **|** *lookup* *(**,***lookup*)
lookup-not	::= **~** *lookup* *(**,** *lookup*)
lookup-and	::= **+** *lookup* *(**,** *lookup*)
lookup	::= <any legal *ALPHANUM* value for the field as defined in the metadata>
string-list	::= 1*(*string* *(**,** *string*))
string	::= *string-eq* | *string-start* | *string-contains* | *string-char*
string-eq	::= *1*ALPHANUM*
string-start	::= *1*ALPHANUM* **
string-contains	::= *** *1*ALPHANUM* **
string-char	::= ***ALPHANUM 1?*** **ALPHANUM*
string-literal	::= **"** 1*(**(*PLAINTEXT* except ") *(*2* ")** *(*PLAINTEXT* except ")) **"**
range-list	::= *range* *(**,** *range*)
range	::= *between* | *greater* | *less*

between	::= (*period* | *number* | *string-eq*) **-** (*period* | *number* | *string-eq*)
greater	::= (*period* | *number* | *string-eq*) **+**
less	::= (*period* | *number* | *string-eq*) **-**
period	::= (*date* | *datetime* | *time*)
number	::= [-]1*DIGIT ["." *DIGIT]
date	::= (*year* **-** *month* **-** day) | **TODAY**
datetime	::= (*year* **-** *month* **-** *day* **T** *hour* **:** *minute* **:** *second* [**.** *fraction*]) | **NOW**
time	::= (hour ":" minute ":" second ["." fraction])
fraction	::= 1*3DIGIT
second	::= 2DIGIT
minute	::= 2DIGIT
hour	::= 2DIGIT
day	::= 2DIGIT
month	::= 2DIGIT
year	::= 4DIGIT

7.7.2 Query parameter interpretation

Query literal values are interpreted in the value space of the searched field. That is, the data type of the searched field determines the interpretation of the search literal values, which MUST be valid in that value space.
All datetimes submitted in queries MUST be in GMT. All other dates or times are interpreted in host time. The host MUST interpret the token **NOW** as the current date and time, and the token **TODAY** as the current date. If a DateTime is used in a Date context, the host MUST either reject the expression or interpret the date as a datetime with a time of 0000.

In processing a literal string, a server MAY substitute a *string-char* expression (**?**s) for the range of characters that contain any non-ALPHANUM not supported by that server.

In processing decimal numbers, where rounding is necessary, a server SHOULD round down for the bottom of ranges or values less than 0.5 and round up for the tops of ranges or values 0.5 or greater.

There are four types of field values that can be passed in the query string. They are a *lookup-list*, a *range*, a *string* and the special token **.EMPTY.**. A *lookup-list* is a field that may only contain predefined values, or the special token **.ANY.**, indicating that any value is acceptable. "Status" and "Type" are typical examples of fields with a limited range of predefined values.
The **.ANY.** token, if used, is to be interpreted exactly as if it contained all possible values for the given field. In particular, the use of **.ANY.** does not alter any limitation on the number of lookup values allowed for the field. It is merely a shorthand method of specifying all possible lookup values.

range fields can be searched based on a range of values. "ListPrice" and "ListDate" fall into this category. All values specified in a *range* are to be treated as inclusive (e.g. 2+ is the same as 2 or greater, inclusive of 2; 2-3 is the same as 2 to 3, inclusive of 2 and 3; 2- is the same as 2 or less, inclusive of 2). The types of the range endpoints MUST match the data type of the field being searched. In addition, the *range-start* value MUST be less than the *range-end* value in the value space defined by the searched field, or the result is undefined.

A *string* field is any other character field not falling into the other two categories. These are usually freeform text fields. An example of this kind of field is "OwnerName".

The special value **.EMPTY.** is to be interpreted as whatever the value of the field would be if no value had been entered.

Note that this is implementation-defined: it may be the same as a search for a null value, or it may be blank or zero. A client should not expect to be able to distinguish unentered values from any other values using this search token.

Each *field* MUST be a SystemName, as defined in the metadata, unless the StandardName argument is set to "1", in which case the *field*s MUST be StandardNames. All values submitted for lookup-lists must be the Value in compact format, as defined in Section 13 .

The data types for field values may be determined by examining the metadata for the searched field. In a query using StandardNames, the RETS Data Dictionary gives the acceptable data type for search values.

7.7.3 Sub-queries

This query language provides for a nesting of sub-queries. For example:

```
Query=((AREA=|1,2) | (CITY=ACTON)),(LP=200000+)
```

Example:

```
Query=(ST=|ACT,SOLD),
(LP=200000-350000),
(STR=RIVER*),
(STYLE=RANCH),
(EXT=WTRFRNT,DOCK),
(LDATE=2000-03-01),
(REM=FORECLOSE),
(TYPE=~CONDO,TWNHME),
(OWNER=P?LE)
```

Verbally, this would be interpreted as "return properties with (ST equal ACT or SOLD) and (LP between 200000 and 350000, inclusive) and (STR beginning with RIVER) and (STYLE equal RANCH) and (EXT equal WTRFRNT and DOCK) and (LDATE great er than or equal to 2000-03-01) and (REM containing FORECLOSE) and (TYPE not equal to CONDO and not equal to TWNHME) and (OWNER starting with P and having LE in the 3rd and 4th characters)."

7.8 Reply Codes

Table 7-1 Search Transaction Reply Codes

Reply Code	Meaning
0	Operation successful.
20200	Unknown Query Field The query could not be understood due to an unknown field name.
20201	No Records Found No matching records were found.
20202	Invalid Select The Select statement contains field names that are not recognized by the server.
20203	Miscellaneous Search Error The quoted-string of the body-start-line contains text that MAY be displayed to the user.
20206	Invalid Query Syntax The query could not be understood due to a syntax error.
20207	Unauthorized Query The query could not be executed because it refers to a field to which the supplied login does not grant access.

20208	Maximum Records Exceeded Operation successful, but all of the records have not been returned. This reply code indicates that the maximum records allowed to be returned by the server have been exceeded. Note: reaching/exceeding the "Limit" value in the client request is not a cause for the server to generate this error.
20209	Timeout The request timed out while executing
20210	Too many outstanding queries The user has too many outstanding queries and new queries will not be accepted at this time.
20211	Query too complex The query is too complex to be processed. For example, the query contains too many nesting levels or too many values for a lookup field.
20212	Invalid key request The transaction does not meet the server's requirements for the use of the Key option.
20213	Invalid Key The transaction uses a key that is incorrect or is no longer valid. Servers are not required to detect all possible invalid key values.
20514	Requested DTD version unavailable. The client has requested the metadata in STANDARD-XML format using a DTD version that the server cannot provide.

SECTION 8 - Get Transaction

Gets an arbitrary file from the server or performs an arbitrary action, specified by URI. This is a standard HTTP GET, per RFC 2616. The file to get is passed as part of the
Request-URI.

RETS servers need not support the GET transaction to any greater extent than is necessary to implement the functionality of the Action URL (see 4.10, “Capability URL List”). If a RETS server does not intend to include an Action URL in its login responses, it need not support the GET transaction.

- 8.1 Required Request Arguments
- 8.2 Optional Request Arguments
- 8.3 Required Response Arguments
- 8.4 Optional Response Arguments
- 8.5 Status Conditions

8.1 Required Request Arguments

There are no required request arguments.

8.2 Optional Request Arguments

There are no optional request arguments.

8.3 Required Response Arguments

There are no required response arguments.

8.4 Optional Response Arguments

There are no optional response arguments.

8.5 Status Conditions

See the General Status Codes in Section 3.9 for typical Status-Codes.

SECTION 9 - Change Password Transaction

- 9.1 Required Request Arguments
- 9.2 Optional Request Arguments
- 9.3 Required Response Arguments
- 9.4 Optional Response Arguments
- 9.5 Reply Codes
- 9.6 Encryption Key Construction
- 9.7 ECB Padding
- 9.8 Effect of change

9.1 Required Request Arguments

PWD	**::= PWD= <***BASE64***(<*DES**(* *Password* **:** *UserName* **)>)**

This is the Base64 representation of the DES-encrypted UserName and Password. The new Password and the UserName are appended together with a colon (":") between and the resulting string is encrypted using DES in Electronic Code Book (ECB) mode. The DES key is constructed using the procedure in Section 9.6 . Base64 encoding is defined in RFC 2045 section 6.8.

9.2 Optional Request Arguments

There are no optional request arguments.

9.3 Required Response Arguments

There are no required response arguments.

9.4 Optional Response Arguments

There are no optional response arguments.

9.5 Reply Codes

Table 9-1 Change Password Reply Codes

Reply Code	Meaning
0	Operation successful.
20140	Insecure password. The password does not meet the site's rules for password security.
20141	Same as Previous Password. The new password is the same as the old one.
20142	The encrypted user name was invalid.

9.6 Encryption Key Construction

The new password is communicated to the host as a string encrypted with the Data Encryption Standard, ANSI X3.92. DES requires a 64-bit key, which is constructed as follows:

1. The old password and username are converted to uppercase and concatenated together.
2. The resulting string is hashed using MD5.
3. The key is taken as the first 64 bits of the resulting hash value. Parity bits must be corrected for encoders that check parity.

9.7 ECB Padding

The input to the DES ECB encryption process shall be padded to a multiple of 8 octets in the following manner:

Let *n* be the length in octets of the input. Pad the input by appending 8 – (*n* mod 8) octets to the end of the input, each having the value 8 – (*n* mod 8), the number of octets being added. In hexadecimal, the possible paddings are `0x01`, `0x0202`, `0x030303`, `0x04040404`, `0x0505050505`, `0x060606060606` and `0x07070707070707` and `0x0808080808080808`. All input is padded with 1 to 8 octets to produce an input string that is a multiple of 8 octets in length. The padding can be unambiguously removed after decryption.

This padding method is compatible with RFC 2315 section 10.3, note 2.

9.8 Effect of change

Servers that return a success status MUST accept the new password and reject the old password for all subsequent Login transactions and sessions. Servers that return a success status MAY require the use of the new password for all subsequent transactions in the current session by issuing a WWW-Authenticate challenge for transactions that do not contain the correct credentials.

If a client fails to receive a response to this transaction, it SHOULD retain both the old and new passwords until the effect of the Change Password transaction can be ascertained via a successful login.

SECTION 10 - Update Transaction

The update transaction is used to modify data on the server. The client transmits information describing the update to perform. The information is then validated by the server. If there are errors in the data, the server returns an error reply. If there are no errors, the record as it was inserted/updated on the server will be returned. The record is returned in the same manner as a record is returned from a search.

Update requests MUST use the POST method (rather than the GET method). This allows the client to transmit characters beyond the HTTP length limit for the GET method. The request MUST use a content-type appropriate to the encoding of the request, per [16]. A content-type of `text/www-url-formencoded` is recommended, but any other method of encoding HTML form parameters may be used.

- 10.1 Required Request Arguments
- 10.2 Optional Request Arguments
- 10.3 Required Response Arguments
- 10.4 Optional Response Arguments
- 10.5 Update Response Body Format
- 10.6 Validation
- 10.7 Reply Codes

10.1 Required Request Arguments

The request has the following format:

```
Resource= resource-name
&ClassName= class-name
&Validate= validate-flag &Type= update-type
&Delimiter= field-delimiter
&Record= field-name = field-value *( field-delimiter field-name = field-value )
[&WarningResponse= warning-response *(field-delimiter warning-response)
```

resource-name	::= *1*32ALPHANUM*

The name of the resource to be updated, as specified in the metadata. This is the SystemName as defined in Section 11.2.2 .

class-name	::= *1*24ALPHANUM*

The name of the class to be updated, as defined in the metadata. This is the ClassName as defined in section 11.3.1 .

validate-flag	::= **0** | **1** | **2**

If this parameter is set to one ("**1**"), then the record is validated by the host. Any fields with metadata field "Attributes" set to "Autopop" in the metadata (see Section 11.3.4) will have their field values filled in by the server and returned to the client. The record in the server database is not updated. If this entry is set to zero ("**0**") and there are no errors in the record the record is updated on the server. If this entry is set to two ("2"), the server validates all fields and returns any errors found, but does *not* store the record.

update-type	::= *1*24 ALPHANUM*}}

The type of update to perform, as specified by the metadata. This is the UpdateType as defined in Section 11.3.4 .

field-name	::= *1*32ALPHANUM*

The name of the field to be updated, as specified in the metadata. This is the SystemName as defined in Section 11.3.2 .

field-delimiter	::= *OCTET*

The octet which will separate fields in the record. If this is not specified, an ASCII HT character is assumed.

field-value	::= <varies depending on the field>

The text representation of the field value as defined by the metadata in Section 11.3.2 subject to the business rules. The value MUST be submitted as if in COMPACT format.

warning-response	::= *warning-num* **=** *user-response*
warning-num	::= *1*5DIGIT*
user-response	::= **256TEXT excluding* delimiter

The *warning-num* value is the host warning number that was returned in the prior Update Response body. The *user-response* value is the text of the warning response in response to the specified warning. If a *warning-num* sent in the prior UpdateResponse body had a *response-required* value of 2, then the *user-response* value MUST NOT be null.

10.2 Optional Request Arguments

There are no optional request arguments.

10.3 Required Response Arguments

There are no required response arguments.

10.4 Optional Response Arguments

There are no optional response arguments.

10.5 Update Response Body Format

The body of the update response has the following format when there are no errors:

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP > CRLF
transaction-id-tag
[ delimiter-tag ]
column-tag
compact-data
[<RETS-STATUS 1*SP ReplyCode= quoted-end-reply-code 1*SP
ReplyText= quoted-string *SP/>
</RETS> CRLF
```

The body of the update response has the following format when there are errors or warnings:

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
ReplyText= quoted-string *SP > CRLF
transaction-id-tag
[ delimiter-tag ]
column-tag
compact-data
[error-block]
[warning-block
</RETS> CRLF
```

error-block	= <ERRORBLOCK> *CRLF* 1*(<ERRORDATA>*fielderror-numerror-offseterror-text* </ERRORDATA>) </ERRORBLOCK>
warning-block	= <WARNINGBLOCK> 1*(<WARNINGDATA>*fieldwarning-numwarning-offsetwarning-textresponse-required* </WARNINGDATA>) </WARNINGBLOCK>

The format of the <ERRORDATA> and <WARNINGDATA> tag content is identical to COMPACT format.

10.5.1 Error block

An Error Block is returned when there is a problem with one or more of the fields. The error block contains information about the fields that have errors. It contains the field name, an error number, some additional text about the error (*error-text*), and where in the field data the error occurred (*error-offset*).

error-num	::= *1*5DIGIT*

This is the host error number. This number along with the *error-text* MAY be displayed to the user when looking at the corresponding field in the client application.

error-offset	::= *1*5DIGIT*

This is the offset into the field data that was sent by the client application to the server. It indicates at what character in the field data the problem was encountered. This number is set to zero ("0") if the offset of the error is unknown.

error-text	::= *64ALPHANUM

This is the error text generated by the host to assist the user in determining the problem with the field data. This text is associated with the *error-num*.

The error return format follows the COMPACT data format in all particulars. This affects primarily the quoting of special characters and the selection of the delimiter that separates the field values. In effect, the error return is a COMPACT data block

without the usual COLUMNS element.

10.5.2 Warning block

A Warning Block is returned when there is a problem with one or more of the fields that would not prevent the record from being saved in the database. It contains a field name, a warning number, some additional text about the warning (*warning-text*), where in the field data the warning occurred (*warning-offset*) and an indicator whether an end-user response to this warning is requested or required. The delimiter is the same as the one defined for the *error-block*.

field	::= *1*32ALPHANUM*

The SystemName of the field to which the warning applies.

warning-num	::= *1*5DIGIT*

The host warning number. This number, along with the *warning-text*, MAY be displayed to an end-user in association with the corresponding field in the client application.

warning-text	::= *TEXT*
warning-offset	::= *1*5DIGIT*

The offset into the field data that was sent by the client application to the server. It indicates at what character in the field data the problem was encountered. This number is set to zero if the offset of the error is unknown or if an offset is inapplicable.

response-required	::= **0** | **1** | **2**

The *response-required* value indicates whether an end-user response is requested or required:

0	No response is permitted.
1	A response is requested.
2	A response is mandatory.

If the *response-required* field indicates that a response is mandatory, the client MUST send the end-user response for the specific warning-num in the WarningResponse request argument in order for this record to be saved to the database.

10.6 Validation

Validation routines are indications of the checks the host system will perform against a field value before it is accepted for storage on the host. Some of these routines require data available only on the host system. However, others are relatively simple and could be performed by any RETS client to prevent invalid field values from being submitted.There are several different types of validation to be performed by the client.

A compliant client is not required to enforce the local validations provided in this section. However, if a client does not enforce the validations then the likelihood of the server rejecting the record is greatly increased.

10.6.1 Lookup

The entry is validated against a list of acceptable values. If the metadata described in Section 11.3.2 specifies the Interpretation as Lookup the only acceptable values for the field are defined in the METADATA-LOOKUP referenced by LookupName. Alternatively, if the metadata specifies a ValidationLookup the only acceptable values for the field are defined in the METADATA-VALIDATION_LOOKUP referenced by the ValidationLookup field.

10.6.2 MultiSelect Lookup

The entry is validated against a list of acceptable values. If the metadata described in Section 11.3.2 specifies the Interpretation as LookupMulti, LookupBitstring or LookupBitmask the only acceptable values for the field are defined in the METADATA-LOOKUP referenced by LookupName. The maximum number of values that can be selected is defined by MaxUpdate.

10.6.3 Range

The entry must be between the `Minimum` and `Maximum` values specified in the metadata (see Section 11.3.2).

10.6.4 Test Expression

The parameter list contains an expression evaluated by the routine. If the expression is true, the value of the field is acceptable. If the expression is false, the value is rejected. See Section 11.4.9 for more information on Test Expressions. Test expressions are always exceuted in the order in which they are presented in the metadata.

10.6.5 External

The entry may be validated by searching a server resource. The Resource is defined for searching and the parameter list includes a set of suggested input fields, a set of result fields to display and a set of result fields to populate into the fields of the resource being updated. Information for external validation is provided in Section 11.4.10 .

10.7 Reply Codes

Table 10-1 Update Transaction Reply Codes

Reply Code	Meaning
0	Operation successful.
20301	Invalid parameter. Additional information is provided in the error block.
20302	Unable to save record on server.
20303	Miscellaneous Update Error.
20311	`WarningResponse` was not given for all warnings that contained a *`response-required`* value of 2.
20312	`WarningResponse` was given for a warning that contained a *`response-required`* value of 0.

The quoted-string of the body-start-line contains text that MAY be displayed to the user.

SECTION 11 - Metadata Format

Metadata enables a client that receives data from a compliant server to better format the data for display, and to store it efficiently for future retrieval. While use of the metadata is not necessary to retrieve data for simple display purposes, more sophisticated clients will want to use the metadata to make more intelligent use of the information retrieved. Metadata MUST be supplied by a compliant server.

- 11.1 Organization and Retrieval
- 11.2 System-Level Metadata
- 11.3 Metadata Format for Class Elements
- 11.4 Metadata Format for Shared Elements

11.1 Organization and Retrieval

11.1.1 Metadata Organization

Metadata is organized by table/object, with each table having its own unique set of metadata describing the fields available in that table/object. The organization permits access to summary or detailed information about one or more resources (see Figure 11.1 "Metadata Structure").

The client retrieves the metadata by using the GetMetadata Transaction specifying the METADATA table/object(s) of interest as the Type, and the specific instance in the ID (see Section 5). The server supplies the metadata as documents using the formats described in this section. The client MUST accept fields and attributes in the metadata that are not defined in this standard, although it is not required to process those fields in any way.

The client may cache the metadata between sessions. If it does, it MUST record the value of the `METADATA-SYSTEM` timestamp attribute from each session in which it caches retrieved metadata, and MUST request new metadata whenever the `MetadataTimestamp` Login response value changes except when previous versions are permitted by the `MinMetadataTimestamp` value. If a client continues to send transactions using outdated metadata, the server's operation is undefined.

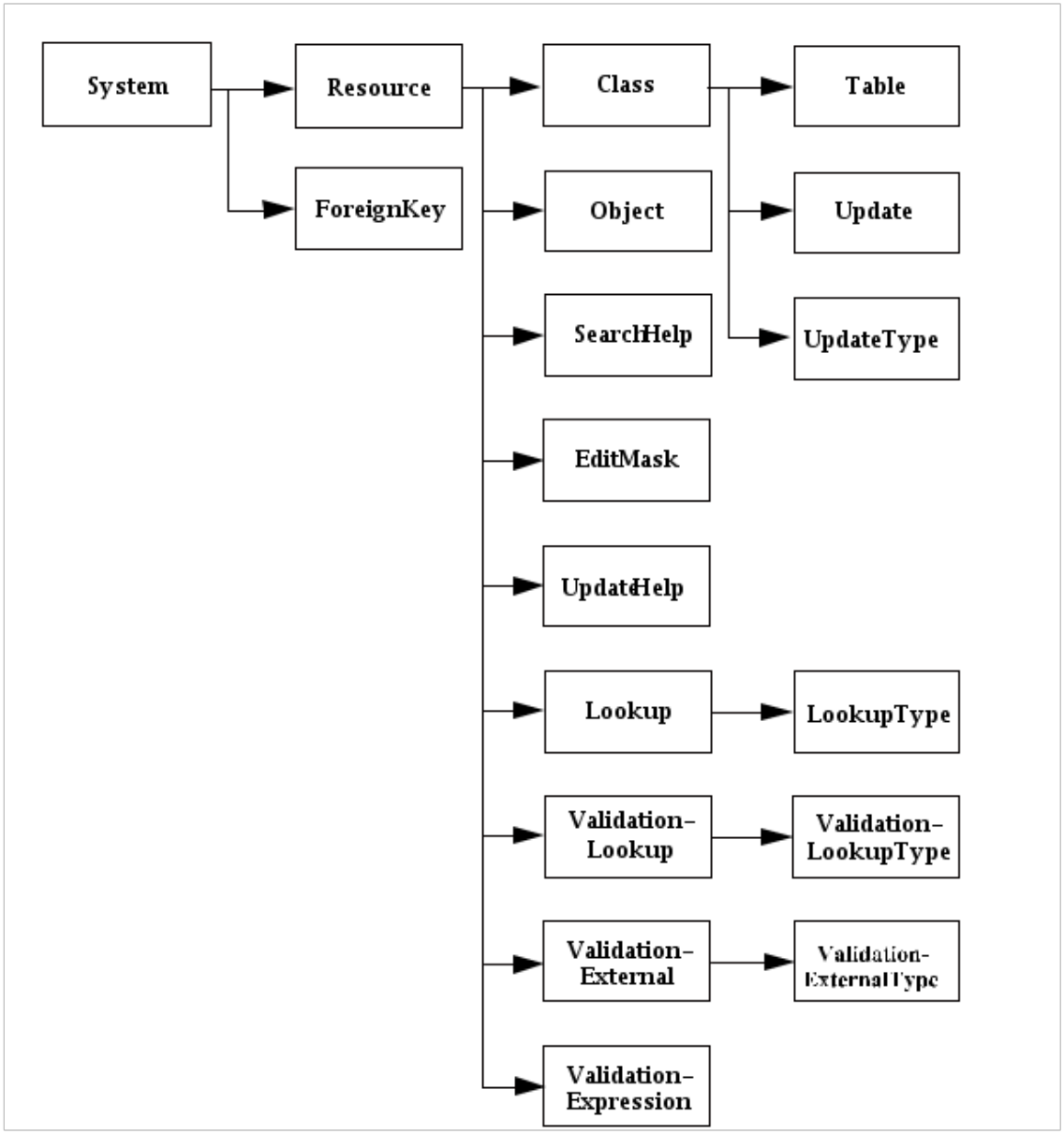


Figure 11.1 Metadata Structure

11.1.2 General Rules for Interpretation

In general, metadata keywords defined in this standard such as field names and reserved values are not case-sensitive. However, implementors are urged to adopt the strict-generation/tolerant-acceptance rule and follow the case shown in this standard.

Servers may choose to extend the content of any metadata table by including additional keywords. Metadata field names for such extensions SHOULD begin with the letter "X" followed by a hyphen, followed by an implementation-defined token in

order to insure compatibility with future versions of the standard.

Clients requesting metadata in COMPACT format MUST ignore any metadata fields which they do not understand. In addition, the servers are permitted to send columns in any order. The order shown in the examples is not normative.

Clients requesting metadata in XML format MUST ignore any <EXTENSION> or <PROPRIETARY> elements that they do not understand.

NOTE:

> RETS 1.7 requires all server responses to be well-formed XML, and additionally requires GetMetadata responses to be valid XML. In addition, RETS requires that clients parse server responses as XML, not as simple text streams. The response formats shown here are normative with respect to content, but not normative with respect to form. That is, servers are free to produce response XML in any format that complies with the W3C XML 1.0 recommendation, so long as it is valid with respect to the appropriate DTD. XML escaping of content is implied, as is XML processing of whitespace and line endings. See the W3C *XML Recommendation 1.0, Third Edition*, for full information on XML.

11.1.3 Metadata Retrieval Hierarchy

The ID argument in the GetMetadata transaction reflects the metadata hierarchy as shown in Figure 11.1. For any metadata element, the ID argument is a list of the names of the parent elements for the desired element, separated by colons. For example, to retrieve the EditMask table for a given named Resource, the argument is simply the ResourceID:

`Type:`	`METADATA-EDITMASK`
`ID`	`Propert`

where `Property` is the `ID` of one of the Resources listed in the Metadata-Resource table.

Since Tables are children of Classes, which are in turn children of Properties, the ID parameter contains both parents:

`Type`	`METADATA-TABLE`
`ID`	`Property : Res`

where `Res` is a class listed in the Metadata-Class table under the resource `Property`.

11.1.4 Metadata Format

Compliant RETS servers MUST supply metadata in either of two formats: COMPACT, described below and valid according to the RETS Compact DTD (public identifier `-//RETS//DTD Compact Metadata 1.7//EN`), or XML, valid according to the RETS XML Metadata DTD (public identifier `-//RETS//DTD Metadata Content 1.7//EN`). See Appendix A for system identifiers.

The COMPACT metadata format consists of a sequence of segments with identical structure, except for System-level metadata, which has its own structure. The general structure for non-System metadata is a tab-delimited table, XML-encapsulated with the header record contained within a <COLUMNS> element, and each successive row contained within a <DATA> element.:

```
<METADATA-HEADER header-attributes>
<COLUMNS>fieldname *(fieldname)</COLUMNS>
*(<DATA>fielddata *(fielddata)</DATA>)
</METADATA-HEADER>
```

METADATA-HEADER is the header name for the segment, given with the description of each type of metadata, as are the *header-attributes* associated with each header. Each *fieldname* is the name of one of the metadata fields given below. Each *fielddata* value corresponds to the similarly-positioned *fieldname*, first to first, second to second and so on.

11.2 System-Level Metadata

Clients can determine the number and type of searchable and updateable entities by referencing the Resources. A server MUST advertise its resources. It MAY advertise all of its available resources or MAY restrict the advertised list by logon or

other criteria. A server's advertisement of a resource does not require that the server be able to accommodate any arbitrary search for that user; the server MAY restrict access to resources that it advertises. If the server supports multimedia objects then it MUST advertise the supported types.

All resources that can be searched or updated are defined in the metadata described in this section. There are three parts to the metadata. The first part provides system information and describes the available resources, the second part describes the class specific metadata for a resource, and the third part describes the shared metadata for a resource.

11.2.1 System

The System metadata starts with a <METADATA-SYSTEM> tag with Version and Date attributes. This tag is followed by a <SYSTEM> section, which contains the system identification information. An optional <COMMENTS> section completes the System metadata. The System metadata has the following format:

```
<METADATA-SYSTEM · Version="system-version" · Date="system-date" >
<SYSTEM · SystemID="code-name" · SystemDescription="long-name" />
[<COMMENTS>
*( comment )
</COMMENTS> ]
</METADATA-SYSTEM>
```

System Version

system-version	::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version of the document. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time any metadata element changes the version number MUST be increased.

System Date

system-date	::= *DATE*

The System Date is latest change date of any System metadata.

System Description

code-name	::= *1*10ALPHANUM*
long-name	::= *1*64PLAINTEXT*
comments	::= *TEXT*

11.2.2 Resources

RETS does not require that any particular type of data be made available by a server. However, a server MUST use a standard well-known name under which to make its data available if a suitable well-known name is defined in the standard. Table 11-1 contains the list of well-known resource names.

Table 11-1 Well-Known Resource Names

Resource Name	Purpose
ActiveAgent	A resource that contains information about active agents. These are agents that are currently authorized to access the server (paid-up, not retired, etc.)
Agent	A resource that contains information about agents.

History	A resource that contains information about the accumulated changes to each listing.
Office	A resource that contains information about broker offices.
OpenHouse	A resource that contains information about open-house activities.
Property	A resource that contains information about listed properties. Information in this resource is described by Real Estate Transaction XML DTD in addition to appropriate metadata.
Prospect	A resource that contains information about sales or listing prospects.
Tax	A resource that contains tax assessor information.
Tour	A resource that contains information about tour activities.

Resource Metadata Content

Table 11-2 Resource Metadata Compact Header Attributes

COMPACT header tag: **METADATA-RESOURCE**

Attribute	Content
Version	This is the version of the Resource metadata. The convention used is a "<*major*>.<*minor*>.<*release*>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.

Table 11-3 Metadata: Resource Description Fields

Field Name	Content Type	Description
ResourceID	*1*32ALPHANUM*	The name which acts as a unique ID for this resource.
StandardName	*1*64ALPHANUM*	The name of the resource. This must be a well-known name if applicable.
VisibleName	*1*64PLAINTEXT*	The user-visible name of the resource.
Description	*1*64PLAINTEXT*	A user-visible description of the resource.
KeyField	*1*32ALPHANUM*	The SystemName (see table 11-3) of the field that provides a unique ResourceKey for each element in this resource. All classes within a resource must use the same KeyField.
ClassCount	Numeric	The number of classes in this resource. There MUST be ClassCount METADATA_CLASS descriptions for the resource. There MUST be at least one Class for each Resource.

ClassVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The latest version of the Class metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only.
ClassDate	*DATE*	The date on which the Class metadata for this Resource was last changed. Clients MAY rely on this date for cache management.
ObjectVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the Object metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no Object metadata is available for this Resource.
ObjectDate	*DATE*	The date on which the Object metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no Object metadata is available for this Resource.
SearchHelpVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the SearchHelp metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no SearchHelp is available for this Resource.
SearchHelpDate	*DATE*	The date on which the SearchHelp metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no SearchHelp is available for this Resource.
EditMaskVersion	1*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the EditMask metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no EditMask is available for this Resource.
EditMaskDate	*DATE*	The date on which the EditMask metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no EditMask is available for this Resource.
LookupVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the Lookup metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no Lookup is available for this Resource.

LookupDate	*DATE*	The date on which the Lookup metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no Lookup is available for this Resource.
UpdateHelpVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the UpdateHelp metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no UpdateHelp is available for this Resource.
UpdateHelpDate	*DATE*	The date on which the UpdateHelp metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no UpdateHelp is available for this Resource.
Validation-ExpressionVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the ValidationExpression metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no ValidationExpression is available for this Resource.
Validation-ExpressionDate	*DATE*	The date on which the ValidationExpression metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no ValidationExpression is available for this Resource.
ValidationLookupVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the ValidationLookup metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no ValidationLookup is available for this Resource.
ValidationLookupDate	*DATE*	The date on which the ValidationLookup metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no ValidationLookup is available for this Resource.

ValidationExternalVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the ValidationExternal metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only. A blank version indicates no ValidationExternal is available for this Resource.
ValidationExternalDate	*DATE*	The date on which the ValidationExternal metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no ValidationExternal is available for this Resource.

11.2.3 Foreign Keys

The ForeignKeys metadata table allows a server to advertise relationships among its offered resources. A RETS client MAY use this information to provide a richer display of related information. The ForeignKeys metadata consists of tuples containing a parent resource type, a child resource type, and the foreign keys used to traverse the relation.

The nesting of foreign keys MUST be such that recursive searches are NOT REQUIRED to obtain data for well-known fields as defined in the RETS DTD. However, nesting of foreign keys is allowed except in these cases.

ForeignKeys Metadata Content

COMPACT header tag: **METADATA-FOREIGNKEYS**

Table 11-4 ForeignKeys Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the ForeignKeys metadata. The convention used is a "*<major>.<minor>.<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.

Table 11-5 Metadata Content: Foreign Keys

Metadata Field	Content Type	Description
ForeignKeyID	*RETSNAME*	A Unique ID that represents the foreign key combination.
ParentResourceID	*RETSNAME*	The ResourceID (Table 11-3) of the resource for which this field functions as a foreign key. The name given MUST appear in the METADATA-RESOURCE table..
ParentClassID	*RETSNAME*	The name of the resource class for which this field functions as a foreign key. This name MUST appear in the RESOURCE-CLASS table for the given ParentResourceID.

ParentSystemName	*RETSNAME*	The SystemName of the field in the given resource class that should be searched for the value given in the this field. This name must appear as a SystemName in the METADATA-TABLE section of the metadata for the ParentClassID, and the named item must have its Searchable attribute set to TRUE.
ChildResourceID	*RETSNAME*	The ResourceID (Table 11-3) of the resource for which this field functions as a foreign key. The name given MUST appear in the METADATA-RESOURCE table.
ChildClassID	*RETSNAME*	The name of the resource class for which this field functions as a foreign key. This name MUST appear in the RESOURCE-CLASS table for the given ChildResourceID.
ChildSystemName	*RETSNAME*	The SystemName of the field in the given resource class that should be searched for the value given in this field. This name must appear as a SystemName in the METADATA-TABLE section of the metadata for the ChildClassID, and the named item must have its Searchable attribute set to TRUE.
ConditionalParentField	*RETSNAME*	The SystemName of a field in the parent's METADATA-TABLE that should be examined to determine whether this parent-child relationship should be used. If this is blank, the relationship is unconditional. If ConditionalParentField is present and nonblank, then ConditionalParentValue MUST be present and nonblank.
ConditionalParentValue	*RETSNAME*	The value of the field designated by ConditionalParentField indicating that this relation should be used. If the type of the field named in ConditionalParentField is numeric, then this value is converted to numeric type before comparison. If the type of the field named in ConditionalParentField is character, then the shorter of the two values is padded with blanks and the comparison made for equal length. If ConditionalParentField is present and nonblank, then ConditionalParentValue MUST be present and nonblank.

11.3 Metadata Format for Class Elements

All tables that can be searched are defined in a document with the format defined in this section. There are three parts to this section. The first part describes the searchable tables, the second part describes the lookups referenced within the table

section, and the third describes the help text associated with searches and edit masks associated with updates.

11.3.1 Class

A given data resource may contain multiple classes of entries that can be searched or updated separately. The metadata for a resource supporting searchable classes MUST contain a class description for each class supported.
COMPACT header tag: **`METADATA-CLASS`**

Table 11-6 Class Metadata Compact Header Attributes

Attribute	Content
`Version`	This is the version of the Class metadata. The convention used is a "<*major*>.<*minor*>.<*release*>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
`Date`	The latest change date of any contained metadata. This is an RFC 1123 date.
`Resource`	The ResourceID for the resource in which this class resides.

Table 11-7 Metadata Content: Resource Class

Metadata Field	Content Type	Description
`ClassName`	*`RETSNAME`*	The name which acts as a unique ID for the class.
`StandardName`	***`Residential-Property`*** ***`LotsAndLand`*** ***`CommonInterest`*** ***`MultiFamily`***	The XML standard name. This is the name from the Real Estate Transaction XML DTD.
`VisibleName`	*`1*64PLAINTEXT`*	The user-visible name of the class.
`Description`	*`1*128PLAINTEXT`*	A user-visible description of the class.
`TableVersion`	*`1*2DIGITS . 1*2DIGITS . 1*5DIGITS`*	The version of the Table metadata that describes this Class. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only.
`TableDate`	*`DATE`*	The date on which the Table metadata for this Class was last changed. Clients MAY rely on this date for cache management.
`UpdateVersion`	*`1*2DIGITS . 1*2DIGITS . 1*5DIGITS`*	The latest version of any of the Update metadata for this Class. The convention used is a "<major>.<minor>.<release>" numbering scheme. A blank version indicates no Update is available for this Class. The version number is advisory only.

UpdateDate	*DATE*	The date on which any of the Update metadata for this Class was last changed. Clients MAY rely on this data for cache management. A blank date indicates no Update is available for this Class.
ClassTimeStamp	RETSNAME	The SystemName of the field in the METADATA-TABLE that acts as the last-change timestamp for this class.
DeletedFlagField	RETSNAME	The SystemName of the field in the METADATA-TABLE that indicates that the record is logically deleted. If this element is specified, then DeletedFlagValue MUST be specified as well.
DeletedFlagValue	1*32ALPHANUM	The value of the field designated by DeletedFlagField indicating that a record has been logically deleted. If the type of the field named by DeletedFlagField is numeric, then this value is converted to a number before comparison. If the type of the field named by DeletedFlagField is character, then the shorter of the two values is padded with blanks and the comparison made for equal length.

11.3.2 Table

The following table lists the minimum acceptable content for server-supplied metadata used in describing a table.
COMPACT header tag: **METADATA-TABLE**

Table 11-8 Table Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Table metadata. The convention used is a "*<major>.<minor>.<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource in which this table resides.
Class	The ClassName for the class in which this table resides.

Table 11-9 Metadata Content - Tables

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that never changes as long as the semantic definition of this field remains unchanged. In particular, it should be managed so as to allow the client to detect changes to the SystemName.

SystemName	*RETSNAME*	The name of the field as it is known to the native server. The system name MUST be unique within the Table.
StandardName	*1*64IDALPHA*	The name of the field as it is known in the Real Estate Transaction XML DTD.
LongName	*1*256TEXT*	The name of the field as it is known to the user. This is a localizable, human-readable string. Use of this field is implementation-defined; it is expected that clients will use this value as a title for this datum when it appears on a report.
DBName	*1*10ALPHANUM*	A short name that can be used as a database field name. This name may not start with a number nor can it be an ANSI-SQL92 reserved word. This value can be used by a client as the name of an internal database field, so servers should attempt to provide a value for this field that is unique within the table.
ShortName	*1*64TEXT*	An abbreviated field name that is also localizable and human-readable. Use of this field is implementation-defined. It is expected that clients will use this field in human-interface elements such as pick lists.
MaximumLength	Numeric	The maximum length of the field, in characters. For numeric fields (small, int, long and decimal) this is the display length rather than the storage length, and includes all formatting such as the sign, decimal point, commas or other insertion edits.
DataType	**Boolean**	A truth-value, stored as 1 for true and 0 for false.
	Character	An arbitrary sequence of printable characters.
	Date	A date, in YYYY-MM-DD format.
	DateTime	A timestamp, in YYYY-MM-DD Thh:mm:ss[.sss] format.
	Time	A time, stored in hh:mm:ss[.sss] format.
	Tiny	A signed numeric value that can be stored in no more than 8 bits.
	Small	A signed numeric value that can be stored in no more than 16 bits.
	Int	A signed numeric value that can be stored in no more than 32 bits.
	Long	A signed numeric value that can be stored in no more than 64 bits.

	Decimal	A decimal value that contains a decimal point (see Precision).
Precision	Numeric	The number of digits to the right of the decimal point when formatted.
Searchable	Boolean	A truth-value which indicates that the field is searchable.
Interpretation	**Number**	An arbitrary number.
	Currency	A number representing a currency value.
	Lookup	A value that should be looked up in the Lookup Table. This is a single selection type lookup (e.g. STATUS). This interpretation is also valid for Boolean data types, in which case the LookupType specified by the Lookup Name entry MUST contain exactly two elements, one with a Value of 0, and the other with a Value of 1.
	LookupMulti	A value that should be looked up in the Lookup Table. This is a multiple-selection type lookup (e.g. FEATURES) where the character strings representing each selection are separated by commas.
	LookupBitstring	A value that should be looked up in the Lookup Table. This is a multiple-selection lookup that is stored as a bit string. The bit string is represented as a character string containing only the characters 0 and 1. The leftmost character represents the least-significant bit. The lookup value of the bitstring element is the ordinal position of each bit with the rightmost bit designated as bit 0.
	LookupBitmask	A value that should be looked up in the Lookup Table. This is a multiple-selection type lookup that is stored as a bitmask field. Fields of this type are limited to 31 choices.(e.g. VIEW). When converted to binary, each bit represents one of the possible choices. The choices are from lsb to msb. Lookup values are the numeric equivalent of each bit's binary value (i.e., the low order bit represents the first lookup and the high order bit represents the last lookup choice). 2^(value--1)^ is added to the total choice when querying for its applicability.
Alignment	**Left**	The value MAY be displayed left aligned.
	Right	The value MAY be displayed right aligned.

	Center	The value MAY be centered in its field when displayed.
	Justify	The value MAY be justified within its field when displayed.
UseSeparator	Boolean	A truth-value which indicates that the numeric value MAY be displayed with a thousands separator.
EditMaskID	*RETSNAME* Multiple masks are separated by commas	The name of the entry in the METADATA-EDITMASK table (see 11.4.5, "Edit Mask").
LookupName	*RETSNAME*	The name of the METADATA-LOOKUP containing the lookup data for this field (see Section 11.4.2). Required if Interpretation is Lookup, LookupMulti, LookupBitstring or LookupBitmask.
MaxSelect	Numeric	This field is required if Interpretation is LookupMulti, LookupBitstring or LookupBitmask. This value indicates the maximum number of entries that may be selected in the lookup.
Units	(**Feet** | **Meters** | **SqFt** | **SqMeters** | **Acres** | **Hectares**)	Unit of measure.
Index	Numeric	An indicator that specifies this field is part of an index. The client MAY use this information to help the user create faster queries.
Minimum	Numeric	The minimum value that may be stored in a field (applies to numeric fields only).
Maximum	Numeric	The maximum value that may be stored in a field (applies to numeric fields only).
Default	Serial	The order that fields should appear in a default one-line search result. Fields that should not appear in the default one-line format should have a value of 0, Fields that should never be visible to the user should have a value of --1.
Required	Numeric	A non-zero value indicates the field is required when searching. This value should be sequential starting with one. If multiple fields share the same value, then one of the fields with the same value is required. (e.g. City = 1 & ZipCode = 1 implies that the user is required to include either City or ZipCode in their query).
SearchHelpID	RETSNAME	The name of the entry in the METADATA-SEARCH_HELP table (see Section 11.4.4).

Unique	Boolean	A truth-value which indicates that this field is a unique identifier for the record in which it appears.
ModTimeStamp	Boolean	When true, indicates that changes to this field update the class's ModTimeStamp field.
ForeignKey	*RETSNAME*	When nonblank, indicates that this field is normally populated via a foreign key. The value is the ForeignKeyID from the METADATA-FOREIGNKEYS table.
ForeignField	*RETSNAME*	The SystemName from the child record accessed via the specified foreign key.
KeyRetrievalQuery	Boolean	When true, indicates that this field may be included in a query that uses the Key optional argument.
KeyRetrievalSelect	Boolean	When true, indicates that this field may be included in the Select list of a query that uses the Key optional argument.

11.3.3 Update

A given data resource may contain multiple classes of entries that can be updated separately. The metadata for a resource supporting updateable classes MUST contain a Class Table description for each class supported.
COMPACT header tag: **METADATA-UPDATE**

Table 11-10 Update Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Update metadata. The convention used is a "*<major>.<minor>.<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.
Class	The ClassName for the class to which this metadata table applies.

Table 11-11 Metadata Content – Update

Metadata Field	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that never changes so long as the semantic definition of this entry remains unchanged.

UpdateName	*1*24ALPHANUM*	This identifies the nature of the update, such as "add" or "modify". Some update types, such as changes to a property record (e.g. "Sell", "Back on Market"), will imply a set of business rules specific to the server. However, where possible, the following standard type names should be used: **Update Name** / **Function** **Add** / Add a new **Clone** / Create a ne **Change** / Change an **Delete** / Delete an e
Description	*1*64PLAINTEXT*	A user visible description of the Update Type.
KeyField	*1*32ALPHANUM*	The SystemName (see Section 11.3.2) of the field that must be used to retrieve an existing record for the update.
Version	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The latest version of this Update Type metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only.
Date	*DATE*	The date on which any of the content of this Update Type was last changed. Clients MAY rely on this date for cache management.

11.3.4 Update Type

A given resource may contain multiple classes of entries that can be updated separately. Each of these classes may have different types of updates that can be performed. There might be different test expressions or sequences. This section describes how each of those are specified.

COMPACT header tag: **METDATA-UPDATE_TYPE**

Table 11-12 UpdateType Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Update Type metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.

Class	The ClassName for the class to which this metadata table applies.

Table 11-13 Metadata Content – Update Type

Metadata Field	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that never changes as long as the semantic definition of this entry remains unchanged.
SystemName	*RETSNAME*	This is the SystemName of the field as defined in Section 11.3.2 .
Sequence	*1*5DIGIT*	Sequence number of the field, representing the order of entry
Attributes	1*(**1** | **2** | **3** | **4** | **5** [,])	Multiple entries are separated by commas. . **Value** / **Meaning** / **Desc** 1 / DisplayOnly / Field 2 / Required / Field 3 / Autopop / Field 4 / Interactive Validate / Wher the f All fi MUST 5 / ClearOnCloning / The conta
Default	*<PLAINTEXT>*	Default value of field (i.e. value if not specified by user)
ValidationExpressionID	*(RETSNAME [","])*	<multiple entries are separated by commas> The names of the ValidationExpressions to use. See section 11.4.9
UpdateHelpID	*RETSNAME*	The name of the entry in the METADATA-UPDATE_HELP table (see Section 11.4.6).
ValidationLookupName	*RETSNAME*	The name of the ValidationLookup to use. See section 11.4.7
ValidationExternalName	*RETSNAME*	The name of the ValidationExternal to use. See section 11.4.10
MaxUpdate	*1*5DIGIT*	For LookupMulti fields, the maximum number of values that may be specified for the field. This value has no meaning for fields with any other interpretation.

11.4 Metadata Format for Shared Elements

11.4.1 Object

Object type names allow the operator of a particular server to advertise its supported multimedia types. These types are standard MIME types as registered with IANA. RETS does not require that a server make available any particular type of multimedia object. However, a server MUST use a standard well-known name under which to make its multimedia objects available, if a suitable well-known name is defined in the standard. Multimedia names are defined in Table 11-14.

Table 11-14 Well-known Object Types

Object Name	Purpose
`Photo`	A representation image related to the element defined by the resource KeyField.
`Plat`	An image of the property boundaries related to the element defined by the resource KeyField
`Video`	A moving image with or without sound related to the element defined by the resource KeyField.
`Audio`	A sound clip related to the element defined by the resource KeyField.
`Thumbnail`	A lower-resolution image related to the element defined by the resource KeyField.
`Map`	A location image related to the element defined by the resource KeyField.
`VRImage`	A multiple-view, possibly-interactive image related to the element defined by the resource KeyField.

COMPACT header tag: **`METDATA-OBJECT`**

Table 11-15 Object Metadata Compact Header Attributes

Attribute	Content
`Version`	This is the version of the Object metadata. The convention used is a "*<major>.<minor>.<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
`Date`	The latest change date of any contained metadata. This is an RFC 1123 date.
`Resource`	The ResourceID for the resource to which this metadata table applies.

Table 11-16 Metadata Content: Resource Object

Metadata Field	Content Type	Description
`MetadataEntryID`	*`1*32ALPHANUM`*	A value that never changes as long as the semantic definition of this field remains unchanged.
`ObjectType`	*`1*24ALPHANUM`*	The classification of the object. If one of the well-known object types in Table 11-14 applies, then it MUST be used.

MIMEType	A MIME type per RFC 2045	The name of the object type. This is the "mime type" that a client can pass to the "Accept" parameter in the Get Object transaction (see Section 5.1).
VisibleName	*1*64PLAINTEXT*	The user-visible name of the object type.
Description	*1*128PLAINTEXT*	A user-visible description of the object type.
ObjectTimeStamp	*RETSNAME*	The SystemName of the field in a METADATA-TABLE that acts as the timestamp for objects of this type. This SystemName MUST be one that appears in every class that has objects of this type.
ObjectCount	*RETSNAME*	The SystemName of the field in a METADATA-TABLE that acts as the count for objects of this type. This SystemName MUST be one that appears in every class that has objects of this type.

11.4.2 Lookup

This section describes the lookup tables that are referenced by the LookupName in the Table section. There MUST be a corresponding lookup table for every "LookupName".
COMPACT header tag: **METADATA-LOOKUP**

Table 11-17 Lookup Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Lookup metadata. The convention used is a "*<major>.<minor>.<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource in which this table resides.

Table 11-18 Metadata Content: Lookup

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that never changes as long as the semantic definition of this entry remains unchanged.
LookupName	*RETSNAME*	The name of Lookup Table. There MUST be an entry for each LookupName value used in the Table metadata.
VisibleName	*1*32PLAINTEXT*	A description of the table that is human-readable.

Version	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The latest version of this Lookup Table metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only.
Date	*DATE*	The date on which any of the content of this Lookup was last changed. Clients MAY rely on this date for cache management.

11.4.3 Lookup Type

This section describes the content of a lookup table that is referenced by the LookupName in the Table section. There MUST be a corresponding lookup table for every "Lookup", "LookupMulti", "LookupBitstring" and "LookupBitmask".

Table 11-19 Lookup Type Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Lookup Type metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource in which this table resides.
Lookup	The LookupName for the class in which this table resides.

COMPACT header tag: **METADATA-LOOKUP_TYPE**

Table 11-20 Metadata Content: Lookup Type

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that never changes so long as the semantic definition of this entry remains unchanged.
LongValue	*1*128TEXT*	The value of the field as it is known to the user. This is a localizable, human-readable string. Use of this field is implementation-defined; expected uses include displays on reports and other presentation contexts.
ShortValue	*1*32TEXT*	An abbreviated field value that is also localizable and human-readable. Use of this field is implementation-defined; expected uses include picklist values and other human interface elements.

Value	*1*32ALPHANUM*	The value to be sent to the server when performing a search. This field must be numeric for LookupBitmask and LookupBitstring types. For LookupBitmask fields, 2^(value-1)^ is used to compute this component as part of the applicable choices. For LookupBitstring fields, this is the position within the field, 1-based, at which the value contains a"1".

11.4.4 Search Help

This section describes the Search Help text tables that are referenced in the Table section. There MUST be a corresponding table entry for each Search HelpTextID referenced in the METADATA-TABLE.
COMPACT header tag: **METADATA-SEARCH_HELP**

Table 11-21 Search Help Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Search Help metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.

Table 11-22 Metadata Content: Search Help

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that never changes so long as the semantic definition of this entry remains unchanged.
SearchHelpID	*RETSNAME*	A unique ID for the help text. This ID is referenced as the SearchHelpID in section 11.3.2
Value	*1*1024TEXT*	The value to be displayed to the user.

11.4.5 Edit Mask

This section describes the Edit Mask table that is referenced in the Table section. There MUST be a corresponding table entry for each Search EditMaskID referenced in the METADATA-TABLE.
A Regular Expression is used to define the edit mask. Table 11-25 describes the structures that make up RETS regular expressions.

Table 11-23 EditMask Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Edit Mask metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.

Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.

Table 11-24 Metadata Content: Edit Mask

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that remains unchanged so long as the semantic definition of this field remains unchanged.
EditMaskID	*RETSNAME*	A unique ID for the Edit Mask. This ID is referenced as the EditMaskID in section 11.3.2
Value	*1*256TEXT*	The Regular Expression to be used.

COMPACT header tag: **METADATA-EDITMASK**

RETS Regular Expression Specification

RETS regular expressions are a subset of POSIX 1003.2 extended regular expressions [12], supporting the metacharacters in Table 11-25.

Table 11-25 RETS Regular Expression Metacharacters

Metacharacter	Function
. (period)	Matches any single character
*****	Matches zero or more of the preceding pattern
+	Matches one or more of the preceding pattern
?	Matches zero or one of the preceding pattern
|	Alternation: used between two subpatterns, matches either the one to its left or the one to its right.
( ) parentheses	Grouping: causes the enclosed pattern to be treated as atomic. Parentheses may not be nested; that is, only one level of grouping is required.
{*min*[**,***max*]**}** (braces)	Quantifier: matches at least *min* and at most *max* of the preceding pattern, where *min* and *max* are both nonnegative integer values. If *max* is omitted, matches exactly *min* of the preceding pattern.
[ ] brackets	Character class: matches any of the characters contained in the brackets. Except for the circumflex, described below, and the closing bracket, characters within a character class are never treated as metacharacters.
^ (circumflex)	Used as the first character of a character class, reverses the sense of the character class; for example, [^0] matches any character except a "0".

-	Operates only within brackets. Except as the first or last character, denotes a range of characters on the default host collating sequence. For example, [0-9] matches any digit. When - is the first or the last character, it is treated as a member of the character class.
\	Escape: treats the following character as an ordinary character rather than a metacharacter. For example, * matches a single asterisk. The \ character itself must be escaped. The escape character is not needed within character classes.

The following is a simple example:

```
[0-9]+[a-fA-F][1-8][A]?[0-9]{2}[A-C]{1,3}
```

One or more digits, followed by an upper or lower case letter A - F, followed by a digit 1 – 8, optionally followed by one letter A, followed by two digits 0 – 9, followed by between one and three of the letters A – C.
A phone number example:

```
[0-9]{3}[0-9]{4}
```

11.4.6 Update Help

This section describes the Update Help Text tables that are referenced in the Update Type section of the document. There MUST be a corresponding table entry for each Update Help Text ID referenced in any of the `METADATA-UPDATE_TYPEs`.

COMPACT header tag: **`METADATA-UPDATE_HELP`**

Table 11-26 Update Help Metadata Compact Header Attributes

Attribute	Content
`Version`	This is the version of the Update Help metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
`Date`	The latest change date of any contained metadata. This is an RFC 1123 date.
`Resource`	The ResourceID for the resource to which this metadata segment belongs.

Table 11-27 Metadata Content: Update Help

Field Name	Content Type	Description
`MetadataEntryID`	*`1*32ALPHANUM`*	A value that remains unchanged so long as the semantic definition of this entry remains unchanged.
`UpdateHelpID`	*`RETSNAME`*	A unique ID for the help text. This ID is referenced as the UpdateHelpID in section 11.4.6 .
`Value`	*`1*1024TEXT`*	The value to be displayed to the user.

11.4.7 Validation Lookup

This section describes the Validation Lookup tables that are referenced in the Update Type section of the document. There

MUST be a corresponding Validation Lookup Table for each one referenced in the METADATA-UPDATE_TYPEs.

Table 11-28 ValidationLookup Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Table metadata. The convention used is a "<*major*>.<*minor*>.<*release*>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource in which this table resides.

Table 11-29 Metadata Content: Validation Lookup

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that remains unchanged so long as the semantic definition of this entry remains unchanged.
ValidationLookupName	*RETSNAME*	The unique name of this Validation Lookup. Each Name in the Update Type ValidationLookupName field MUST have a definition.
Parent1Field	*RETSNAME*	If a value is present, it is a SystemName field in the same table as defined in Section 11.3.2 and indicates a dependency on this field.
Parent2Field	*RETSNAME*	If a value is present it is a SystemName field in the same table as defined in Section 11.3.2 and indicates an additional dependency on this field.
Version	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of this Validation Lookup metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. This version number is advisory only.
Date	*DATE*	The date on which any of the content of this Validation Lookup metadata was last changed. Clients MAY rely on this date for cache management.

11.4.8 Validation Lookup Type

This section describes the content of the Validation Lookup tables that are referenced in the Table section of the document. There MUST be a corresponding Validation Lookup Type table for each one referenced in the METADATA-UPDATE_TYPE.

The Validation Lookup Type provides a list of all the valid values for a field. This is different than the Lookup described in Section 11.4.2 . The Validation Lookup is used for two cases: 1) the list is too long to be provided as a standard lookup (e.g. Street Name) and 2) there is a dependency on the value in another field. For example, a valid entry for a School District might depend on the Area and SubArea that is entered.

COMPACT header name: **METADATA-VALIDATION_LOOKUP_TYPE**

Table 11-30 Validation Lookup Type Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Validation Lookup metadata. The convention used is a "<*major*>.<*minor*>.<*release*>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource in which this table resides.
ValidationLookup	The ValidationLookupName for the METADATA-VALIDATION_LOOKUP entry to which this entry belongs.

Table 11-31 Metadata Content: Validation Lookup Type

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that remains unchanged so long as the semantic definition of the entry remains unchanged.
ValidText	*1*32ALPHANUM*	A valid value for the field.
Parent1Value	*1*32ALPHANUM*	If this field is present then the ValidText can be used if the data in the Parent1 field is set to this value. If Parent1 is present in the PARENTFIELDS tag then this field is required.
Parent2Value	*1*32ALPHANUM*	If this field is present then the ValidText can be used if the data in the Parent2 field is set to this value. If Parent2 is present in the PARENTFIELDS tag then this field is required.

11.4.9 Validation Expression

This section describes the ValidationExpression table that is referenced in Section 11.3.4 . There MUST be a corresponding table entry for each ValidationExpressionID referenced in the METADATA-UPDATE_TYPEs for a Resource.

The table contains expressions that are to be evaluated when a field value is entered by the user. Expressions in the list MUST be evaluated in the order in which they appear in the list. There are three types of validation expressions, each introduced by a reserved token preceding the expression, given in Table 11-32:

Table 11-32 Validation Expression Types

Keyword	Type	Purpose
ACCEPT	Boolean	If the expression is true, the field value is considered accepted without further testing. Subsequent SET expressions MUST be executed.
REJECT	Boolean	If the expression is true, the field value is considered rejected without further testing. Subsequent SET expressions MUST NOT be evaluated.

SET	Assignment	The expression MUST begin with a field name and an equal sign ("="). The following expression is evaluated and the result stored in the designated field.

Expressions are algebraic formulas containing keywords and operators. Expressions may contain parentheses, and consist of keywords representing any of:

- The current value of any field in the input list
- The current value of any Well-Known Name field in the user's agent record that is returned in the response to the login transaction (see 4.9, "Well-Known Names").
- Literal values.
- A special token (Table 11-18 Metadata Content – Validation Expression Special Operand Tokens).

together with the operators in Table 11-33. Arithmetic operations MUST be carried out using IEEE-754 arithmetic with a representation of at least 64 bits. Comparison operations on strings MUST use simple binary collation. If an error or arithmetic exception occurs during expression evaluation, field value is considered erroneous, regardless of the expression type.

Table 11-33 Validation Expression Operators

Operator	Prece-dence	Operation
/, *, **.MOD.**	1	Division, multiplication, and remainder (modulo)}}
+,**--**	2	Addition and subtraction, applied as follows: 1. If both operands are numeric, the operation is algebraic. 2. If either operand is a string, it is converted to numeric and the operation is algebraic. If an error occurs during the conversion, the field value MUST be rejected. 3. For "+", if either operand is a date, the other must be an integer, a string that can be converted to an integer, or a string representing an interval in ISO8601 format. If no conversion is possible, the field value MUST be rejected 4. For "-", if the left operand is a date or time, the other operand must be a date, a time, or a string representing an interval, and the result must be a string representing an interval in ISO8601 format.
.CONTAINS.	2	A Boolean operator taking strings as its left and right operands. The operation is TRUE if the left operand contains the right operand as a substring anywhere within it.
<, **>**, **<=**, **>=**,	3	Comparison operators with their conventional meaning. If one operand is numeric and the other is a string, the string MUST be converted to a number prior to the comparison. If an error occurs during the conversion, the field value must be rejected.

=, **!=**	4	Comparison operators with their conventional meaning. If one operand is numeric and the other is a string, the string MUST be converted to a number prior to the comparison. If an error occurs during the conversion, the field value must be rejected.
.AND.	5	A Boolean operator that takes two Boolean operands, and whose value is TRUE if and only if both of its operands are TRUE.
.OR.	6	A Boolean operator that takes two Boolean operands, and whose value is TRUE if either of its operands is TRUE.
.NOT.	7	A Boolean operator that takes a single Boolean operand and returns its inverse.

Literal values to be compared against dates or times are expressed in the ISO8601 format.

Table 11-34 Validation Expression Special Operand Tokens

Token	Value
.TODAY.	The current date.
.NOW.	The current time.
.ENTRY.	The current field text, as a string.
.EMPTY.	A value that matches an empty or all-blank field. Supplies an empty (zero-length) field when used in a SET expression.
.OLDVALUE.	The text that was in the field as returned from the host in the search operation. If the field is new, `.OLDVALUE.` is an empty string.
.USERID.	The value of the user-id field returned in the Login transaction (Section 4.9).
.USERCLASS.	The value of the user-class field returned in the Login transaction (Section 4.9).
.USERLEVEL.	The value of the user-level field returned in the Login transaction (Section 4.9).
.AGENTCODE.	The value of the agent-code field returned in the Login transaction (Section 4.9).
.BROKERCODE.	The value of the broker-code field returned in the Login transaction (Section 4.9).
.BROKERBRANCH.	The value of the broker-branch field returned in the Login transaction (Section 4.9).

The Validation Expression metadata starts with a `<METADATA-VALIDATION_EXPRESSION>` tag

Table 11-35 Validation Expression Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Validation Expression metadata. The convention used is a "<*major*>.<*minor*>.<*release*>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.

Table 11-36 Metadata Content: Validation Expression

Field Name	Content Type	Description
MetadataEntryID	*1*32ALPHANUM*	A value that remains unchanged so long as the semantic definition of this entry remains unchanged.
Validation-ExpressionID	*RETSNAME*	A unique ID for the ValidationExpression. This ID is referenced as the ValidationExpression in Section 11.3.4 .
Validation-ExpressionType	*1*32ALPHANUM*	A validation expression type from Table 11-32.
Value	*1*512TEXT*	The test expression to be evaluated.

COMPACT header tag: **METADATA-VALIDATION_EXPRESSION**

11.4.10 Validation External

This section describes the Validation External tables that are referenced in the Update Type section of the document. There MUST be a corresponding Validation External table for each one referenced in any of the METADATA-UPDATE_TYPEs for the Resource.

COMPACT header tag: **METADATA-VALIDATION_EXTERNAL**

Table 11-37 Validation External Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Validation External metadata. The convention used is a "<*major*>.<*minor*>.<*release*>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.

Table 11-38 Metadata Content: Validation External

Field Name	Content Type	Description

MetadataEntryID	*1*32ALPHANUM*	A value that remains unchanged so long as the semantic definition of this entry remains unchanged.
ValidationExternalName	*RETSNAME*	The unique name of this Validation External. Each Name in the Update Type ValidationExternalName field MUST have a definition.
SearchResource	*RETSNAME*	The ResourceID of the Resource to be searched from 11.2.2 .
SearchClass	*RETSNAME*	The ClassName within the Resource to be searched from 11.3.1 .
Version	*1*2DIGITS "."* *1*2DIGITS "."* *1*5DIGITS*	The latest version of this Validation External metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. The version number is advisory only.
Date	*DATE*	The date on which any of the content of this Validation External was last changed. Clients MAY rely on this date for cache management.

11.4.11 Validation External Type

This section describes the content of the Validation External Type tables that are referenced in the Table section of the document. There MUST be a corresponding Validation External Type table for each one referenced in the METADATA-UPDATE_TYPEs for the Resource.

The Validation External Type provides lists of search, display, and results fields. The Validation External may be used for several cases: 1) The database involved is too large or dynamic to be provided as a standard lookup (e.g. Tax). 2) There are business rules that can only be enforced on the server (e.g. expiration dates). 3) The content of a field populates fields from another database (e.g. Sale_agent_name , Sale_office_name , Sale_office_id from Sale_agent_id).

Table 11-39 Validation External Type Metadata Compact Header Attributes

Attribute	Content
Version	This is the version of the Validation External Type metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.
Date	The latest change date of any contained metadata. This is an RFC 1123 date.
Resource	The ResourceID for the resource to which this metadata table applies.
ValidationExternalName	The ValidationExternalName to which this entry type applies.

COMPACT header tag: **METADATA-VALIDATION_EXTERNAL_TYPE**

Table 11-40 Metadata Content: Validation External Type

Field Name	Content Type	Description

MetadataEntryID	*1*32ALPHANUM*	A value that remains unchanged so long as the semantic definition of this entry remains unchanged.
SearchField	*1*512PLAINTEXT*	A comma separated list of valid fields using SystemName from Section 11.3.2 .
DisplayField	*1*512PLAINTEXT*	A comma separated list of valid fields using SystemName from Section 11.3.2 .
ResultFields	*1*1024PLAINTEXT*	A comma separated list of valid field pairs joined by = (equal) the first is a target field in the table being updated and the second is a source field in the table being searched. The fields use a SystemName from Section 11.3.2 .

SECTION 12 - GetMetaData Transaction

The GetMetadata transaction is used to retrieve structured information known as metadata related to the system entities. Metadata requested and returned from this transaction are requested and returned as MIME media types.

- 12.1 Required Client Request Header Fields
- 12.2 Required Request Arguments
- 12.3 Optional Request Arguments
- 12.4 Required Server Response Header Fields
- 12.5 Required Response Arguments
- 12.6 Optional Response Arguments
- 12.7 Metadata Response Body Format
- 12.8 Reply Codes

12.1 Required Client Request Header Fields

There are no additional required client header fields.

12.2 Required Request Arguments

Type	::= <A grouping of related metadata elements (see Section 11)>

The type of metadata being requested. The Type MUST begin with METADATA and MAY be one of the defined metadata types (see Section 11).

ID	::= *metadata-id*[: metadata-id]
metadata-id	::= *1*ALPHANUM* | *

Metadata is organized hierarchically. Each level specifies in its first field an identifier for the metadata contained within that level (e.g. for the Resource level: ResourceID-Agent, Property, etc. for the Lookup level: LookupName-Status, Area, etc.). This identifier can be used to restrict requests to the Type metadata contained within specific instances of higher levels. If the last metadata-id is 0 (zero), then the request is for all Type metadata contained within that level; if the last metadata-id is "*", then the request is for all Type metadata contained within that level and all metadata Types contained within the requested Type. This means that for a metadata-id of METADATA-SYSTEM, for example, the server is expected to return *all* metadata.

NOTE: The metadata-id for METADATA-SYSTEM and METADATA-RESOURCE must be 0 or *.

12.3 Optional Request Arguments

Format	= **COMPACT** | **STANDARD-XML** | **STANDARD-XML:***version*
version	::= <RETS metadata public identifier>

"COMPACT" means a table descriptor, field list <COLUMNS> followed by a delimited set of the data fields. See Section 11 for more information on the COMPACT formats. "STANDARD-XML" means an XML presentation of the data in the format defined by the RETS Metadata XML DTD. Servers MUST support all formats. If the format is not specified, the STANDARD-XML presentation will be returned.

When the client requests the STANDARD-XML representation, it may also specify the public identifier of the DTD that it expects. The server SHOULD be prepared to support at least the current version and the prior version.

12.4 Required Server Response Header Fields

In addition to the other Required Server Header Fields specified in Section 3.3 the following response header fields are required.

Content-Type	The media type of the underlying data. The server MUST return this field in all replies. This field MUST be set to the type of media returned.

Content-Type	::= **Content-Type** : *type* / *subtype*

Example: Content-Type: text/xml

12.5 Required Response Arguments

There are no required response arguments.

12.6 Optional Response Arguments

There are no optional response arguments.

12.7 Metadata Response Body Format

The body of the metadata response has the following format when replying to a request with the format set to "COMPACT":

```
<RETS 1*SP ReplyCode=quoted-reply-code 1*SP
ReplyText=quoted-string SP > CRLF
[metadata-segment]
[rets-status-tag]
</RETS> CRLF
```

metadata-segment ::= <A metadata segment as defined in Section 11 .>

The body of the metadata response has the following format when replying to a format request of "STANDARD-XML" data:

```
<?xml version="1.0" ?>
[doctype]
<RETS 1*SP ReplyCode=quoted-reply-code 1*SP ReplyText=quoted-string SP >
[XML-metadata-segment]
[rets-status-tag ]
</RETS> CRLF
```

doctype	::= **<!DOCTYPE RETS PUBLIC "-//RETS//DTD Metadata Content 1.7//EN">**

XML-metadata-segment	::= A metadata segment as defined by the RETS Metadata XML DTD.

NOTE:

RETS 1.7 requires all server responses to be well-formed XML, and additionally requires GetMetadata responses to be valid XML. In addition, RETS requires that clients parse server responses as XML, not as simple text streams. The response formats shown here are normative with respect to content, but not normative with respect to form. That is, servers are free to produce response XML in any format that complies with the W3C XML 1.0 recommendation, so long as it is valid with respect to the appropriate DTD. XML escaping of content is implied, as is XML processing of whitespace and line endings. See the W3C *XML Recommendation 1.0, Third Edition,* for full information on XML.

12.8 Reply Codes

Table 12-1 GetMetadata Reply Codes

Reply Code	Meaning
20500	Invalid Resource The request could not be understood due to an unknown resource.
20501	Invalid Type The request could not be understood due to an unknown metadata type.
20502	Invalid Identifier The identifier is not known inside the specified resource.
20503	No Metadata Found No matching metadata of the type requested was found.
20506	Unsupported Mimetype The server cannot return the metadata in any of the requested MIME types.
20507	Unauthorized Retrieval The metadata could not be retrieved because it requests metadata to which the supplied login does not grant access (e.g. Update Type data).
20508	Resource Unavailable The requested resource is currently unavailable.
20509	Metadata Unavailable The requested metadata is currently unavailable.
20510	Request Too Large Metadata could not be retrieved because a system limit was exceeded.
20511	Timeout The request timed out while executing.
20512	Too many outstanding requests The user has too many outstanding requests and new requests will not be accepted at this time.
20513	Miscellaneous error The server encountered an internal error.

20514	Requested DTD version unavailable. The client has requested the metadata in STANDARD-XML format using a DTD version that the server cannot provide.

SECTION 13 - Compact Data Format

Clients may choose to access data from a server in a “COMPACT” data format that does not use full XML representation. When a client requests information from a compliant server in “COMPACT” format, it will typically need to interpret the result by using the metadata that the server makes available.

- 13.1 Overall format
- 13.2 Decoded Format
- 13.3 Transmission standards

13.1 Overall format

Compact-format records are sequences of fields separated by delimiter. A tab character (an octet with a binary value of 9) is the default delimiter unless another is specified as part of the transaction. The sequence of fields MUST be described by a <COLUMNS> tag in the body of the message that carries the compressed records. No field may be omitted from the <DATA>; if the value of a particular field for some record is undefined, the value SHOULD be represented by two delimiters with no intervening space.

Compact records are enclosed within a <DATA> start tag and a </DATA> end tag.

13.2 Decoded Format

Compact-decoded format requires sending data in its most people-readable form. Coded data is data that is stored as an enumeration, multivalue, boolean, abbreviation, or arbitrary string with its meaning defined elsewhere in the system. At minimum, a server SHOULD perform the lookup or expansion from the lookup or validation-lookup values defined in the metadata for that field but it MAY be a richer value provided by the system's reporting capabilities. If the field is multivalued, commas and a space separate the decoded values.

13.3 Transmission standards

A client or server transmitting a compact record MUST encode the data according to Table 13-1.

Table 13-1 Compact Data Format Representation

Type	Encoding Format
Numeric	An optional sign, followed by zero or more digits, followed by an optional period, followed optionally by zero or more digits. A valid number MUST contain at least one digit if it includes a decimal point or sign. The value may contain leading zeros before the period and/or trailing zeros after the decimal point and fraction, if any. Data types Tiny, Small, Int and Long (Table 11-9) may be signed but may not have nonzero digits after the decimal point. Values with the interpretation LookupBitmask must not be signed, nor may they have nonzero digits after the decimal point.
Character	The plain character sequence, except for LookupMulti, which contains multiple sequences of characters separated by commas. Values with the interpretation LookupBitstring must contain only the characters "0" and "1".
Date	Eight digits in YYYY-MM-DD order, with dashes separating the year from the month and the month from the day.

Time	Six digits in hh:mm:ss[.sss], with colons separating the hour from the minute and the minute from the second, with a three-digit optional fractions of a second format separated from the seconds with a decimal <".">.
Date-Time	A fourteen-digit string with separators as above, and a space between the day and the hour, as `YYYY-MM-DDThh:mm:ss[.sss],` with a three-digit optional fractions of a second separated from the seconds with a decimal <".">
MultiSelect	A string consisting of one or more substrings, comma-delimited, each of which corresponds to an entry in the field's associated MetadataLookup table.
Boolean	A single character, either 1 for true or 0 for false.

SECTION 14 - Session Protocol

A RETS session follows a well-defined timing sequence in becoming established and in terminating. In particular, the authorization sequence MUST be followed in order to begin using other transactions within the protocol. The protocol contains four phases: connection establishment, authorization, session and termination.

- 14.1 Connection Establishment
- 14.2 Authorization
- 14.3 Session
- 14.4 Termination

14.1 Connection Establishment

A client initiates communication with a server by beginning a TCP connection on any mutually agreed TCP port, with the default being 6103 for unencrypted connections, and port 12109 for SSL-encrypted connections. When the TCP connection has entered the Established state, the session proceeds to the start of the Authorization phase.

14.2 Authorization

Authorization begins when the client sends the server a Login transaction. The Login transaction contains the basic information that the server requires in order to start an authorization decision: the user ID and optionally, some information about the client software.

A server responds to the Login request by sending back a "401 Unauthorized" status code and a WWW-Authenticate header. This is part of an authentication challenge to the client. Part of the WWW-Authenticate header may contain a checksum (nonce) of a concatenation of the following:

1. The client-IP.
2. The server-supplied timestamp.
3. The server's private-key.

Server implementers should note that because of intervening proxy servers, the client IP address may change from connection to connection.

The client concatenates the nonce to the checksum of the Request-URI; then performs an MD5 digest using a concatenation of the username, realm and password as the secret. This result is then returned to the server as part of an Authorization header. The server MUST then compute the equivalent function using its own stored copy of the user's password. If the two match and the nonce is the same, the user is considered authenticated, and the login can proceed with the server informing the client of the available capabilities. The login has been accomplished without actually sending the password. A server MAY provide an anonymous login. A client wishing an anonymous login sends an empty Authentication field in its Login transaction, after which the authorization proceeds as before.

14.3 Session

Once the Authorization phase has been completed, both endpoints enter the Session phase. During the Session phase, clients may issue any combination of requests for which they are authorized. The first of these MUST be to issue a GET requests for the "Action" URL, if any, included in the Login response (Section 4.10). After this, clients may issue other

transactions.

Clients MAY issue multiple transactions without waiting for responses. However, servers are not required to process these requests in parallel, nor are servers required to complete the requests in the order in which they were issued. If a client issues a request before receiving a response to some earlier request, the client MUST be prepared to receive the responses in any order. The only way for a client to guarantee sequential execution of requests on every server is to wait for a response to any outstanding request before issuing a new request.

14.4 Termination

A client SHOULD initiate termination of the session by sending a Logoff transaction. If a server receives a Logoff transaction while other operations are pending, it SHOULD abort those pending operations. However, a server MUST NOT rely on receiving a Logoff transaction in order to terminate a session, due to the possibility of communications problems preventing the transmission of the Logoff transaction by the client.

Servers SHOULD provide a timeout mechanism, and if they do, MUST inform the client of the timeout interval during the Login transaction (Section 4.7).

SECTION 15 - ServerInformation Transaction

The ServerInformation transaction allows retrieving global information about a server, or dynamic information about resources offered by a server.

- 15.1 Required Request Arguments
- 15.2 Optional Request Arguments
- 15.3 Response Format
- 15.4 Well-known names
- 15.5 Reply Codes

15.1 Required Request Arguments

There are no required request arguments. A ServerInformation transaction with no request arguments requests global information.

15.2 Optional Request Arguments

`Resource`	The name of the resource for which dynamic information is requested. This is interpreted as a SystemName unless the `StandardNames` argument is present and nonzero.
`Class`	The name of the class within the resource for which dynamic information is requested. This is interpreted as a SystemName unless the `StandardNames` argument is present and nonzero.
`StandardNames`	A numeric value which, if zero, indicates that Resource and Class are both SystemName values, and which, if equal to 1, indicates that the Resource and Class names are both StandardName values.

15.3 Response Format

The response to the ServerInformation transaction is a well-formed XML document:

```
<RETS ReplyCode="replycode" ReplyText="replytext">
<ServerInformation>
<Parameter name="parametername" [resource="resourceID"
[class="classID"]]>
  value
</Parameter>
</ServerInformation>
</RETS>
```

NOTE:

> RETS 1.7 requires all server responses to be well-formed XML, and additionally requires ServerInformation transaction responses to be valid XML. In addition, RETS requires that clients parse server responses as XML, not as simple text streams. The response formats shown here are normative with respect to content, but not normative with respect to form. That is, servers are free to produce response XML in any format that complies with the W3C XML 1.0 recommendation, so long as it is valid with respect to the appropriate DTD. XML escaping of content is implied. See the W3C *XML Recommendation 1.0, Third Edition*, for full information on XML.

The server MUST supply the information that applies to the Class level even if the information is global to the system. That is, the client is not required to infer information from the class hierarchy.

The well-known names for parameters are given in Table 15-1.

15.4 Well-known names

Table 15-1 lists the well-known names for parameters defined in this specification. Servers may extend this list, but MUST precede their parameter names with the string "X-".

Table 15-1 Well-Known Parameter Names

Parameter	Level	Type	Description
`CurrentTimeStamp`	System	DateTime	The current system date and time, including the server time zone, in ISO 8601 format.
`LastTimeStamp`	ResourceClass	DateTime	The most recent modification timestamp of any record in the given resource and class, in ISO 8601 format.
`MinimumLimit`	ResourceClass	Numeric/String	The minimum `Limit` value for any search in this class. the value **`NONE`** may be returned if there is no minimum limit.
`KeyLimit`	ResourceClass	Numeric/String	The minimum `Limit` for any search in this class that includes a Key optional parameter. the value **`NONE`** may be returned if there is no minimum limit.

ReplicationSupport	Resource/Class	Character	An indication of the level of replication support available for the given resource/class: **N** indicates that replication is not supported for this resource/class. **Y** indicates that replication is supported, that the server supports the optional Key search argument, and that all fields are marked as to their controlling timestamp or foreign key. A blank query may be used to retrieve all records that the user is permitted to access. **K** indicates that replications is supported, and that the server supports the optional Key search argument. A query MUST contain one or more of the fields marked in the metadata with the KeyQuery flag.

15.5 Reply Codes

Table 15-2 ServerInformation Reply Codes

Reply Code	Meaning
0	Operation successful.
20601	Not supported. The transaction is not supported for the given resource and class.
20602	Miscellaneous error. The transaction could not be completed. The ReplyText gives additional information.

SECTION 16 - Acknowledgements

The creation of this specification would not have been possible without the sponsorship and coordination of efforts provided by the National Association of REALTORS®.

This document has benefited greatly from the comments of all those participating in the National Association of REALTORS®-Standards Work Group.

In addition to the authors, valuable discussion instrumental in creating this document has come from:

Richard Mendenhall
National Association of REALTORS®

Dale Stinton
National Association of REALTORS®

Larry Colson
Moore Data Management Services

Tom Curtis

Metro MLS

Kevin Knoepp
GTE Enterprise Solutions

Tom McLean
Resolution Software Consulting, Inc.

Tony Salvati
Grant Thornton

Errol Samuelson
RealSelect, Inc.

Allan Shapiro
Wantao Zhou
Interealty Corporation

Stuart Schuessler
MarketLinx Corporation

Michael DelGaudio
MRIS, Inc.

Mark Lesswing
National Association of Realtors®

Maggie Diaz
Brita Brodin
Laure Chipman
WyldFyre, Inc.

Joshua Vosper
Rapattoni Corporation
Laila Sharshar
NewportWorks, Inc.

Eric Schlosser
Hewlett-Packard Company

Frank Tadman
RE Infolink

SECTION 17 - Authors

Leo Bijnagte
Vista Information Systems
100 Washington Square, Suite 1000
Minneapolis, MN 55401

Email: leob@fnis.com

Dan Musso
WyldFyre Technologies, Inc.
900 East Hamilton Ave.
Suite 500
Campbell, CA 95008

Email: dan@WyldFyre.com

Bruce Toback
OPT, Inc.
11801 N. Tatum Blvd.
Suite 142
Phoenix, AZ 85028

Email: btoback@optc.com

SECTION 18 - References

- Appendix A - DTD References
- Appendix B - Sample Compact Metadata Response
- Appendix C - Summary of RETS Reply Codes
- Appendix D - Interpretation of Maximum Field Length and Display Information

Number	Reference
[1]	Braden, R., “Requirements for Internet Hosts — Communication Layers” STD 3, RFC 1123, IETF 1989.
[2]	Fielding, R., “Hypertext Transfer Protocol — Version 1.1”, RFC 2616, January 1997
[3]	Rivest, R., “The MD5 Message Authentication Algorithm”, RFC 1321, April 1992
[4]	Crocker, D., “Standard for ARPA Internet Text Messages”, RFC 2822, IETF 2001
[5]	US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.
[6]	Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E., and L. Stewart, “An Extension to HTTP: Digest Access Authentication”, RFC 2617, January 1997.
[7]	International Organization for Standards, “Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times”, ISO 8601, June 1988.
[8]	Borenstein, N., Freed, F., “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies”, RFC 2045, November 1996.
[9]	American National Standard for Data Encryption Algorithm (DEA). Standard ANSI X3.92, ANSI, 1981.
[10]	Data Encryption Standard, FIPS46-2, December 30, 1993.
[11]	DES Modes of Operation, FIPS81, December 2, 1980
[12]	IEEE/ANSI Std. 1003.2-1992, Information Technology – Portable Operating System Interface (POSIX®) Part 2
[13]	Berners-Lee et al., “Uniform Resource Identifiers (URI): Generic Syntax”, RFC 2396, IETF 1998
[14]	Kaliski, “PKCS #7: Cryptographic Message Syntax Version 1.5”, RFC 2315, IETF 1998
[15]	Kristol, D. and Montulli, L., “HTTP State Management Mechanism”, RFC 2109, IETF 1997
[16]	W3C, “HTML 4.01 Specification”, W3C Recommendation 24 December 1999 (http://www.w3.org/TR/html401/)
[17]	W3C, “Extensible Markup Language (XML) 1.0 (Third Edition)”, W3C Recommendation 4 February 2004 (http://www.w3.org/TR/2004/REC-xml-20040204/

[18]	Rescorla, E., “HTTP Over TLS”, RFC 2818, May 2000

Appendix A - DTD References

Table A-1 DTD References

Real Estate Transaction Standard Data DTD	
Description	The document returned by a search specifying STANDARD-XML format. This DTD describes the document only, not the entire response. It may be used when transmitting listing or membership data through a channel other than a RETS server (for example, FTP).
Public Identifier	`-//RETS//DTD RETS Data Content 1.7//EN`
System Identifier	`http://www.rets.org/dtd/2002/10/REData-20040915.dtd`
Real Estate Transaction Standard STANDARD-XML Search Response	
Description	The response returned by a search specifying STANDARD-XML format. This DTD simply encapsulates the REData DTD (above) in a standard RETS response element.
Public Identifier	`-//RETS//DTD RETS XML Search Response 1.7//EN`
System Identifier	`http://www.rets.org/dtd/2002/10/RETS-20040915.dtd`
Real Estate Transaction Standard COMPACT Search Response	
Description	The response returned by a search specifying COMPACT or COMPACT-DECODED format.
Public Identifier	`-//RETS//DTD RETS COMPACT Search Response 1.7//EN`
System Identifier	`http://www.rets.org/dtd/2004/09/rets-compact-search-1_7.dtd`
RETS Metadata Content DTD	
Description	This DTD describes the STANDARD-XML metadata format. It may be used when transmitting metadata through a channel other than a RETS server.
Public Identifier	{{-//RETS//DTD Metadata Content 1.7//EN
System Identifier	`http://www.rets.org/dtd/2004/09/rets-metadata-content-1_7.dtd`
RETS Metadata STANDARD-XML Response DTD	
Description	The document returned by a GetMetadata transaction specifying a format of STANDARD-XML. This encapsulates the RETS Metadata Content DTD in a standard RETS response element.

Public Identifier	-//RETS//DTD Metadata 1.7//EN
System Identifier	http://www.rets.org/dtd/2004/09/rets-metadata-1_7.dtd
RETS Metadata COMPACT format response	
Description	The document returned by a GetMetadata transaction specifying a format of COMPACT.
Public Identifier	{{-//RETS//DTD Compact Metadata 1.7//EN
System Identifier	{{http://www.rets.org/dtd/2004/09/rets-compact-metadata-1_7.dtd}}

Appendix B - Sample Compact Metadata Response

This appendix contains examples for COMPACT metadata responses. It is NON-NORMATIVE: these examples illustrate one way of formatting COMPACT metadata, and one set of values. Section 11 describes the content and formatting rules in detail.

B.1 System

```
<METADATA-SYSTEM Version="1.00.000" Date="Sat, 20 Mar 2002 12:03:38 GMT">
<SYSTEM SystemID= "NTREIS" SystemDescription= "North Texas Real Estate Information System"
/>
<COMMENTS>
This is a comment line
</COMMENTS>
</METADATA-SYSTEM>
```

B.2 Resource

```
<METADATA-RESOURCE Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ResourceIDStandardNameVisibleNameDescription
ClassCountKeyFieldClassVersionClassDateObjectVersion
ObjectDateSearchHelpVersionSearchHelpDateEditMaskVersion
EditMaskDate LookupVersionLookupDateUpdateHelpVersion
UpdateHelpDate ValidationExpressionVersion
ValidationExpressionDateValidationLookupVersion
ValidationLookupDateValidationExternalVersion
ValidationExternalDate</COLUMNS>
<DATA>AgentAgent AgentAgent Table1 Agentid1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>PropertyPropertyPropertyProperty Tables5
LN1.00.000 Sat, 20 Mar 2002 12:03:38 GMT1.00.000
Sat, 20 Mar 2002 12:03:38 GMT1.00.000
Sat, 20 Mar 2002 12:03:38 GMT 1.00.000
Sat, 20 Mar 2002 12:03:38 GMT 1.00.000
Sat, 20 Mar 2002 12:03:38 GMT 1.00.000
Sat, 20 Mar 2002 12:03:38 GMT 1.00.000
Sat, 20 Mar 2002 12:03:38 GMT 1.00.000
Sat, 20 Mar 2002 12:03:38 GMT 1.00.000
Sat, 20 Mar 2002 12:03:38 GMT </DATA>
<DATA>TaxTaxTaxMultimedia objects20 PID1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-RESOURCE>
```

B.3 Foreign Keys

```
<METADATA-FOREIGNKEYS Version="1.00.000000"
Date="Wed, 23 Jan 2002 12:37:38 GMT">
<COLUMNS>PARENT_RESOURCE_IDPARENT_CLASS_IDPARENT_SYSTEMNAME
CHILD_RESOURCE_IDCHILD_CLASS_IDCHILD_SYSTEMNAME</COLUMNS>
<DATA>PropertyRESMLSNUMTAXTAXMLSNUM</DATA>
<DATA>PropertyRESMLSNUMHistoryHistoryMLSNUM</DATA>
<DATA>PropertyRESMLSNUMOpenHouseOpenHouseMLSNUM</DATA>
<DATA>PropertyRESListingAgentIDAgentAgentAgentID</DATA>
<DATA>PropertyRESCOListingAgentIDAgentAgentAgentID</DATA>
<DATA>PropertyRESSellingAgentIDAgentAgentAgentID</DATA>
<DATA>PropertyRESCOSellingAgentIDvAgentAgentAgentID</DATA>
<DATA>PropertyRESListingOfficeIDOfficeOfficeOfficeID</DATA>
<DATA>PropertyRESSellingOfficeIDOfficeOfficeOfficeID</DATA>
</METADATA-FOREIGNKEYS>
```

B.4 Class

GetMetadata request:

```
Type: METADATA-CLASS
ID: 0
```

Compact reply:

```
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>MOBMobile HomeResidentialProperty
Mobile Homes1.00.000Sat, 20 Mar 2002 12:03:38 GMT
1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>LNDLots and LandLots and LandLots nd Land
1.00.000Sat, 20 Mar 2002 12:03:38 GMT1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-CLASS>
<METADATA-CLASS Resource="Agent" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" />
<COLUMNS>ClassNameVisibleNameStandardNameDescription
TableVersionTableDateUpdateVersion UpdateDate </COLUMNS>
<DATA>AgentAgentAgentAll Agents1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-CLASS>
```

B.5 Table

GetMetadata request:

```
Type: METADATA-TABLE
ID: Property: RES
```

Compact reply:

```
<METADATA-TABLE Resource="Property" Class="RES" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SystemNameStandardNameLongNameDBNameShortName
MaximumlengthDataTypePrecisionSearchableInterpretation
AlignmentUseSeparatorEditMaskIDLookupNameMaxSelectUnits
IndexMinimumMaximumDefaultRequiredSearchHelpID
MetadataEntryIDModTimeStampForeignKeyForeignFieldKeyQuery
KeySelect</COLUMNS>
<DATA>LNListIDListing IDLNListID8Int01
NumberLeft011</DATA>
<DATA>PTYPPropTypeProperty TypePTProp Type
2Int01NumberLeft0</DATA>
<DATA>LPListPriceList PriceLPLst Pr8Int01
CurrencyRight1142</DATA>
<DATA>OWNOwnerOwner NameOWNOwn Name20Character
00Left0</DATA>
<DATA>VEWViewViewVEWView10Long01LookupBitmaskLeft
0VEW1</DATA>
<DATA>EFExtFeatFeaturesEFExt Feat10Character01
LookupMultiLeft0EFT2</DATA>
<DATA>SDSchDistSchool DistrictSDSchDist10Character
01LookupLeft0SD</DATA>
<DATA>ARMLSAreaMLS AreaARArea4Int01LookupLeft
0AR3031</DATA>
</METADATA-TABLE>
```

B.6 Update

GetMetadata request:

```
Type:METADATA_UPDATE
ID: Property: RES
```

Compact reply:

```
<METADATA-UPDATE Resource="Property" Class="RES" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS> UpdateNameDescriptionKeyFieldVersionDate
MetadataEntryID</COLUMNS>
<DATA>AddAdd a new Residential Listing1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>ChangeChange a Residential ListingListNumber1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>BOMPut a Residential Listing Back on Market ListNumber
1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-UPDATE>
```

B.7 Update Type

GetMetadata request:

```
Type: METADATA-UPDATE_TYPE
ID: Property: RES: Add
```

Compact reply:

```
<METADATA-UPDATE_TYPE Resource="Property" Class="RES" Update="Add"
Version="1.00.000" Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SystemNameSequenceAttributesDefault
ValidationExpressionIDUpdateHelpIDValidationLookupName
ValidationExternalNameMetadataEntryIDMaxUpdate</COLUMNS>
<DATA>STNUM12StNumHelp</DATA>
<DATA>STNAME22StreetName</DATA>
<DATA>LD32ListDateDateHelp</DATA>
<DATA>LISTOFF42,3</DATA>
</METADATA-UPDATE_TYPE>
```

B.8 Object

GetMetadata request:

```
Class:METADATA-OBJECT
ID:0
```

Compact reply:

```
<METADATA-OBJECT Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ObjectTypeStandardNameVisibleNameDescription
MetadataEntryIDMIMETypeObjectTimeStampObjectCount</COLUMNS>
<DATA>PhotoimageFull PhotosHigh Resolution Property Photos
1image/jpegPhotoTimestapPhotoCount</DATA>
<DATA>ThumbnailimageSmall PhotosLow Resolution Property Photos
1image/jpegPhotoTimestapPhotoCount</DATA>
</METADATA-OBJECT>
```

B.9 Lookup

GetMetadata request:

```
Type: METADATA-LOOKUP
ID: 0
```

Compact reply:

```
<METADATA-LOOKUP Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>LookupNameVisibleNameVersionDateMetadataEntryID</COLUMNS>
<DATA>1Status1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>2Phone Type1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-LOOKUP>
<METADATA-LOOKUP Resource="Agent" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>LookupNameVisibleNameVersionDateMetadataEntryID</COLUMNS>
<DATA>1Status1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-LOOKUP>
```

B.10 Lookup Type

GetMetadata request:

```
Type: METADATA-LOOKUP_TYPE
ID: *
```

Compact reply:

```
<METADATA-LOOKUP_TYPE Resource="Property" Lookup="AR" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT">
><COLUMNS>LongValueShortValueValueMetadataEntryID</COLUMNS>
<DATA>Capitol HillCap Hill1</DATA>
<DATA>Juanita HillJuanita2</DATA>
<DATA>Maple ValleyMpl Valley3</DATA>
<DATA>Downtown RedmondDntn Rdmd<4></DATA>
</METADATA-LOOKUP_TYPE>
<METADATA-LOOKUP_TYPE Resource="Agent" Lookup="STAT" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>LongValueShortValueValueMetadataEntryID</COLUMNS>
<DATA>Active ACT1</DATA>
<DATA>SuspendedSUS2</DATA>
<DATA>InactvieINA3</DATA>
</METADATA-LOOKUP_TYPE>
```

B.11 Search Help

GetMetadata request:

```
Type: METADATA-SEARCH_HELP
ID: Property
```

Compact reply:

```
<METADATA-SEARCH_HELP Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SearchHelpIDValueMetadataEntryID</COLUMNS>
<DATA>1Enter the number in the following format dxd</DATA>
<DATA>2Enter the number in the following format d.dd</DATA>
</METADATA-SEARCH_HELP>
```

B.12 Edit Mask

GetMetadata request:

```
Type: METADATA-EDITMASK
ID: Property
```

Compact reply:

```
<METADATA-EDITMASK Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>EditMaskIDValueMetadataEntryID</COLUMNS>
<DATA>1[0-9]{1,2}[x][0-9]{1,2} </DATA>
<DATA>2[0-9]{3}[0-9]{2}[0-9}{4} </DATA>
</METADATA-EDITMASK>
```

B.13 Update Help

GetMetadata request:

```
Type: UPDATE_HELP
ID: Property
```

Compact reply:

```
<METADATA-UPDATE_HELP Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>UpdateHelpIDValueMetadataEntryID</COLUMNS>
<DATA>1Enter the number in the following format dxd</DATA>
<DATA>2Enter the number in the following format d.dd</DATA>
</METADATA-UPDATE_HELP>
```

B.14 Validation Lookup

GetMetadata request:

```
Type: METADATA-VALIDATION_LOOKUP
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_LOOKUP Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidationLookupNameParent1Field Parent2Field
VersionDateMetadataEntryID</COLUMNS>
<DATA>SchoolAreaSubarea1.00.000Sat, 20 Mar 2002 12:03:38 GMT
</DATA>
<DATA>ZipCodeArea1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
<DATA>City1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
</METADATA-VALIDATION_LOOKUP>
```

B.15 Validation Lookup Type

GetMetadata request:

```
Type: METADATA-VALIDATION_LOOKUP_TYPE
ID: Property: School
```

Compact reply:

```
<METADATA-VALIDATION_LOOKUP_TYPE Resource="Property"
ValidationLookup="School" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidTextParent1Value Parent2ValueMetadataEntryID</COLUMNS>
<DATA>133AREA1SUBAREA1</DATA>
<DATA>134AREA1SUBAREA2</DATA>
<DATA>135AREA2</DATA>
</METADATA-VALIDATION_LOOKUP_TYPE>
```

B.16 Validation Expression

GetMetadata request:

```
Type: METADATA-VALIDATION_EXPRESSION
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_EXPRESSION Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidationExpressionIDValidationExpressionTypeValue
MetadataEntryID</COLUMNS>
<DATA>Office1ACCEPT>
LAG=.AGENTCODE. .OR. (LO=.BROKERCODE. .AND. .ENTRY.=OFFICE)</DATA>
<DATA>Agent1ACCEPT(LAG=.AGENTCODE.) .OR. (SAG=.AGENTCODE.)</DATA>
<DATA>ListDateACCEPT LD>.TODAY. - 3 .AND. LD<.TODAY. + 3</DATA>
</METADATA-VALIDATION_EXPRESSION>
```

B.17 Validation External

GetMetadata request:

```
Type: METADATA-VALIDATION_EXTERNAL
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_EXTERNAL Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidationExternalNameSearchResourceSearchClassVersionDate
MetadataEntryID</COLUMNS>
<DATA>1Office Office1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
<DATA>2TaxHENN1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
</METADATA-VALIDATION_EXTERNAL>
```

B.18 Validation External Type

GetMetadata request:

```
Type: METADATA-VALIDATION_EXTERNAL_TYPE
ID: Property: VET1
```

Compact reply:

```
<METADATA-VALIDATION_EXTERNAL_TYPE Resource="Property"
ValidationExternal="VET1" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SearchFieldDisplayFieldResultsFieldsMetadataEntryID
</COLUMNS>
<DATA>AgentID, AgentCodeAgentName, OfficeNameSaleAgentID=AgentID,
SaleAgentName=AgentName, SaleOfficeID=OfficeID,
SaleOfficeName=OfficeName</DATA>
</METADATA-VALIDATION_EXTERNAL_TYPE>
```

Appendix C - Summary of RETS Reply Codes

Table C-1 Consolidated list of RETS reply codes

Reply Code	Meaning
0	Operation successful

10000	System error The server has detected an error with the request that prevents it from identifying the type of request, or that prevents the server from routing the request for processing. This return code MUST NOT be used when a more specific return code can be determined.
20003	Zero Balance The user has zero balance left in their account.
20004 thru 20011	RESERVED
20012	Broker Code Required The user belongs to multiple broker codes and one must be supplied as part of the login. The broker list is sent back to the client as part of the login response (see section 4.6).
20013	Broker Code Invalid The Broker Code sent by the client is not valid or not valid for the user
20014 thru 20019	RESERVED
20022	Additional login not permitted There is already a user logged in with this user name, and this server does not permit multiple logins.
20036	Miscellaneous server login error The quoted-string of the body-start-line contains text that SHOULD be displayed to the user
20037	Client authentication failed. The server requires the use of a client password (section 4.1.2), and the client either did not supply the correct client password or did not properly compute its challenge response value.
20050	Server Temporarily Disabled The server is temporarily offline. The user should try again later
20140	Insecure password. The password does not meet the site's rules for password security.
20141	Same as Previous Password. The new password is the same as the old one.
20142	The encrypted user name was invalid.
20200	Unknown Query Field The query could not be understood due to an unknown field name.
20201	No Records Found No matching records were found.
20202	Invalid Select The Select statement contains field names that are not recognized by the server.
20203	Miscellaneous Search Error The quoted-string of the body-start-line contains text that MAY be displayed to the user.

20206	Invalid Query Syntax The query could not be understood due to a syntax error.
20207	Unauthorized Query The query could not be executed because it refers to a field to which the supplied login does not grant access.
20208	Maximum Records Exceeded Operation successful, but all of the records have not been returned. This reply code indicates that the maximum records allowed to be returned by the server have been exceeded. Note: reaching/exceeding the "Limit" value in the client request is not a cause for the server to generate this error.
20209	Timeout The request timed out while executing
20210	Too many outstanding queries The user has too many outstanding queries and new queries will not be accepted at this time.
20211	Query too complex The query is too complex to be processed. For example, the query contains too many nesting levels or too many values for a lookup field.
20212	Invalid key request The transaction does not meet the server's requirements for the use of the `Key` option.
20213	Invalid `Key` The transaction uses a key that is incorrect or is no longer valid. Servers are not required to detect all possible invalid key values.
20301	Invalid parameter. Additional information is provided in the error block.
20302	Unable to save record on server.
20303	Miscellaneous Update Error.
20311	`Warning Response` was not given for all warnings that contained a {{*response-required*}}value of 2.
20312	`Warning Response` was given for a warning that contained a *`response-required`* value of 0.
20400	Invalid Resource The request could not be understood due to an unknown resource.
20401	Invalid Type The request could not be understood due to an unknown object type for the resource.
20402	Invalid Identifier The identifier does not match the KeyField of any data in the resource.
20403	No Object Found No matching object was found to satisfy the request.

20406	Unsupported MIME type The server cannot return the object in any of the requested MIME types.
20407	Unauthorized Retrieval The object could not be retrieved because it requests an object to which the supplied login does not grant access.
20408	Resource Unavailable The requested resource is currently unavailable.
20409	Object Unavailable The requested object is currently unavailable.
20410	Request Too Large No further objects will be retrieved because a system limit was exceeded.
20411	Timeout The request timed out while executing
20412	Too many outstanding requests The user has too many outstanding requests and new requests will not be accepted at this time.
20413	Miscellaneous error The server encountered an internal error.
20500	Invalid Resource The request could not be understood due to an unknown resource.
20501	Invalid Type The request could not be understood due to an unknown metadata type.
20502	Invalid Identifier The identifier is not known inside the specified resource.
20503	No Metadata Found No matching metadata of the type requested was found.
20506	Unsupported Mimetype The server cannot return the metadata in any of the requested MIME types.
20507	Unauthorized Retrieval The metadata could not be retrieved because it requests metadata to which the supplied login does not grant access (e.g. Update Type data).
20508	Resource Unavailable The requested resource is currently unavailable.
20509	Metadata Unavailable The requested metadata is currently unavailable.
20510	Request Too Large Metadata could not be retrieved because a system limit was exceeded.
20511	Timeout The request timed out while executing.

20512	Too many outstanding requests The user has too many outstanding requests and new requests will not be accepted at this time.
20513	Miscellaneous error The server encountered an internal error.
20514	Requested DTD version unavailable. The client has requested the metadata in STANDARD-XML format using a DTD version that the server cannot provide.
20701	Not logged in The server did not detect an active login for the session in which the Logout transaction was submitted.
20702	Miscellaneous error. The transaction could not be completed. The ReplyText gives additional information.

Appendix D - Interpretation of Maximum Field Length and Display Information

This appendix contains examples for METADATA-TABLE Maximum Field Length and sample displays for various combinations of data types, interpretation and other attributes of a field. It is NON-NORMATIVE: these examples illustrate one case of calculating and formatting field values and their metadata and one set of values. Section 11.3.2 describes the rules in detail.

D.1 Datatype Boolean

Interpretation	Precision	Separator	Units	Max Select	Extreme Example	Maximum Length	Display
null	n/a	n/a	n/a	n/a	0	1	False
null	n/a	n/a	n/a	n/a	1	1	True
Lookup	n/a	n/a	n/a	n/a	12345678	8	from lookup LookupName, longvalue, lookup shortvalue, the value from the corresponding lookup values, from value, 0 or 1

D.2 Datatype Character

Interpretation	Precision	Separator	Units	Max Select	Extreme Example	Maximum Length	Display
null	n/a	n/a	n/a	n/a	random_string	13	random_string
Lookup	n/a	n/a	n/a	n/a	random_string	13	from lookup LookupName, True

D.3 Datatype Decimal

Interpretation	Precision	Separator	Units	Max Select	Extreme Example	Maximum Length	Display
Numeric	2	,	null	n/a	-12342.21	9	-12,342.21
Numeric	1	,	Feet	n/a	123.1	5	123.1 feet
Currency	2	,	n/a	n/a	1246.227	7	$1246.22

D.4 Class

GetMetadata request:

```
Type: METADATA-CLASS
ID: 0
```

Compact reply:

```
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>MOBMobile HomeResidentialProperty
Mobile Homes1.00.000Sat, 20 Mar 2002 12:03:38 GMT
1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>LNDLots and LandLots and LandLots nd Land
1.00.000Sat, 20 Mar 2002 12:03:38 GMT1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-CLASS>
<METADATA-CLASS Resource="Agent" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" />
<COLUMNS>ClassNameVisibleNameStandardNameDescription
TableVersionTableDateUpdateVersion UpdateDate </COLUMNS>
<DATA>AgentAgentAgentAll Agents1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-CLASS>
```

D.5 Table

GetMetadata request:

```
Type: METADATA-TABLE
ID: Property: RES
```

Compact reply:

```
<METADATA-TABLE Resource="Property" Class="RES" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SystemNameStandardNameLongNameDBNameShortName
MaximumlengthDataTypePrecisionSearchableInterpretation
AlignmentUseSeparatorEditMaskIDLookupNameMaxSelectUnits
IndexMinimumMaximumDefaultRequiredSearchHelpID
MetadataEntryIDModTimeStampForeignKeyForeignFieldKeyQuery
KeySelect</COLUMNS>
<DATA>LNListIDListing IDLNListID8Int01
NumberLeft011</DATA>
<DATA>PTYPPropTypeProperty TypePTProp Type
2Int01NumberLeft0</DATA>
<DATA>LPListPriceList PriceLPLst Pr8Int01
CurrencyRight1142</DATA>
<DATA>OWNOwnerOwner NameOWNOwn Name20Character
00Left0</DATA>
<DATA>VEWViewViewVEWView10Long01LookupBitmaskLeft
0VEW1</DATA>
<DATA>EFExtFeatFeaturesEFExt Feat10Character01
LookupMultiLeft0EFT2</DATA>
<DATA>SDSchDistSchool DistrictSDSchDist10Character
01LookupLeft0SD</DATA>
<DATA>ARMLSAreaMLS AreaARArea4Int01LookupLeft
0AR3031</DATA>
</METADATA-TABLE>
```

D.6 Update

GetMetadata request:

```
Type:METADATA_UPDATE
ID: Property: RES
```

Compact reply:

```
<METADATA-UPDATE Resource="Property" Class="RES" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS> UpdateNameDescriptionKeyFieldVersionDate
MetadataEntryID</COLUMNS>
<DATA>AddAdd a new Residential Listing1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>ChangeChange a Residential ListingListNumber1.00.000
Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>BOMPut a Residential Listing Back on Market ListNumber
1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-UPDATE>
```

D.7 Update Type

GetMetadata request:

```
Type: METADATA-UPDATE_TYPE
ID: Property: RES: Add
```

Compact reply:

```
<METADATA-UPDATE_TYPE Resource="Property" Class="RES" Update="Add"
Version="1.00.000" Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SystemNameSequenceAttributesDefault
ValidationExpressionIDUpdateHelpIDValidationLookupName
ValidationExternalNameMetadataEntryIDMaxUpdate</COLUMNS>
<DATA>STNUM12StNumHelp</DATA>
<DATA>STNAME22StreetName</DATA>
<DATA>LD32ListDateDateHelp</DATA>
<DATA>LISTOFF42,3</DATA>
</METADATA-UPDATE_TYPE>
```

D.8 Object

GetMetadata request:

```
Class:METADATA-OBJECT
ID:0
```

Compact reply:

```
<METADATA-OBJECT Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ObjectTypeStandardNameVisibleNameDescription
MetadataEntryIDMIMETypeObjectTimeStampObjectCount</COLUMNS>
<DATA>PhotoimageFull PhotosHigh Resolution Property Photos
1image/jpegPhotoTimestapPhotoCount</DATA>
<DATA>ThumbnailimageSmall PhotosLow Resolution Property Photos
1image/jpegPhotoTimestapPhotoCount</DATA>
</METADATA-OBJECT>
```

D.9 Lookup

GetMetadata request:

```
Type: METADATA-LOOKUP
ID: 0
```

Compact reply:

```
<METADATA-LOOKUP Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>LookupNameVisibleNameVersionDateMetadataEntryID</COLUMNS>
<DATA>1Status1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
<DATA>2Phone Type1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-LOOKUP>
<METADATA-LOOKUP Resource="Agent" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>LookupNameVisibleNameVersionDateMetadataEntryID</COLUMNS>
<DATA>1Status1.00.000Sat, 20 Mar 2002 12:03:38 GMT</DATA>
</METADATA-LOOKUP>
```

D.10 Lookup Type

GetMetadata request:

```
Type: METADATA-LOOKUP_TYPE
ID: *
```

Compact reply:

```
<METADATA-LOOKUP_TYPE Resource="Property" Lookup="AR" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT">
><COLUMNS>LongValueShortValueValueMetadataEntryID</COLUMNS>
<DATA>Capitol HillCap Hill1</DATA>
<DATA>Juanita HillJuanita2</DATA>
<DATA>Maple ValleyMpl Valley3</DATA>
<DATA>Downtown RedmondDntn Rdmd<4></DATA>
</METADATA-LOOKUP_TYPE>
<METADATA-LOOKUP_TYPE Resource="Agent" Lookup="STAT" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>LongValueShortValueValueMetadataEntryID</COLUMNS>
<DATA>Active ACT1</DATA>
<DATA>SuspendedSUS2</DATA>
<DATA>InactvieINA3</DATA>
</METADATA-LOOKUP_TYPE>
```

D.11 Search Help

GetMetadata request:

```
Type: METADATA-SEARCH_HELP
ID: Property
```

Compact reply:

```
<METADATA-SEARCH_HELP Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SearchHelpIDValueMetadataEntryID</COLUMNS>
<DATA>1Enter the number in the following format dxd</DATA>
<DATA>2Enter the number in the following format d.dd</DATA>
</METADATA-SEARCH_HELP>
```

D.12 Edit Mask

GetMetadata request:

```
Type: METADATA-EDITMASK
ID: Property
```

Compact reply:

```
<METADATA-EDITMASK Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>EditMaskIDValueMetadataEntryID</COLUMNS>
<DATA>1[0-9]{1,2}[x][0-9]{1,2} </DATA>
<DATA>2[0-9]{3}[0-9]{2}[0-9}{4} </DATA>
</METADATA-EDITMASK>
```

D.13 Update Help

GetMetadata request:

```
Type: UPDATE_HELP
ID: Property
```

Compact reply:

```
<METADATA-UPDATE_HELP Resource="Property" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>UpdateHelpIDValueMetadataEntryID</COLUMNS>
<DATA>1Enter the number in the following format dxd</DATA>
<DATA>2Enter the number in the following format d.dd</DATA>
</METADATA-UPDATE_HELP>
```

D.14 Validation Lookup

GetMetadata request:

```
Type: METADATA-VALIDATION_LOOKUP
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_LOOKUP Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidationLookupNameParent1Field Parent2Field
VersionDateMetadataEntryID</COLUMNS>
<DATA>SchoolAreaSubarea1.00.000Sat, 20 Mar 2002 12:03:38 GMT
</DATA>
<DATA>ZipCodeArea1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
<DATA>City1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
</METADATA-VALIDATION_LOOKUP>
```

D.15 Validation Lookup Type

GetMetadata request:

```
Type: METADATA-VALIDATION_LOOKUP_TYPE
ID: Property: School
```

Compact reply:

```
<METADATA-VALIDATION_LOOKUP_TYPE Resource="Property"
ValidationLookup="School" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidTextParent1Value Parent2ValueMetadataEntryID</COLUMNS>
<DATA>133AREA1SUBAREA1</DATA>
<DATA>134AREA1SUBAREA2</DATA>
<DATA>135AREA2</DATA>
</METADATA-VALIDATION_LOOKUP_TYPE>
```

D.16 Validation Expression

GetMetadata request:

```
Type: METADATA-VALIDATION_EXPRESSION
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_EXPRESSION Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidationExpressionIDValidationExpressionTypeValue
MetadataEntryID</COLUMNS>
<DATA>Office1ACCEPT>
LAG=.AGENTCODE. .OR. (LO=.BROKERCODE. .AND. .ENTRY.=OFFICE)</DATA>
<DATA>Agent1ACCEPT(LAG=.AGENTCODE.) .OR. (SAG=.AGENTCODE.)</DATA>
<DATA>ListDateACCEPT LD>.TODAY. - 3 .AND. LD<.TODAY. + 3</DATA>
</METADATA-VALIDATION_EXPRESSION>
```

D.17 Validation External

GetMetadata request:

```
Type: METADATA-VALIDATION_EXTERNAL
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_EXTERNAL Resource="Property" Version="1.00.000"
Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>ValidationExternalNameSearchResourceSearchClassVersionDate
MetadataEntryID</COLUMNS>
<DATA>1Office Office1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
<DATA>2TaxHENN1.00.000Sat, 20 Mar 2002 12:03:38 GMT </DATA>
</METADATA-VALIDATION_EXTERNAL>
```

D.18 Validation External Type

GetMetadata request:

```
Type: METADATA-VALIDATION_EXTERNAL_TYPE
ID: Property: VET1
```

Compact reply:

```
<METADATA-VALIDATION_EXTERNAL_TYPE Resource="Property"
ValidationExternal="VET1" Version="1.00.000"
Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>SearchFieldDisplayFieldResultsFieldsMetadataEntryID
</COLUMNS>
<DATA>AgentID, AgentCodeAgentName, OfficeNameSaleAgentID=AgentID,
SaleAgentName=AgentName, SaleOfficeID=OfficeID,
SaleOfficeName=OfficeName</DATA>
</METADATA-VALIDATION_EXTERNAL_TYPE>
```