

# RETS Change Proposal: Object Data and Upload

Author: Libor Viktorin

Organization: MarketLinx

Telephone Number: (865) 470-1627

Address: 1400 Centerpoint Blvd. Suite 100; Knoxville, TN 37932

Email: lviktorin@marketlinx.com

Status: Proposal

Date: July 20, 2007

Version: 1.8

## 1. Synopsis

This proposal adds a missing functionality to the RETS specifications. It specifies a method to upload binary objects, such as images, to the server. It also addresses the need to store and expose detailed information about these objects, and makes it easier to search for them.

This is a second updated version of "RETS Change Proposal: Object Upload" filed on August 8, 2003.

## 2. Rationale

The RETS currently does not specify a way to upload objects to the server. This proposal adds a PostObject transaction, which complements the GetObject transaction already specified. It also adds a way of exposing additional data about objects such as descriptions, captions, image sizes, modification dates, etc. With this additional info, a client can search for objects matching some criteria, rather than requesting objects only by their order number.

## 3. Proposal

### 3.1. Specification Changes and Additions

#### 3.1.1 ObjectData field in Object metadata

The following line should be added to the Table 11-16 (Object Metadata Content). Adequate change should be done to the METADATA.dtd file.

Field Name	Content Type	Description
PostSupport	0	PostObject transaction is unavailable for this object type
	1	PostObject transaction is available for this object type
ObjectData	RETSNAME:RETSNAME	Resource and Class of a table that provides additional data about objects described by this metadata. If an object contains no additional data, this field <b>MUST</b> be empty or missing.
MaxFileSize	Numeric	Indicates that maximum file size (in bytes) that is accepted by the server. The server <b>MAY</b> refuse any objects sent that are greater than this size. The server <b>MAY</b> return an http error code if an object bigger than this is received.

### 3.1.2 Object Data classes

The following should be added to the specs as chapter 5.12. The current chapter 5.12 (Error codes) should be renumbered to 5.13, and its table 5-1 should be renumbered to 5-2.

## 5.12 ObjectData classes

The server **MAY** expose additional data for object files. If it does, such a data **MUST** be exposed in a class linked to an object type via the ObjectData metadata field in the Object metadata. Any table linked this way to the Objects **MUST** hold exactly one record for each object file available through the GetObject transaction on the linked Resource and ObjectType. The data must always correctly describe the object (eg. if a FileSize field is exposed, its value **MUST** be the length of a file sent by the GetObject transaction).

A good practice would be for the server to expose all such tables within one resource named OBJECT; however, the server is free to use any resource and class name as it sees fit.

Any class linked to an Object metadata item **MAY** expose any data about the object; however, if the data is one of the fields listed in table 5-1, the class **MUST** use the standard names from that table. Any class **MUST** expose fields shown in **bold**. Data from the ObjectData table will be communicated via HTTP headers (see 5.4.2 and X 1.2).

### TABLE 5-1.

<b>Standard Name</b>	<b>Data Type</b>	<b>Description</b>
<b>UID</b>	CHARACTER	Unique ID. This field must be unique within the class and its resource.
<b>ObjectType</b>	CHARACTER	ObjectType of this object. This is the Type parameter in the GetObject request.
<b>ResourceName</b>	CHARACTER	Standard name of the resource this object belongs to. This is the Resource parameter in the GetObject request.
<b>ResourceID</b>	CHARACTER	Value of the Key Field identifying a record within the resource described by ResourceName. This is the first part of the ID parameter in the GetObject request.
<b>ObjectID</b>	INT	Ordinal number of this object within all objects belonging to the record identified by ResourceName and ResourceID. This is the ObjectID in the GetObject request.
<b>MimeType</b>	CHARACTER	MimeType of the object
<b>IsDefault</b>	BOOL	1 if this object is the default one (sent when an object with ObjectID= 0 was requested). This is the main object that should be displayed for the ObjectType.
<b>ObjectModificationTimestamp</b>	DATETIME	Time of the last modification of the object
<b>ModificationTimestamp</b>	DATETIME	Time of the last modification of this data record (including the object modification)
<b>OrderHint</b>	INT	<p>Provides an override for the ObjectID value so a client can specify an alternate ordering of ObjectData while preserving the one specified by the ObjectIDs. This allows clients performing updates using the UID field perform operations such as resource reordering more easily.</p> <p>Unlike the ObjectID, where ordinal values are defined by starting with the number 1 and using the formula <math>n+1</math> to arrive at each consecutive term in the sequence, the OrderHint values can form any integer set provided that sequence of items within the set is preserved.</p>

		<p>If A and B are two objects in a collection then the following expression must be true:          If ObjectA.OrderHint &gt; ObjectB.OrderHint then          ObjectA.ObjectId &gt; ObjectB.ObjectId</p> <p>If the OrderHint is supported, then the server must maintain the number as sent by the client.</p>
Description	CHARACTER	Description of the object.
Caption	CHARACTER	Short title of the object.
FileSize	INT	Size, in bytes, of the object
WidthPix	INT	Width of an image, in pixels
HeightPix	INT	Height of an image, in pixels
Duration	INT	Length of a movie or audio, in seconds
WidthInch	INT	Width of an image, in inches
HeightInch	INT	Height of an image, in inches
...		Other well-known fields will be defined later

**Searching for objects:** Any ObjectData class is searchable as any other class. The object data may be searched by any searchable field specified in the class. The ResourceName, ResourceID and Order or UID of the matching results may be used as parameters in a subsequent GetObject transaction requesting the matching objects.

**Updating object data:** The server MAY expose updates for the ObjectData class. However, the data MUST stay consistent with the object files, so eg. the system must not allow changing the WidthPix field unless it is able to crop or resize the underlying image file.

The system MUST NOT expose an Add update for the ObjectData class, since objects will be added via the PostObject transaction. The system MAY expose a Delete update, which will have a similar effect as the PostObject transaction with the Update=Delete (namely, it will remove a record from the Object table AND delete the object file). The system MAY also expose a CLONE transaction to allow for reusing an existing object in more than one record.

The system MAY expose an UPLOAD update. Clients SHOULD NOT use this update directly; if they do, server MUST refuse it. This update type is used to hook up validation expressions and update help to the PostObject transaction. If the UPLOAD update exists, both client and server SHOULD use it to check data sent via the PostObject transaction. The UPLOAD update's validation expressions may place constraints on any field defined

in the Object table, thus specifying acceptable file characteristics.

Any document listing Standard Names should be updated to reflect the names in Table 5-1.

### 3.1.3 GetObject transaction modification

The following section should be added to the end of RETS specification, chapter 5.3 (GetObject Required Request Arguments):

**UID** The string identifying the object(s) being requested:

UID ::= TOKEN \*( "," TOKEN)

The UID argument allows for requesting objects by their UIDs, as described in table 5-1. Either ID or UID argument **MUST** be present in the request, but not both of them. If the server does not support UIDs for the requested type of objects, it **MUST** respond with an error (Preferred code would be 20403: No object found. Servers that do not implement the functionality proposed in this document may respond with 20402: Invalid Identifier). However, if the requested type of objects has an ObjectData class linked in its metadata, the server **MUST** support this argument.

The following section should be added to the RETS specifications, chapter 5.4 (GetObject Optional Request Arguments):

### 5.4.2 ObjectData

ObjectData \* | FieldName \*(, FieldName)

FieldName ::= RETSNAME

This parameter indicates that data relevant to the object should be sent as HTTP headers in the server response . If ObjectData is set to "\*", the server **MUST** include a header line for each field in the ObjectData class linked to the requested Object Type, as described in chapter 5.6. If ObjectData is set to a list of fields, ObjectData headers for the requested fields only **MUST** be sent in the response. If this argument is missing, no ObjectData will be sent.

The following section should be added to the RETS specification, chapter 5.6 (GetObject Optional Server Response Header Fields)

**UID**

The UID of the object. This field is required and **MUST** be returned if the request used UID rather than ID argument, and **MAY** be sent if ID has been used.

## ObjectData

If the client has submitted a request with “ObjectData” the header of the response **MUST** contain the ObjectData header field(s). The server **MUST** include a header line for each requested field in the ObjectData class linked to the requested Object Type. Each such header will have the name of “ObjectData”, and its value will be the SystemName of the field, followed by an equals sign, followed by the value of the field.

Example:

```
ObjectData: PropMediaCaption=caption for kitchen
ObjectData: PropMediaDescription=details about kitchen
```

### 3.1.4 PostObject transaction

The following section should be added to the RETS specifications as a special chapter (replace X with the chapter number)

## Section X: PostObject Transaction

The PostObject transaction is used to upload structured information related to known system entities. This transaction allows the client to send one file as a MIME type along with more information. The server's response is similar to that of the Update transaction.

If the server supports the PostObject transaction, it sets the PostSupport field in the Object metadata to 1.

Before a client tries to upload an object, it **SHOULD** check for the presence of the ObjectData class in the metadata. If such a class exists, the client **SHOULD** check all its validation expressions, supplying the characteristics of the uploaded file as values of the known fields of the ObjectData table. [See the **table 5-1**].

The PostObject request **MUST** be sent using POST method. The request arguments for the transaction are sent using HTTP headers.

### X.1 Required request header fields

In addition to the Required client request header fields specified in section 3.4, the header of any single-file message MUST contain the following fields:

UpdateAction	= <i>Add   Replace   Delete</i>
Content-type	= <mime-type as defined in 5.1>
Content-length	= <length of the posted data>
Type	= <object type as defined in table 11.11>
Resource	= <ResourceID of a resource as defined in table 11.3>

## **X.2 Conditionally required request header fields**

ResourceID	= <i>resource-id</i>
ObjectID	= <i>1*5DIGIT</i>
UID	= <i>TOKEN</i>
OrderHint	= <i>1*5DIGIT</i>

*resource-id* is a value from the KeyField of the Resource for which the object is to be uploaded.

ObjectID is the order number of an object within the ID. It corresponds to the Object-ID argument for the GetObject transaction.

UID is the UID of an existing object, as reported by the server in a previous PostObject, or in the related ObjectData class.

OrderHint is a number suggesting where in the sequence of all objects belonging to the same ResourceID an uploaded object should be placed. Unlike ObjectID numbers, which must be an uninterrupted sequence of integral numbers starting with 1, the OrderHint may be any number. After an update, the server must modify ObjectID values for a given ResourceID, to ensure that the ObjectID order is the same as the OrderHint order.

ObjectID numbers MUST follow the ordering of OrderHint numbers in the sense that if the OrderHint for object A is lower than the OrderHint for object B, then the ObjectID for object A MUST be lower than ObjectID for object B. In the case where multiple objects are set to share the same OrderHint value, the resulting ObjectID ordering is non-predictive. In the case of a multi-part upload, any ObjectID reordering that needs to be done to synchronize with the OrderHint order will be done by the server after all the objects are loaded.

The OrderHint MUST NOT be used in the PostObject request if it is not exposed in the ObjectData class linked to this object metadata (see 5.12 and Table 5-1). If it is used, the server should ignore the parameter.

Depending on the UpdateAction value, the ResourceID, ObjectID, UID or OrderHint may be missing from the request.

If UpdateAction=Add and the ResourceID and either ObjectID or OrderHint number is

used, the uploaded file will be added to the list of objects. The server will adjust ObjectIDs to ensure that the uploaded object assumes appropriate position in the list of existing objects. If ObjectID is used, the server MUST increase the ObjectID by one for existing objects that have an ObjectID that is of the same value or greater than the ObjectID of the inserted object. If the number of existing objects is less than the ObjectID, the uploaded object becomes the last one in the list. If OrderHint is used, the server recalculates ObjectIDs of all objects so that they stay in sync with the order of the OrderHint values. If a client sends values for both ObjectID and OrderHint, the server MUST return error, preferably 20804 (Inconsistent parameters).

If UpdateAction=Add and the UID is given, the behavior is the same as if ResourceID and ObjectID of the object identified by the UID were requested. Namely, the uploaded file will be inserted before the object identified by UID. If any of ResourceID, ObjectID or OrderHint are used along with UID, the server MUST return error, preferably 20804 (Inconsistent parameters).

If UpdateAction=Add and none of ObjectID, OrderHint or UID is provided, the uploaded file becomes the last in the list of existing objects. ResourceID is required in this case. The server may set its OrderHint to any number higher than all existing OrderHints for this ResourceID.

If UpdateAction=Replace, either the ResourceID and ObjectID, or the UID MUST be used. The uploaded file replaces the original object. Any ObjectData fields not sent with the PostObject request (and not affected by the uploaded file) keep their previous values. If a client sends values for both ObjectID and UID, the server must return error, preferably 20804 (Inconsistent parameters).

If UpdateAction=Delete, either the ResourceID and ObjectID, or the UID MUST be used, and they MUST identify an existing object. The body of the request SHOULD be empty and MUST be ignored by the server. It is expected that the server will re-adjust the ObjectIDs such that there is no gap introduced by a Delete. If a client sends values for both ObjectID and UID, the server must return error, preferably 20804 (Inconsistent parameters).

The Delete request may also be sent with ResourceID without ObjectID, in which case all objects for the given ResourceID are to be deleted.

### **X.3 Optional request header fields**

The client MAY specify any other headers. If the header coincides with a System Name of a field in a correspondent ObjectData class (see 5.12), the server MAY use the header's value to update that field. It's the server's decision whether such a field will be set to the client provided value, or calculated based on the uploaded file or other data. However, the server SHOULD calculate the values (rather than using the client-provided values) whenever it's able to do so. Specifically, the field with standard name FileSize SHOULD

always reflect the length of the file as it is stored with the server.

The client SHOULD check the ObjectData class to see what headers may be needed for the server.

Warning-Response = *warning-num=user-response*

**Comment [LV1]:** This should be changed if we change the Warning-response in Update

In case there were any warning while validating the request, the client MAY send Warning-Response header. If a server responded to a previous PostObject request with a WarningBlock (see X.1.4), the client SHOULD include a Warning-Response header(s) when re-posting the request. The syntax and semantics of this header should resemble that of the WarningResponse argument from the Update transaction.

#### X.4. Request body

The body of the request is the file being uploaded. If UpdateAction=Delete, the body MAY be empty, and SHOULD be ignored by the server.

#### X.5 Server response body format

The response from the server is similar to that of the Update transaction (10.5):

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

In the compact-data, the server MUST send the values of Resource, Type, ResourceID, ObjectID and UID, if they were used in the request. The UID, if it exists in the related ObjectData class, MUST be sent even if it was not requested. The UID MAY also be sent if no ObjectData class is linked to this Object metadata, but the server is able to honor GetObject requests with UID.

Unless the UpdateAction requested was Delete, all other fields from the ObjectData table that were requested or changed MUST also be sent. Other fields from the ObjectData table MAY be sent as well.

*activation-tag* ::= TIMESTAMP [ ; TEXT ]

If the object is not immediately accessible, the server MUST send a datetime when it is supposed to be activated. An explanation why the object is delayed MAY be appended.

The reply code MUST be zero (success) even if the object is accepted for testing only. However, if the server knows in the time of the transaction that the object will be refused, it SHOULD reply with an error reply code (eg. 20809).

*error-block* and *warning-block* are explained in section 10.5.

## **X.6. Reply Codes**

**Table X.1 Standard Reply Codes**

0	Upload successful.
20800	Unknown resource
20801	Invalid object type
20802	Invalid identifier
20803	Invalid update action
20804	Invalid (inconsistent) request parameters
20805	No object found (for Delete)
20806	Unsupported MIME type
20807	Unauthorized
20808	Some objects not deleted (in case of Delete without ObjectID or UID, if some objects could not be deleted, while some were)
20809	Refused: object does not meet business rules
20810	FileSize too large Note that some servers MAY respond with HTTP status “413 – Request entity too large” if the uploaded file is larger than any acceptable limit.
20811	Timeout
20812	Too many outstanding requests
20813	Miscellaneous error

Chapter 4.10 (Capability URL List) should be modified to account for an optional PostObject transaction.

## **4. Compatibility**

The ObjectData and PostSupport metadata fields in the Object metadata are optional. Clients implementing this proposal should make sure that they can handle metadata without these items to stay compatible with older servers. Older clients may have

problems with new servers, since the metadata DTD is changed, but there is a chance even old clients will work correctly, if they are able to just ignore unknown metadata items.

Object data classes are regular classes, so they should not pose any compatibility risk.

Since previous versions of the specs did not allow for object upload, there are no compatibility issues with the PostObject transaction.

## **5. Acknowledgements**

Thanks to Sergio Del Rio and all participants in the Rets Upload Workgroup for revising this proposal and making many valuable comments, which shaped this version of the document.