Real Estate Data Interchange Standard:
Real Estate Transaction Specification
(RETS)

Draft of May 11, 1999
Dan Musso

Table of Contents

1 Introduction

1.1 Purpose

This specification is the first part of a two part series defining the interchange of real estate information. The purpose of this document is to define a specification for the exchange of real estate property information, with the intent of eventually describing all interchangeable aspects of a real estate transaction. It defines a standard interface by which a client program may communicate with an property data server. The specification defines a protocol for implementing transactions, and incorporates an Extensible Markup Language (XML) specification for general purpose interchange. The specification also provides for a compressed data interchange format and specification to allow interchange of machine-interpretable property information. This second document, the Real Estate Transaction DTD defines the general structure of the XML DTD that can be used in transferring information from the server.

1.2 Certification

The National Association of REALTORS® (NAR) intends to implement a certification program under which a third-party product may be declared RETS Compliant, and include an NAR-designated logo on the client or server based product and any associated promotional materials.

1.3 Scope

This specification is intended to define only the minimum a product or service must do in order to earn a “Compliant” certification. 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 earn certification.

1.4 Requirements

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 receive certification.
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 certification 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 receive certification. 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."

1.5 Terminology

Arguments	Tag/value pairs passed to a transaction as part of the Argument-List. The tag and the value are separated by an equal sign ("=").
Argument-List	All the tag/value pairs for a request are Transfer-Encoded (see RFC 2068 [2]) and turned into a stream with each pair being separated by an ampersand (&) character. The tag/value stream is appended to the URI after a delimiting question mark (?) for the GET method. The tag/value stream is sent as the first entity body for the POST method. Transfer-Encoding MUST be used to indicate any transfer codings applied by an application to ensure safe and proper transfer of the Argument-List. The data and Arguments for a response are turned into a stream with each data record and Argument being separated by a carriage return/line feed.
Client	The system requesting data. This may well be another server seeking to update itself from the a server with a cross-presentation agreement. The specification does not assume any particular kind of client.
Endpoint	Either a server or client.
Metadata	The set of data that describes real estate data fields in detail.
Metadata Dictionary	The set of data that describes the available Metadata. This is basically the Meta-Metadata. It is used to determine the different classes of property. Not the fields within the property classes, but what property classes there are. It also defines what different types of searches are available (tax, open house, etc.).
Optional	A compliant server or client is not required to include any field designated as *optional*. The specification states the action to be taken by a compliant system in the absence of an optional field. The fact that the specification designate 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.
Server	The system providing data (also, referred to as the "host").
Session ID	A character string of up to 64 printable characters that uniquely identifies a session to a server. The contents are implementation-defined.
Request ID	A character string of up to 64 printable characters which uniquely identifies a request to a client. The contents are implementation-defined.

Standard-Name	The name of a real estate data field as it is known in the Real Estate Transaction XML DTD.
System-Name	The name of a real estate data field as it is known in the metadata.

2 Notational Conventions

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 822 [4].

2.2 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].

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

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)>

<"> = <US-ASCII double-quote mark (34)>

NULL = <no character>

CRLF = 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 CTLs, but including LWS>

tspecials = "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | <"> | "/" | "[" | "]" | "?" | "=" | "{" | "}" | SP | HT

token = 1*<any CHAR except CTLs or tspecials>

quoted-string = (<"> *(qdtext) <">)

qdtext = <any TEXT except <">>

3 Message Format

The Real Estate Transaction Specification is modeled on, and is compliant with HTTP/1.1. This specification does not require the implementation of persistent connections as defined in RFC 2068, however, it also does not preclude the use of them.

3.1 General Message Format

RETS messages consist of requests from a client to a server, answered by responses from the server and returned to the client. Both request and response use the generic message format of RFC 822, consisting of a start line, one or more header lines, an empty line, and zero or more body lines. The body may be null, depending on the message.

3.2 Header Field Format

A header field consists of three elements: a field name, a colon (":"), and a field value followed by a CRLF. The field value may be preceded or followed by any amount of LWS, though a single SP is preferred between the colon and the field value and no LWS is preferred after the field value; the LWS is not interpreted as part of the field value. The value itself may consist of any sequence of characters except CR or LF.

3.3 Required Server Response Header Fields

The 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 2068. Example: Date: Sat, 20 Mar 1999 12:03:38 GMT The date/time stamp MUST be represented in Greenwich Mean Time (GMT), without exception.
Cache-Control	The RFC 2068 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 2068. 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/plain". There are only two exceptions to this: (1) When the Search Transaction returns XML data the Content-Type should be set to "text/xml". (2) See Section 5.2 in the GetObject Transaction for the other exception and more information on this field. Example: "Content-Type: text/plain"
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 2068. 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/1.0

Applications sending request or response messages, as defined by this specification, MUST include a RETS-Version of "RETS/1.0". Use of this version number indicates that the sending application is at least conditionally compliant with this specification.

3.4 Required Client Request Header Fields

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

Accept

This is a standard HTTP header field as defined in RFC 2068. Except for the GetObject Transaction, this value should be set to "*/*".

Example: "Accept: */*"

See Section 5.1 for more information on this field.

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 2068.

User-Agent = "User-Agent" ":" SP product
product = token ["/" product-version]
product-version = token

Example: User-Agent: MLSWindows/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 2068.

A server MAY advertise additional capabilities based on the User-Agent, and MAY refuse to proceed with the authorization if valid User-Agent has not been supplied.

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 2068 and is required for all requests containing a message-body.

RETS-Version

The RETS-Version as defined in Section 3.3.

Cookie

The implementation of this specification is intended to create a stateless system however, because the user is required to login there are at least two states. The use of the Cookie is required to provide a mechanism that can ensure that there are not multiple simultaneous sessions with a single username/password, if required by the server, and also to provide an added level of security. A new RETS-Session-ID Cookie is issued by the server at Login (see Section 4.6). This MUST be saved by the client application and sent in the HTTP header as "Cookie: RETS-Session-ID=" to all subsequent requests. The RETS-Session-ID SHOULD be set to '0' for the initial Login.

```
Cookie                  = "RETS-Session-ID=" session-id
session-id              = 1*64ALPHANUM
```

Example: Cookie: RETS-Session-ID=AE872BC1DDFE7880DAD31233

3.5 Optional Server Response Header Fields

RETS-Request-ID	A character string of printable characters which uniquely identifies a request to a client. 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
Server	The server standard response-header field contains information about the software used to handle the request. The format of this field is the same as the User-Agent field in Section 3.4. Example: Server: Microsoft-IIS/4.0

3.6 Optional Client Request Header Fields

RETS-Request-ID	RETS-Request-ID as defined in Section 3.5

3.7 Request Format

The request takes two forms. The form used is dependent on the Method. For the POST Method the post-request form is used. For the GET Method the get-request form is used. The major difference between the two forms is the location of the Argument-List. 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.

```
post-request            = "POST" SP Request-URI SP HTTP-Version CRLF
                            *message-header
                            CRLF
                            [Argument-List]

get-request             = "GET" SP Request-URI [ "?" Argument-List] SP HTTP-Version CRLF
                            *message-header
                            CRLF
```

The Request-URI, HTTP-Version and message-header are defined in RFC 2068.

3.8 Response Format

The server response to a request includes a Status-Line, zero or more header-lines, a CRLF and a reply body. The Status-Line of a response consists of a status code and a (possibly empty) reason phrase.

server-reply = Status-Line
*(header-line)
CRLF
[body-start-line *(response-arguments) body-end-line]

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Status-Code = 1*4DIGIT

The list of allowable Status-Codes can be found in RFC 2068. The more useful Status-Codes are provided in Section 3.9. Servers MUST use an appropriate predefined status codes when communicating with the client. When an error is encountered a client MAY display both the status code and the associated Reason-Phrase in its communication with the user.

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.

Reason-Phrase = *<TEXT, excluding CR/LF>

body-start-line = "<RETS" 1*SP reply-code 1*SP quoted-string *SP ">" CRLF

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

response-arguments = ((tag "=" value) | data) CRLF

body-end-line = "</RETS" [1*SP end-reply-code 1*SP quoted-string *SP] ">" CRLF

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

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.

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 <tag>, <value> and <data> elements are defined in the Response Arguments section for each transaction.

An example server-reply:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Sat, 20 Mar 1999 12:03:38 GMT
Content-Type: text/plain
Cache-Control: private

<RETS 0 "SUCCESS">
Tag1=Value1
Tag2=Value2
</RETS>
```

3.9 General Status Codes

Any of the following status codes (in addition to the others provided in RFC 2068) may be returned by a server in response to any request:

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.

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. If there are no problems with the information a RETS-Session-ID cookie MAY be issued. If issued, the Session ID MUST be passed from the client application to the server in all subsequent transactions in the request header and is validated by the server before the user is allowed to access the service.

The login transaction also provides for a mechanism that servers MUST implement to handle the support of client software copy protection. The server itself is not required to implement the client software copy protection but it must provide a means for it to be implemented.

4.1 Security

The Login transaction uses a simple challenge-response method called Digest Authentication. Digest Authentication suffers from many known limitations. It does not provide a secure authentication mechanism and provides no encryption of object content. That is not the intent. It is intended solely to replace a much weaker and even more dangerous authentication mechanism: Basic Authentication. Unlike Basic Authentication, Digest Authentication can ensure that the password is never sent in the clear.

Needs for secure HTTP transactions cannot be met by Digest Authentication. For those needs SSL or SHTTP are more appropriate protocols.

For security reasons all servers SHOULD, at a minimum, implement the modified version of Digest Authentication in this specification. For more information on the Digest Authentication Scheme refer to RFC 2069 [6].

Refer to Section 12.2 for a simplified explanation of the authentication scheme.

4.2 Digest Algorithm

The algorithm used to generate the digest and checksum is MD5 [3]. In this specification the string obtained by applying the digest algorithm to the data "data" with secret "secret" will be denoted by KD(secret, data), and the string obtained by applying the checksum algorithm to the data "data" will be denoted H(data).

For the "MD5" algorithm

H(data) = MD5(data)

and

KD(secret, data) = H(concat(secret, ":", data))

i.e., the digest is the MD5 of the secret concatenated with a colon concatenated with the data.

4.3 WWW-Authentication Response Header

If the server receives a Login request and an acceptable Authorization header (as defined in Section 4.4) is not sent, the server MUST respond with a "401 Unauthorized" status code, and a WWW-Authenticate header.

WWW-Authenticate = "WWW-Authenticate" ":" "Digest" digest-challenge digest-challenge = 1#(realm | nonce | opaque)

realm = "realm" "=" realm-value

realm-value = quoted-string
A string to be displayed to users so that they know which username and password to use. An example might be Users@TheSite.com.

nonce = "nonce" "=" nonce-value

nonce-value = quoted-string
A server-specified data string which MAY be uniquely generated each time a "401 Unauthorized" response is made and MUST be returned by the client unchanged. This string SHOULD be base64 or hexadecimal data. The nonce SHOULD have the following format:

H(client-IP ":" time-stamp ":" private-key)

Where client-IP is the dotted quad IP address of the client making the request, time-stamp is a server-generated time value, private-key is data known only to the server.

With a nonce of this form a server would normally recalculate the nonce after receiving the client authentication header and reject the request if it did not match the nonce from that header. In this way the server can limit the reuse of a nonce to the IP address to which it was issued and limit the time of the nonce's validity.

opaque = "opaque" "=" opaque-value

opaque-value = quoted-string
A server-specified data string which MUST be returned by the client unchanged. This string SHOULD be base64 or hexadecimal data. The opaque-value is useful for transporting state information around.

4.4 The Authorization Request Header

When the client receives a "401 Unauthorized" status code it is expected to retry the request, passing an Authorization header, which is defined as follows.

Authorization = "Authorization" ":" "Digest" digest-response

digest-response = 1#(username | realm | nonce | digest-uri | response | opaque)

username = "username" "=" username-value

username-value = quoted-string

digest-uri = "uri" "=" digest-uri-value

digest-uri-value = request-uri ; As specified by HTTP/1.1

response = "response" "=" response-digest

response-digest = <"> *LHEX <">

The definitions of response-digest above indicate the encoding for their values. The following definitions show how the value is computed:

response-digest = <"> < KD (H(A1), unquoted nonce-value ":" H(A2) > <">

A1 = unquoted username-value ":" unquoted realm-value ":" password

password = < user's password >

A2 = Method ":" digest-uri-value

unquoted username-value = "quoted-string" as specified in Section 2.2 of RFC 2068. However, the surrounding quotation marks are removed in forming the string A1. Thus if the Authorization header includes the fields username="joesmith", realm=Users@TheSite.com and the user joesmith has password "SuperAgent" then H(A1) would be:
H(joesmith:Users@TheSite.com:SuperAgent)
with no quotation marks in the digested string.

Method = The HTTP request method as specified in Section 5.1 of RFC 2068.

request-uri = The Request-URI from the request line as specified in Section 5.1 of RFC 2068. This may be "*", an "absoluteURL" or an "abs-path" as specified in Section 5.1.2 of, but it MUST agree with the Request-URI. In particular, it MUST be an "absoluteURL" if the Request-URI is an "absoluteURL".

4.5 Authorization Example

The following example assumes that a client application is trying to access the Login URI on the server. The URI is "http://www.TheSite.com/login". Both client and server know that the username is "joesmith", and the password is "SuperAgent".

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@TheSite.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@TheSite.com",
              nonce="dcd98b7102dd2f0e8b11d0f600bfb0c0",
              opaque="5ccdef346870ab04ddfe0412367fccba",
              uri="/login",
              response="e966c932a9242554e42c8ee200cec7"
```

4.6 Optional Response Header Fields

In addition to the other Optional Server Response Header Fields specified in Section 3.5 the following response header field MAY be sent.

Set-Cookie The use of the Set-Cookie is required to provide a mechanism that, if required by the server, can guarantee that there are not multiple simultaneous sessions with a single username/password and also to provide an added level of security. A new RETS-Session-ID cookie is issued by the server at Login. This MUST be saved by the client application and sent in the HTTP header of any subsequent client requests during the session as "Cookie: RETS-Session-ID=".

Set-Cookie = "RETS-Session-ID=" session-id ";"
SP "path=/"
session-id = 1*64ALPHANUM

Example: Set-Cookie: RETS-Session-ID=AY872YOPOIPPOIP7880;
path=/

Any server implementations that do not require the use of Session IDs should set the session-id in the response to '0'.

4.7 Required Request Arguments

There are no required request arguments.

4.8 Optional Request Arguments

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*4ALPHANUM

4.9 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 reply-code 1*SP quoted-string *SP ">" CRLF
"</RETS [1*SP end-reply-code 1*SP quoted-string *SP] >" CRLF

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

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 reply-code 1*SP quoted-string *SP ">" CRLF
member-name-tag
user-info-tag
broker-tag
[office-list-tag]
[metadata-ver-tag]
[balance-tag]
[timeout-tag]
[pwd-expire-tag]
capability-url-list

"</RETS [1*SP end-reply-code 1*SP quoted-string *SP] >" CRLF

4.10 Required Response Arguments

broker-tag = "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*4ALPHANUM

member-name-tag = "MemberName" "=" member-name CRLF

The member's full name (display name) as it is to appear on any printed output.

member-name = 1*48TEXT

user-info-tag = "User" "=" user-id "," user-level "," user-class "," agent-code CRLF

This tag 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 be set the return value to NULL

user-id = 1*30ALPHANUM
user-class = 1*3ALPHANUM
user-level = 1*2DIGIT
agent-code = 1*14ALPHANUM

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, a value of NULL should be returned.

capability-url-list = see Section 4.12 for format information

The server MUST return a capability list that includes at least a Search URL. The server MAY in addition return any of the other types in Section 4.12. 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 4.12 based on the user-agent. If it does, it MUST follow the format specified in Section 4.12.

4.11 Optional Response Arguments

metadata-ver-tag = "MetadataVersion" "=" new-metadata-version CRLF

This is the new version of the metadata that is available on the server. This value is different than the metadata-version that was passed in the Arguments if there is a newer version available on the server

metadata-version = 1*2DIGITS "." 1*2DIGITS ["."

1*3DIGITS]

It uses a "<major>.<minor>.<release>" numbering scheme.

balance-tag = "Balance" "=" balance CRLF

If the server supports an active billing account then this value is the users balance on the system in dollars.

balance = 1*5DIGIT "." 2DIGIT

timeout-tag = "TimeoutSeconds" "=" 1*5DIGIT

The number of seconds during which a session will remain alive, after which the server will terminates the session automatically. 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-tag = "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.

office-list-tag = "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-tag is an enumeration of the offices they can login to.

broker-code = 1*24ALPHANUM
broker-branch = 1*4ALPHANUM

4.12 Resources

The capability-url-list is the set of functions or URLs to which the login grants access. A capability consists of a tag 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]
"Login" "=" login-URL CRLF
["LoginComplete" "=" login-complete-URL CRLF]
["Logout" "=" logout-URL CRLF]
"Search" "=" search-URL CRLF
["Update" "=" update-URL CRLF]

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.
Change-password-URL	A URL for the ChangePassword 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 it is different that the one currently stored on 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	A URL for the LoginComplete Transaction. This URL is intended to provide a means for allowing client application vendors to implement client software copy protection. This value may be User-Agent dependent.
Logout-URL	A URL for the Logout Transaction.
Search-URL	A URL for the Search Transaction.
Update-URL	A URL for the Update Transaction.

The table is extensible; servers may define additional transactions for clients to access. The tags 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

                                  user-agent-token = token portion of the user-agent (Section 3.4)
                                  function-name = 1*ALPHA

                                  Example:
                                  MLSWindows-special = URL
```

A compliant client need not recognize any transaction that is not included in this specification.

4.13 Reply Codes

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.

	<NOTE: Broker List is sent back to the client>
20013	Broker Code Invalid The Broker Code sent by the client is not valid or not valid for the user.
20014 thru 20019	RESERVED
20036	Miscellaneous server login error The quoted-string of the body-start-line contains text that MUST be displayed to the user.
20050	Server Temporarily Disabled The server is temporarily offline. The user should try again later.

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 as well as system metadata. Objects requested and returned from this transaction are requested and returned as MIME media types. Some servers MAY support multipart messages.

5.1 Required Client Request Header Fields

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     = "*" | "text" | "image" | "audio" | "video"
subtype  = "*" | <A publicly-defined extension token that
            has been registered with IANA>
parameter= "q" "=" < qvalue scale from 0 to 1 >
```

A compliant server MUST support at least text/plain, text/xml and image/jpeg. The more common <types> and <subtypes> are as follows:

text/plain
text/xml
text/html
video/mpeg
video/quicktime
image/gif
image/jpeg
image/tiff
audio/basic

A more complete list is available at:
ftp.isi.edu/in-notes/iana/assigments/media-types

The qvalue is used to specify the desirability of a given media type/format, 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 10.2.

5.2 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. 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 MAY 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 5.10 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
MIME-Version	All responses MUST include a MIME-Version of "1.0" in the response header. Example: MIME-Version: 1.0

5.3 Optional Server Response Header Fields

In addition to the other Optional Server Header Fields specified in Section 3.5 the following response header field are also optional.

Link	If the client has submitted a request with "Link=1" the header of the response MUST contain the Link header field. Link = "Link" ":" "<" URI ">" Example: Link: <http://www.TheSite.com/pic/123456.jpg>
Description	A text description of the object. Description = " Content-Description " ":" *64<TEXT, excluding CR/LF> Example: Content-Description: Front View

5.4 Required Request Arguments

Resource = <resource defined in the Metadata Dictionary (see Section 10.2.2)>

The resource from where the object should come is specified by this entry. For more information see Section 5.9.

ID = 1*ALPHANUM *("," 1*ALPHANUM)

The ID (e.g., MLS number, AgentID) for which the object is to be retrieved. Note: if multiple IDs are sent then the host MAY respond with a multipart MIME response. Not all servers will support multipart MIME responses.

5.5 Optional Request Arguments

Object (1*5DIGIT | ResourceID)

This entry is either an enumeration serial number or a ResourceID defined in the Metadata Dictionary when retrieving metadata (see Section 10.2.2). If omitted, the designated preferred object of the given type is returned. This parameter can be used to specify the photo number. For example, a value of "3" would indicate photo number 3.

Link "0" | "1"

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

5.6 Required Response Arguments

There are no required response arguments.

5.7 Optional Response Arguments

There are no optional response arguments.

5.8 Metadata

To retrieve metadata files the client must first retrieve the Metadata Dictionary, which is itself, a type of metadata. The Metadata Dictionary can be retrieved by specifying the following: Resource=METADATA and Object=0. Once the client application has retrieved the metadata, it can analyze it to determine what other metadata files (Object not equal to 0) it must retrieve. A full description of the Metadata Dictionary is provided in Section 10.

5.9 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 10-1 contains the list of well-known resource names.

5.10 Multipart Responses

In the case where the client has requested multiple IDs, the server MAY 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. The body must then contain 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.

The Content-Type field for multipart entities requires one parameter, "boundary". The boundary delimiter line is then defined as a line consisting entirely of two hyphen characters ("-", decimal value 45) followed by the boundary parameter value from the Content-Type header field, optional linear whitespace, and a terminating CRLF.

The CRLF preceding the boundary delimiter line is conceptually attached to the boundary so that it is possible to have a part that does not end with a CRLF (line break). Body parts that must be considered to end with line breaks, therefore, must have two CRLFs preceding the boundary delimiter line, the first of which is part of the preceding body part, and the second of which is part of the encapsulation boundary.

The boundary delimiter MUST NOT appear inside any of the encapsulated parts, on a line by itself or as the prefix of any line. It must be no longer than 70 characters, not counting the two leading hyphens. Because boundary delimiters must not appear in the body parts being encapsulated, a user agent must exercise care to choose a unique boundary parameter value. The boundary parameter value in the example above could have been the result of an algorithm designed to produce boundary delimiters with a very low probability of already existing in the data to be encapsulated without having to prescan the data.

The boundary delimiter line following the last body part is a distinguished delimiter that indicates that no further body parts will follow. Such a delimiter line is identical to the previous delimiter lines, with the addition of two more hyphens after the boundary parameter value.

Example:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Sat, 20 Mar 1999 12:03:38 GMT
Cache-Control: private
MIME-Version: 1.0
Content-type: multipart/parallel; boundary="simple boundary"

--simple boundary
Content-Type: image/jpeg
Content-ID: 123456
<binary data>

--simple boundary
Content-Type: image/jpeg
```

```
Content-ID: 123457
<binary data>

--simple boundary--
```

6 Logout Transaction

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

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

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 = 1*9DIGITS

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

Billing = *<TEXT, excluding CR/LF>

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

SignOffMessage = *<TEXT, excluding CR/LF>

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.

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 the following formats: "COMPACT", "COMPACT-DECODED" or "STANDARD-XML".

7.1 Search Types

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

ActiveAgent	An ActiveAgent Search is a search against the Agent/Member database/table. This search only returns active agents. These are agents that are currently authorized to access the server (paid-up, not retired, etc.)
Agent	An Agent Search is a search against the Agent/Member database/table. It is used for retrieving information about the agents.
History	A History Search is a search against the history database/table.
Office	An Office Search is a search against the office database/table. It is used for retrieving information about the offices.
OpenHouse	An OpenHouse Search is a search against the Open House database/table.
Property	A Property Search is a search against the property database/tables. If the server supports a cross-property search then the metadata will define a class to use to perform the cross-property search.
Prospect	A Prospect Search is a search against the Prospect database/tables. A Prospect Search is used to retrieve prospect information from the server database.
Tax	A Tax Search is a search against the public records database/tables. Many systems have multiple public record databases. Each public record database is assigned a class number. This class number is used by the client application when submitting a search to distinguish which database the search will be conducted against.
Tour	A Tour Search is a search against the Tour database/table.

7.2 Search Terminology

field-delimiter = HEX HEX

This is the 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.

field = SQLFIELDNAME

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.

field-data = *TEXT

Any valid data for a field.

record-count = 1*9DIGITS

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

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

7.3 Search Response Body Format

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 reply-code 1*SP quoted-string *SP ">" CRLF
[ count-tag ]
[ delimiter-tag ]
[ column-start-tag ]
[ column-end-tag ]
*( compact-data )
[ max-row-tag ]
"</RETS [1*SP end-reply-code 1*SP quoted-string *SP] >" CRLF
```

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

```
"<RETS" 1*SP reply-code 1*SP quoted-string *SP ">"
[ count-tag ]
*( XML-data-record )
[ max-row-tag ]
"</RETS [1*SP end-reply-code 1*SP quoted-string *SP] >" CRLF
```

7.4 Required Request Arguments

SearchType = "ActiveAgent" | "Agent" | "History" | "Office" | "OpenHouse" | "Prospect" | "Property" | "Tax" | "Tour"

The type of search to perform as discussed in Section 7.1 and defined in the Metadata Dictionary (see Section 10.2).

Class = Class = 1*4DIGITs

This parameter is set to a value that represents the class of data within the SearchType. The allowable numbers SHOULD be sequential starting with one ("1") for each SearchType, and are implementation-defined.

Query = The query used to retreive information from the server. The query is based in the language described in Section 7.8.

QueryType = An enumeration giving the language in which the query is presented. The only valid value for RETS/1.0 is “DMQL” which indicates the query language described in Section 7.8.

7.5 Optional Request Arguments

Count = "0" | "1" | "2"

If this parameter is set to one ("1"), then a record-count is returned in the response in addition to the data. Note that on most 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. If this entry is not present or set to zero ("0") there is no record count returned.

Format = "COMPACT " | "COMPACT-DECODED " | "STANDARD-XML"

"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 in a fully-decoded (people-readable) format. 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 Data XML DTD. Servers MUST support all formats.

Limit = "NONE" | 1*9DIGIT

If this entry is set to ("NONE") or is not present, the server should not enforce the standard download limit. The use of "NONE" MAY disable both the <MAXROWS> tag and return-code "20208 Maximum Records Exceeded". Client implementers should be aware that some server implementations might not honor the request to disable the limit.

Alternatively, if the entry is set to a number greater than '0', 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.

Offset = 1*9DIGIT

This entry 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 implementers 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.

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 MUST accept a query, which contains unknown fields in its select list; it is the client's responsibility to decide what to do if the server does not provide a requested field.

7.6 Required Response Arguments

There are no required response arguments.

7.7 Optional Response Arguments

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

If a "COMPACT" of “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 record-count 1*SP "/>" CRLF

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-start-tag = "<COLUMNS>" field-delimiter 1*(field field-delimiter) CRLF

If a "COMPACT" format is specified in the request then a "<COLUMNS>" tag, including a delimited list of all the fields of data being returned, is sent back in the response. The field-delimiter is determined by the delimiter-tag. Note: the "</COLUMNS> end tag is at the end of all the data.

column-end-tag = "</COLUMNS>" CRLF

A closing tag for the column-start-tag. If a "COMPACT" format is specified in the request and the column-start-tag was sent by the server then the column-end-tag MUST also be sent.

delimiter-tag = "<DELIMITER" 1*SP field-delimiter 1*SP "/>" CRLF

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-start-tag.

max-row-tag = "<MAXROWS/>" CRLF

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.

7.8 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.

```
Query               = 1*( "(" sub-query ")" *( "," "(" sub-query ")" ))

sub-query           = 1*( "(" field "=" field-value ")" *( "|" "(" field "=" field-value ")" ))

field-value         = lookup-list | range-list | string-list
lookup-list         = lookup-or | lookup-not | lookup-and
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 "?" *ALPHANUM
range-list          = 1*( range *( "," range ))
range               = between | greater | less
between             = ( date | 1*DIGIT) "-" ( date | 1*DIGIT)
greater             = ( date | 1*DIGIT) "+"
less                = ( date | 1*DIGIT) "-"
date                = (year "-" month "-" day) | TODAY
month               = 2DIGIT
day                 = 2DIGIT
year                = 4DIGIT

TODAY               = "TODAY" <the host should substitute this with today's date>
```

Note: All dates and times submitted in queries MUST be in GMT.

There are three types of field values that can be passed in the query string. They are a <lookup-list>, a <range> and a <string>. A <lookup-list> is a field that may only contain predefined values. "Status" & "Type" typically falls into this category. A <range> field is of type numeric or date. These fields can be searched based on a range of values. "ListPrice" and "ListDate" fall into this category. All values specified in a <range> field 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 & 3; 2- is the same as 2 or less, inclusive of 2). A <string> field is basically 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".

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.

This query language provides for a nesting of sub-queries. For example:
Query=((AREA=|1,2)|(CITY=ACTON)),(LP=200000+)

Query Example:

```
Query=(ST=|ACT,SOLD),
(LP=200000-350000),
(STR=RIVER*),
(STYLE=RANCH),
(EXT=+WTRFRNT,DOCK),
(LDATE=1999-03-01+),
(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) and (STR beginning with RIVER) and (STYLE equal RANCH) and (EXT equal WTRFRNT and DOCK) and (LDATE greater than or equal to 1999-03-01) 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)."

7.9 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

8 Get Transaction

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

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.

9 Update Transaction

This has not yet been defined.

10 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.

Note that this section of this specification uses <t> to show tabs (HT from Section 2.2) and all rows are terminated with a CRLF.

The values referenced in this Section 10 can be found in Sections 2.2 and 11.2.

10.1 Organization and Retrieval

Metadata is organized by table/object, with each table having its own unique set of metadata describing the fields available in that table. The client retrieves the metadata by using the GetObject Transaction specifying the object of interest as the Object field, and specifying METADATA as the Resource. The client MUST specify Object=0 to retrieve the Metadata Dictionary.

The server supplies the metadata as documents using the formats described in this section. The client MUST accept fields 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-Version field from each session in which it caches retrieved metadata, and MUST request new metadata each time the value of the metadata-version field changes.

10.2 Metadata Dictionary for Resources

Clients can determine the number and type of searchable entities by referencing the Searchable Resource Classes. 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 retrieved are defined in a document with the format defined in this section. There are four parts to the document. The first part provides version information, the second part describes the available resources, the third part describes the searchable classes, and the forth part describes the available multimedia types.

10.2.1 Well-Known Resource Names

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. The following table contains the list of well-known resource names.

Table 10-1 Well-known Resource Names

Resource Name	Purpose
"ActiveAgent"	A resource that contains information about active agents. These

Resource Name	Purpose
	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.
"METADATA"	The system resources.
"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.

10.2.2 Version

The Version section will start with a <METATA-VERSION> tag indicating the current version of the file. This tag is followed by a <TIMESTAMP> section, which contains the timestamp for the last time the document was modified. Finally, it is followed by a <COMMENTS> section. The version section has the following format:

```
"<METADATA-VERSION" SP "Version" "=" metadata-version *SP ">" CRLF
"<TIMESTAMP" SP "Date" "=" Date *SP "/>" CRLF
"<SYSTEM" SP "ShortName" "=" code-name SP "Name" "=" long-name *SP "/>" CRLF
[ "<COMMENTS>"  CRLF ]
[ *( comment CRLF) ]
[ "</COMMENTS>"  CRLF ]
"</METADATA-VERSION>" CRLF
```

metadata-version = 1*2DIGITS "." 1*2DIGITS ["." 1*3DIGITS]

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

Date = Date using the format defined in RFC 1123.

code-name = 1*10ALPHANUM

long-name = 1*48TEXT

comments = <any TEXT>

An example Version section follows:

```
<METADATA-Version="1.00.000">
<TIMESTAMP Date= "Sat, 20 Mar 1999 12:03:38 GMT" />
<SYSTEM Code= "NTREIS" Name= "North Texas Real Estate Information System" />
<COMMENTS>
This is a comment line
</COMMENTS>
</METADATA-VERSION>
```

10.2.3 Resources

The Resource definition section will start with a <METADATA-RESOURCE> tag. This tag is followed by a <COLUMNS> section which contains the name of the fields as defined in Table 10-2 followed by the <FIELD> section which contains the actual field information. The Resource section has the following format:

```
"<METADATA-RESOURCE>" CRLF
"<COLUMNS>"  CRLF
resource-field *(HT resource-field) CRLF
"</COLUMNS>"  CRLF
"<FIELDS>"  CRLF
*( resource-data *(HT resource-data) CRLF)
"</FIELDS>"  CRLF
"</METADATA-RESOURCE>" CRLF
```

resource-field = <Field Name from Table 10-2>

resource-data = <valid value as defined in Table 10-2>

An example Resource definition follows:

```
<METADATA-RESOURCE>
<COLUMNS>
StandardName<t>VisibleName<t>Description<t>ResourceID<t>ClassCount<t>ObjectTypeCount<t>Version<t>ChangeDate
</COLUMNS>
<FIELDS>
Agent<t>Agent<t>Agent Table<t>Agent<t>1<t>0<t>1.00.000<t> Sat, 20 Mar 1999 12:03:38 GMT
Property<t>Property<t>Property Table<t>Property<t>5<t>1<t>1.00.000<t> Sat, 20 Mar 1999 12:03:38 GMT
</FIELDS>
</METADATA-RESOURCE>
```

Table 10-2 Metadata Content – Resource

Field Name	Content Type	Description
StandardName	Alphanumeric	The name of the resource. This must be a well-known name if applicable.
VisibleName	Alphanumeric	The user-visible name of the resource.
Description	Printable	A user-visible description of the resource.
ResourceID	Alphanumeric	The name of the item which acts as a unique ID for this resource. This ID is passed as the Object parameter to the GetObject transaction when retrieving metadata. This value must be in the Class and Object sections (see Section 10.2.4 and 10.2.5)

Field Name	Content Type	Description
ClassCount	Numeric	The number of classes sharing this resource description. There must be one class description entry for each class.
ObjectTypeCount	Numeric	The number of predefined object types available within this resource. There must be one object type entry for each object type.
Version	Character	The version of the Metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this date for cache management.
ChangeDate	Date	The date on which the content of this resource was last changed. Clients MAY rely on this date for cache management.

10.2.4 Searchable Resource Classes

A given resource may house multiple classes of entries that can be searched separately. The Metadata for a resource supporting searchable classes MUST contain a class description for each class supported.

The Resource Class definition section will start with a <METADATA-CLASS> tag. This tag is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 10-3 followed by the <FIELD> section, which contains the actual field information. The Class section has the following format:

```
"<METADATA-CLASS>" CRLF
"<COLUMNS>"  CRLF
class-field *(HT class-field) CRLF
"</COLUMNS>"  CRLF
"<FIELDS>"  CRLF
*( class-data *( HT class-data) CRLF)
"</FIELDS>"  CRLF
"</METADATA-RESOURCE>" CRLF
```

class-field = <Field Name from Table 10-3>

class-data = <valid value as defined in Table 10-3>

An example Resource Class definition follows:

```
<METADATA-CLASS>
<COLUMNS>
ResourceID<t>ClassName<t>VisibleName<t>StandardName<t>Description
</COLUMNS>
<FIELDS>
Property<t>RES<t>Single Family<t>ResidentialProperty<t>Single Family Residential
Property<t>CON<t>Condos and Townhomes<t>CommonInterest<t>Condos and Townhomes
Property<t>MUL<t> Multi Family <t>MultiFamily<t> Multi Family Residential
Property<t>MOB<t>Mobile Home <t> ResidentialProperty<t> Mobile Homes
Property<t>LND<t>Lots and Land<t> Lots and Land <t>Lots and Land
</FIELDS>
```

```
</METADATA-CLASS>
```

Table 10-3 Metadata Content – Resource Class

Metadata Field	Content Type	Description
ResourceID	Alphanumeric	This is the same as the ResourceID for the associated parent resource as defined in Section 10.2.3.
ClassName	Alphanumeric	The name of the class. This "Name" will also appear as the class-name in the metadata defining the class (see Section 10.3.2).
VisibleName	Alphanumeric	The user-visible name of the class.
StandardName	"ResidentialProperty" "LotsAndLand" "CommonInterest" "MultiFamily"	The XML standard name. This the name from the Real Estate Transaction XML DTD.
Description	Printable	A user-visible description of the resource.

10.2.5 Object Types

Object type names allow the operator of a particular server to advertise its supported multimedia types. These types are standard MIME types as described in Section 5.1.

The Resource Class definition section will start with a <METADATA-OBJECT> tag. This tag is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 10-4 followed by the <FIELD> section, which contains the actual field information. The Class section has the following format:

```
"<METADATA-OBJECT>" CRLF
"<COLUMNS>"  CRLF
object-field *(HT object-field) CRLF
"</COLUMNS>"  CRLF
"<FIELDS>"  CRLF
*( object-data *(HT object-data) CRLF)
"</FIELDS>"  CRLF
"</METADATA-OBJECT>" CRLF
```

object-field = <Field Name from Table 10-4>

object-data = <valid value as defined in Table 10-4>

An example Resource Object definition follows:

```
<METADATA-OBJECT>
<COLUMNS>
ResourceID<t>StandardName<t>VisibleName<t>Description
</COLUMNS>
<FIELDS>
Property<t>image<t>Photos<t>Property Photos
```

```
</FIELDS>
</METADATA-OBJECT>
```

Table 10-4 Metadata Content – Resource Object

Metadata Field	**Content Type**	**Description**
ResourceID	Alphanumeric	This is the same as the ResourceID for the associated parent resource as defined in Section 10.2.3.
StandardName	Alphanumeric	The name of the object type. This is the "type" that a client can pass to the "Accept" parameter in the Get Object transaction (see Section 5.1).
VisibleName	Alphanumeric	The user-visible name of the object type.
Description	Printable	A user-visible description of the object type.

10.3 Metadata Format for Searchable Resources

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

10.3.1 Version

The metadata contains a Version section exactly like the one defined for the Metadata Dictionary defined in Section 10.2.2 and should exactly match both the TimeStamp and Version provided in the Metadata Dictionary.

10.3.2 Tables

The Table definition section will start with a <METADATA-TABLE> tag indicating the table name. This tag is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 10-5, followed by the <FIELD> section, which contains the actual field information. The Table section has the following format:

```
"<METADATA-TABLE" SP "Name" "=" class-name *SP ">" CRLF
"<COLUMNS>"  CRLF
metadata-field *(HT metadata-field) CRLF
"</COLUMNS>"  CRLF
"<FIELDS>"  CRLF
*( field-data *(HT field-data) CRLF)
"</FIELDS>"  CRLF
"</METADATA-TABLE>" CRLF
```

class-name = 1*32ALPHANUM
This is the same as the ClassName defined in Section 10.2.4.

metadata-field = <Field Name from Table 10-5>

field-data = <valid value as defined in Table 10-5>

An example Table definition follows:

```
<METADATA-TABLE Name="RES">
<COLUMNS>
SystemName<t>StandardName<t>LongName<t>ShortName<t>Maximumlength<t>DataType<t>
Precision<t>Searchable<t>Interpretation<t>Alignment<t>UseSeparator<t>EditMask<t>LookupName
</COLUMNS>
<FIELDS>
LN<t>ListID<t>Listing ID<t>ListID<t>8<t>Int<t>0<t>1<t>Number<t>Left<t>0<t><t><t>
PTYP<t>PropType<t>Property Type<t>Prop Type<t>2<t>Int<t>0<t>1<t>Number<t>Left<t>0<t><t><t>
LP<t>ListPrice<t>List Price<t>Lst Pr<t>8<t>Int<t>0<t>1<t>Currency<t>Right<t>1<t><t><t>
OWN<t>Owner<t>Owner Name<t>Own Name<t>20<t>Character<t>0<t>0<t><t>Left<t>0<t><t><t>
VEW<t>View<t>Type of View<t>View<t>10<t>Long<t>0<t>1<t>LookupBitmask<t>Left<t>0<t><t>VEW<t>
EF<t>ExtFeat<t>Ext Features<t>Ext Feat<t>10<t>Character<t>0<t>1<t>LookupMulti<t>Left<t>0<t><t>EFT<t>
SD<t>SchDist<t>School District<t>SchDist<t>10<t>Character<t>0<t>1<t>Lookup<t>Left<t>0<t><t>SD<t>
AR<t>MLSArea<t>MLS Area<t>Area<t>4<t>Int<t>0<t>1<t>Lookup<t>Left<t>0<t><t>AR<t>
</FIELDS>
</METADATA-TABLE>
```

The following table lists the minimum acceptable content for server-supplied metadata used in describing a table.

Table 10-5 Metadata Content - Tables

Field Name	Content Type	Description
SystemName	Alphanumeric	The name of the field as it is known to the native server.
StandardName	Alphanumeric	The name of the field as it is known in the Real Estate Transaction XML DTD.
LongName	Printable	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.
ShortName	Printable	An abbreviated field name that is also localizable and human-readable. Use of this field is implementation-defined.
MaximumLength	Numeric	The maximum length of the field, in characters or digits as appropriate.
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 HH:MM:SS[.TTT] format.
	"Time"	A time, stored in HH:MM:SS[.TTT] format.

Field Name	Content Type	Description
	"Tiny"	A numeric value that can be stored in no more than 8 bits.
	"Small"	A numeric value that can be stored in no more than 16 bits.
	"Int"	A numeric value that can be stored in no more than 32 bits
	"Long"	A 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)
	"LookupMulti"	A value that should be looked up in the Lookup Table. This is a multiple-selection type lookup (e.g. FEATURES)
	"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)
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. This is decimal alignment for Decimal DataTypes.
UseSeparator	Boolean	A truth-value which indicates that the numeric value MAY be displayed with a thousands

Field Name	Content Type	Description
		separator.
Mask (optional)	Character	A string which can be used by the client to perform simple insertion editing on a text or numeric field.
LookupName	Character	The name of the LookupData section containing the lookup data for this field. Required if Interpretation is Lookup.

Date formats are based on ISO 8601 [7]

10.3.3 Lookups

The lookup section of the document describes all ancillary tables that are referenced in the Table section of the document. There MUST be a corresponding lookup for every "Lookup", "LookupMulti" and "LookupBitmask".].

Each Lookup entry will start with a <METADATA-LOOKUP> tag indicating the lookup name. The name MUST be the name from the LookupName column of the table definition. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 10-6, followed by the <FIELD> section, which contains the actual lookup field information. The Lookup section has the following format:

```
"<METADATA-LOOKUP" SP "Name" "=" lookup-name *SP ">" CRLF
"<COLUMNS>"  CRLF
lookup-field *(HT lookup-field) CRLF
"</COLUMNS>"  CRLF
"<FIELDS>"  CRLF
*( lookup-data *(HT lookup-data) CRLF)
"</FIELDS>"  CRLF
"</RETS-LOOKUP>" CRLF
```

lookup-name	= 1*32ALPHANUM
lookup-field	= <Field Name from Table 10-6>
lookup-data	= <valid value as defined in Table 10-6>

An example Lookup definition follows:

```
<METADATA-LOOKUP NAME="AR">
<COLUMNS>
LongName<t>ShortName<t>Value
</COLUMNS>
<FIELDS>
Capitol Hill<t>Cap Hill<t>1
Juanita  Hill<t>Juanita<t>2
Maple Valley<t>Mpl Valley<t>3
Downtown Redmond<t>Dntn Rdmd<4>
</FIELDS>
</METADATA-LOOKUP>
```

Table 10-6 Metadata Content - Lookup

Field Name	Content Type	Description
LongName	Printable	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.
ShortName	Printable	An abbreviated field name that is also localizable and human-readable. Use of this field is implementation-defined.
Value	Alphanumeric	The value to be sent to the server when performing a search (this field must be numeric for LookupBitmask types).

11 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.

11.1 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. The records are separated from each other by a CRLF line termination sequence.

11.2 Transmission standards

A client or server transmitting a compact record MUST encode the data according to Table 11-1.

Table 11-1 Transmission Formats for Compact Records

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.
Character	The plain character sequence.
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[.TTT], 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-DD HH:MM:SS[.TTT], with a three-digit optional fractions of a second separated from the seconds with a decimal <".">
MultiValue	A numeric representation of a multi-value lookup. This format is also known as a bitmask format, where each bit represents one of the multi-value choices.
Boolean	A single character, either 1 for true or 0 for false.

12 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.

12.1 Connection Establishment

A client initiates communication with a server by beginning a TCP connection on any mutually agreed TCP port, with the default being 80. When the TCP connection has entered the Established state, the session proceeds to the start of the Authorization phase.

12.2 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 contains a checksum (nonce) of a concatenation of the following:

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

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.

12.3 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 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.

12.4 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 4.12).

13 Sample Sessions

To be supplied.

14 Acknowledgments

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:

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
Interealty Corporation

Dale Stinton
National Association of REALTORS®

15 Authors

Dan Musso
WyldFyre Technologies, Inc.
960 Saratoga Ave.
Suite 209
San Jose, CA 95129

Email: dan@WyldFyre.com

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

Email: btobak@optc.com

16 References

[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 2068, January 1997

[3] Rivest, R., "The MD5 Message Authentication Algorithm", RFC 1321, April 1992

[4] Crocker, D., "Standard for ARPA Internet Text Messages", RFC 822, IETF 1982

[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 2069, 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.