RCP - WEBAPI-019 Validation Expression in the WebAPI

Submitter Name	Joshua Darnell
Submitter Organization	kurotek, LLC
Submitter Email	josh@kurotek.com
Co-submitter Name	Paul Stusiak
Co-submitter Organization	Falcon Technologies Corp.
Co-submitter Email	pstusiak@falcontechnologies.com
Attributions	This proposal is based on the RETS Validation Expression language as expressed in RETS 1.9 and RCP 61. Thanks to Libor Viktorin, Mark Sleeman, Sergio Del Rio and Paul Stusiak for that work. Thanks to Rob Larson for his collaboration on the Rules Resource proposal.

Document Name	**RESO Web API v1.1 Draft**
Document Version	1.1.0 (MAY for Add/Edit Endorsement in Web API)
Date Submitted	2018-04-12
Status	IN DRAFT
Status Change Date	
Related Documents	**RCP-019 ANTLR G4 Grammar** **RCP-010 Add/Edit in the Web API** **Web API 2.0.0**

Table of Contents

Synopsis

The ability to express computer-executable rules has several uses. The ability to provide immediate feedback to users when adding a record is valuable. The ability to consume rules has value in server systems that add or modify records or during a server system migration. Using the existing BNF of the RETS 1.9 as a basis, it provides a rich language to express field validation and business rules and is proven in the field. Some minor changes to the BNF have been made to better support the Web API and correct some minor errors.

Rationale

Substantial work was done to review the validation expression language of Odata as it compares with the validation expression language of RETS 1.9. The findings in a RESO white paper (DRAFT) written by Josh Darnell were that the Odata 4.01 validation expression BNF lacked several critical features that are used in RETS 1.9 to create valid records.

Excerpts of that white paper are included here.

> *In RETS 1.9, business rules are represented as an ordered list of field, action, expression tuples. These rules are intended to be written from the perspective of the Business Object being upserted, for instance, a "Listing".*
>
> *The Validation Expression scheme assumes that this sequence of rules will be run, in order, after each relevant change to the intermediate payload being processed. Doing so guarantees that any expression that depends on the current instruction will be re-processed accordingly.*
>
> *Techniques exist to optimize the set of instructions being run so that only the dependent subset of rules is run when a given field changes, making 1.9 Validation Expressions appropriate for implementations where "fast-fail" behavior is important.*
>
> *...*
>
> *While OData does not currently support the Actions used in RETS 1.9 Validation Expressions, parity (in this case) means that both grammars support the same basic set of functionality, meaning the same set of Boolean Expressions, Arithmetic Operations, Lists, Grouping Operators, and Function Calls.*
>
> *This is important for the standard going forward because it means that once OData has a similar set of Actions in place, that any RETS 1.9 Validation Expression can be represented in the OData 4.01 grammar. The parity between the expression grammars also means that automatic translation may be done from 1.9 Validation Expressions to the OData 4.01 Validation grammar, potentially using tools such as ANTLR.*

However, as it is currently provided, Odata 4.01 does not have the concept of ACTION nor of the sequential processing of rules. Further, Odata 4.01 is not a ratified standard and may be subject to change.

Since there is not functional equivalence of the validation expression languages between RETS 1.9 and Odata 4.01, we propose that the existing validation expression language of RETS 1.9 be used to support the Update transaction in RESO Web API 1.1

In addition to the grammar itself, information about a particular session (a client application, with a user in a particular role) may be needed to resolve individual rules. This information is in the form of *info-session-tokens* identical to those of RETS.

The *info-session-tokens* can be used as values to substitute in rules that are not available in the Resource that is being modified.

The grammar is used to express the rules in a machine-readable form. The rules are expressed as a set of rule statements that MAY be limited by the application, user role and user by the implementing system.

To correctly implement the system, a client would implement a parser-generator to execute the rules. For an Entity that is being modified, the client would request the current state of the Entity. For either a new or modified Entity, the client would request the set of rules for the Resource type of the Entity. For either a new or modified Entity, the client would request the *info-session-tokens* for the current session. The client can then guide the user through the Entity, executing the rules as the user moves between the fields of the Entity. When the user is complete, the client can submit the Entity to the server. The client SHOULD expect that the server may issue warnings and errors against the Entity despite the client passing the rules.

Proposal

The following new sections be added to the Web API 1.1 document,

New Section: 2.7.7 Validation Rules for a Resource

A service MAY provide business rules to permit complex validation of data before submission to the service. Clients should use these validation rules to provide the user with a more responsive experience. A service MAY have additional rules that are not provided to the client and SHOULD validate any changes to an Entity on the service when the client submits a changed Entity.

Validation Rules are for a specific Resource and MAY be constrained by the client application, role or other system specific limitation.

Validation Rules for a Resource will change over time. To allow the evolution of the rules, an opaque token is provided to uniquely identify the state of the set of rules. Clients MAY cache Validation Rules.

vrHash	base64	A base64 opaque token, calculated in the service to uniquely identify the state of the Validation Rule set

Validation Rules are returned as a JSON response body from the service. The response body includes the `vrHash` for that rule set.

Accessing a rule set must adhere to the OData standard taking the following form:

```
https://odata.reso.org/RESO/OData/[Resource]/ValidationRules('<vrHash|null>')
```

A service should treat the endpoint `./ValidationRules` identically to the endpoint `./ValidationRules('')`

Changing rules, roles or other constraints MUST change the `vrHash` value returned by the service. If a client submits a stale `vrHash`, the client SHOULD use the Validation Rule set returned in the response body and discard any cached validation rules. A client SHOULD submit the `vrHash` on a ValidationRules request to allow the service to limit the response body to just the `vrHash`. That is, when the request `vrHash` matches that of the service, the service MAY return only the `vrHash` in the response body.

The rule set may be further filtered by adding the action that the client will use the rule set for. The action is defined in section 2.7.8.4.1.

```
https://odata.reso.org/RESO/OData/[Resource]/ValidationRules('<vrHash|null>')?$filter=action
```

The response body is JSON formatted consisting of the `ruleSet`, which is an ordered list of validation expressions, and the `ruleHash`. The `ruleSet` is the collection of rules the service provides for the selected Resource during creation or modification of an Entity of that Resource type. The `ruleSet` is optional, since there may be no rules available for the client for that Resource. In this case, the service MUST return an HTTP status of 204.

The request below returns the collection of *all* validation expression rules for the Property resource:

Request:

```
GET RESO/OData/Property/ValidationRules('32248c144')
```

Response body:

```
 HTTP/1.1 200 OK
 Content-Type: application/json; odata.metadata=minimal; odata.
streaming=true
 OData-Version: 4.0
 Date: Sat, 28 Jun 2018 01:05:32 GMT
 Content-Length: xxxx

 {
   "@odata.context": "https://odata.reso.org/RESO/OData/Property
/ValidationRules('32248c144')",
   "value": {
     "vrHash": "667qa3321158",
     "ruleSet": [
       {"sequence": 1, "field": "ListPrice", "action": "SET_REQUIRED",
"expression": ".TRUE.", message: "ListPrice is Required."},
       {"sequence": 2, "field": "ListingId", "action": "REJECT",
"expression": "UserLevel != 'Admin' .AND. ListPrice <= 0", message:
"ListPrice must be greater than zero."}
     ]
   }
 }
```

The ValidationRules URL allows filtering by UpdateAction, where update_action is one of those specified in 2.7.10.

```
GET Property/ValidationRules('32248c144')?$filter=update_action eq 'Add'
```

The response is similar to that shown above, except that the rules will be only those for the requested update_action.

New Section: 2.7.8 Session Information Tokens for a Session

Validation Rules for a Resource may depend on values that are part of a different Resource or values that depend on the session. To provide this information, a URL is provided to allow the client to receive this information. See section 2.7.9 for more information on Session Information Tokens

Accessing the Session Information Tokens must adhere to the OData standard taking the following form:

```
https://odata.reso.org/RESO/OData/InfoTokens
```

The response body contains the collection of InfoTokens for this service.

Request

```
GET RESO/OData/InfoTokens
```

Response

```
 HTTP/1.1 200 OK
 Content-Type: application/json; odata.metadata=minimal; odata.
streaming=true
 OData-Version: 4.0
 Date: Sat, 28 Jun 2018 01:05:32 GMT
 Content-Length: xxxx

 {
   "@odata.context": "https://odata.reso.org/RESO/OData/InfoTokens",
   "value": {
         "USERID": "ag332354",
     "USERLEVEL": "Agent",
     "AGENTCODE": "22456",
     "BROKERCODE": "553",
     "BROKEROFFICE": "M33"
    }
 }
```

New Section: 2.7.9 Session Information Tokens

The following tokens are defined:

info-token-key	::= **Info =** *info-token-name* [; *info-token-type*] ; *info-token-value CRLF*
info-token-name	::= *RETSNAME*
info-token-type	::= *TOKEN*
info-token-value	::= *TEXT*

- An information token is a named and typed piece of information about the current session. This information is sent by the server to the client for use in various contexts. For example in Validation Expressions.
- Any number of information tokens can be sent in the response, provided all of them have unique names.
- The *info-token-type* may be any of the DataTypes defined in this document or in the Odata standard. The *info-token-value* MUST conform to the token's data type.
- If the *info-token-type* is null or missing, the data type of the token is Character. In this case, the *info-token-value* MUST NOT include semicolons. When the *info-token-name* is a string as defined in the table below, Servers MUST use the DataType described therein. Clients SHOULD be permissive, that is, when a Server has omitted an info-token-type for a well-known name, Clients should infer the DataType based on the name defined in the table below.
- Names and types of well known tokens are listed in the table. The server MUST send all *info-token-name* names shown in bold in that table. The server MUST use the well-known name for optional *info-token-name* names when providing information for those arguments. For forward compatibility, Clients MUST use the Session Information Token *info-token-value.*

Well-Known Information Tokens

Special Operand Name	Special Token Name	Notes
USERID	`user-id`	
USERCLASS	`user-class`	
USERLEVEL	`user-level`	
AGENTCODE	`agent-code`	
BROKERCODE	`broker-code`	
BROKERBRANCH	`broker-branch`	
MEMBERNAME	`member-name`	
OfficeList	Character	The value of the OfficeList token will be comma-delimited, rather than semicolon-delimited as it was in the case of the OfficeList response argument
StandardNamesVersion	Character	
VendorName	Character	
ServerProductName	Character	
ServerProductVersion	Character	
OperatorName	Character	
RoleName	Character	
SupportContactInformation	Character	

The user tokens contain basic information about the user that is stored on the server. If a server does not support a property then it MUST set the returned value to empty (a zero-length string).

`user-id`	`::= 1*30ALPHANUM`
`user-class`	`::= 1*30ALPHANUM`
`user-level`	`::= 1*5DIGIT`

The `agent-code` is the code that is stored in the property records for the listing agent, selling agent, etc.

`agent-code`	`::= 1*30ALPHANUM`

In some implementations this may be the same as the user-id. The property user-class and user-level are implementation dependent and may not exist on some systems, in which case, an empty string should be returned. These parameters are used in the validation expression language for the Update transaction.

The `broker-code` and `broker-branch` parameters are used in the validation expressions of the Update transaction.

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

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

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

The *standard-names-version* indicates the date version of StandardNames that this system supports. A system is only expected to support a single version of the StandardNames and, in most cases, this will be the current version.

`standard-names-version`	`::= 1*128TEXT`

- Server systems that do not provide this optional property make no representation about the version of StandardNames that they support, therefore, client applications should not assume any specific version of the StandardNames.
- Server systems that do provide this optional property **MUST** return a value for the standard-names-version that matches one of the values from the Adopted StandardNames List from Real Estate Transaction Standard website.
- The format of the *standard-names-version* is a string that contains numeric characters of the form M.m where M is the major version and has a range of '1-9' and m is a range of '1-99'. For example, a version of the StandardNames is *1.6.*
- **VendorName** is the name of the product vendor. It is required.
- **ServerProductName** is the name of the server product provided by the vendor. It is required.
- **ServerProductVersion** is the version of the server product. It is required.
- **OperatorName** is the name of the MLS or Association operating the system. It is required.
- RoleName is the name of the role restriction where the metadata may be restricted. It is optional.
- SupportContactInformation is free text that provides a contact email, phone or website for development support. It is optional.
- Vendors may provide additional *info-token-name* Session Information Tokens to meet local business needs. Clients MUST ignore Session Information Tokens that they do not understand.

More well-known Session Information Tokens may be added in later version of this document.

New Section: 2.7.10 UpdateAction

The list of well-known update actions is provided in this table. Subsequent versions of this standard may add actions or deprecate actions. Vendors may add actions specific to their business case, but clients are not required to support extensions to be compliant with the standard.

`UpdateAction`	`ALPHANUM`	This identifies the nature of the update, such as "add" or "modify". Some update types, such as changes to a properties record (e.g. "Sell", "Back on Market"), will imply a set of business rules specific to the server. However, where possible, the following standard type names should be used:
	`Add`	Add a new record
	`Clone`	Create a new record by copying an old record
	`Change`	Change an existing record
	`Delete`	Delete an existing record

New Section: 2.7.11 Validation Expression Language

A Validation Expression Language allows the implementation of rules of the form *field*, *action*, *expression* tuples. These rules are intended to be written from the perspective of the Business Object being upserted, for example, a "Property".

To accomplish this a formal grammar is provided to make the implementation of these tuples possible and the transmission of rules in this grammar that can be provided as input to a rules engine for the purposes of validation.

Rules are provided as the response body for a particular Resource. In a Resource, the rules for a specific field are evaluated when that field is completed by the user. For example, entering data into a field for ListingStatus then moving to another field will cause those rules associated with ListingStatus to be executed. Expressions for that field in the Resource MUST be evaluated in the order that they appear in the Resource.

New Section: 2.7.11.1 Actions

Actions identify the functional behaviour that the rule enforces.

Keyword	Type	Purpose
`ACCEPT`	`BOOLEAN`	If the expression is true, the field value is considered accepted without further testing. Immediately following SET expressions MUST be executed. If the expression is false, following validation expressions MUST be executed. If the expression is ERROR (evaluation failed) in client, the client SHOULD act as if the field was accepted, allowing the server to make the final decision.
`REJECT`	`BOOLEAN`	If the expression is true, the field value is considered rejected without further testing. Subsequent SET expressions MUST NOT be evaluated. If the expression is false, following validation expressions MUST be executed. If the expression is ERROR, evaluation failed in client, the client SHOULD act as if the field was accepted, allowing the server to make the final decision.
`WARNING`	`BOOLEAN`	If the expression is true, the client should show a warning message to the user, and if the warning is okayed by the user, include a Warning-Response in the UPDATE request. If the user does not okay the warning, the field is considered rejected and following SET validation expressions MUST NOT be evaluated. If the expression is false, the following validation expressions MUST be evaluated.
`SET`	`TYPEOF(EXP)`	~~The expression MUST begin with a field name and an equal sign ("=")~~ .

		The following expression is evaluated and the result stored in the designated field.
SET_DEFAULT	TYPEOF(EXP)	This expression MUST be executed ONLY when a NEW record is created. Supersedes the default value as indicated in the Update Metadata.
SET_REQUIRED	BOOLEAN	Expressions of this type are designed to evaluate an expression and set the field that the rule is applied on to Required if the expression returns true and to Non Required if the expression returns false.
SET_READ_ONLY	BOOLEAN	Expressions of this type are designed to evaluate an expression and set the field that the rule is being applied on to Read Only if the expression returns true and to Updateable if the expression returns false. The client application is expected to lock the value of the field the rule is being executed on to the value at the time the SET_REQUIRED expression is evaluated.
RESTRICT_PICKLIST	List of CHAR	Expressions of this type are designed to return one or more LOOKUP values that are to be removed from the LOOKUP list that is defined for the field the rule is being executed on. This is always the entire set of values to remove from the lookup. In other words, if this returns a blank list or .EMPTY., the entire set of LOOKUP values is to be displayed. The value of this expression MUST be a <List>, rather than <Exp>, as defined in 11.4.9.1. All members of the list MUST be of the same type as the type of the field the rule is being executed on.
SET_PICKLIST	List of CHAR	Expressions of this type are designed to return one or more LOOKUP values that are to be used in the LOOKUP list that is defined for the field the rule is being executed on. The value of this expression MUST be a <List>, rather than <Exp>, as defined in 11.4.9.1. Every member of the list MUST exist in the Lookup list as defined in the metadata for the field the rule is being executed on.
SET_DISPLAY	BOOLEAN	Expressions of this type are designed to allow a client to make fields visible or invisible based on the evaluation of an expression. The result of this expression has no effect on whether a field is READ ONLY or not.

New Section: 2.7.11.2 Validation Expression BNF

```
Exp                        ::= OrExp

List                       ::= LPAREN RPAREN
                                 | LPAREN Exp *(COMMA Exp) RPAREN

OrExp                       ::= AndExp *(OR AndExp)

AndExp                      ::= NotExp *(AND NotExp)

NotExp                      ::= NOT NotExp
                           | EqExp

EqExp                       ::= CmpExp
                           | CmpExp EQ CmpExp
                           | CmpExp NE CmpExp

CmpExp                      ::= CntExp
                           | CntExp LTE CntExp
                           | CntExp GTE CntExp
                           | CntExp LT CntExp
                           | CntExp GT CntExp

CntExp                      ::= SumExp
                           | SumExp CONTAINS SumExp
                           | SumExp IN List

SumExp                      ::= ProdExp *( (PLUS|MINUS|CONCAT) ProdExp)
```

```
ProdExp             ::= AtomExp *( (ASTERISK|SLASH|MOD) AtomExp)

AtomExp         ::= LPAREN Exp RPAREN
                        | Value
                        | FuncExp

FuncExp         ::= Func LPAREN Param *(COMMA Param) RPAREN

Func            ::= ALPHA *(ALPHANUM)

Param                   ::= Exp

Value                   ::= SpecValue
                        | CharValue
                        | IntValue
                        | FloatValue
                        | TimeValue
                | TimeSpanValue
                | FieldName

Concat                   ::= PIPE
SpecValue            := DOT FieldName DOT

FieldName       ::= (RETSNAME|RESONAME)
                | LBRACKET (RETSNAME|RESONAME) RBRACKET

CharValue       ::= SQUOTE PLAINTEXT SQUOTE
                        | QUOTE PLAINTEXT QUOTE

TimeValue       ::= HASH (RETSDATE|RESODATE) HASH
FloatValue      ::= IntValue DOT *(DIGIT)
IntValue            ::= 0*1(PLUS|MINUS) 1*(DIGIT
```

Notes:

1. SEE: ANTLR G4 Grammar for RCP-019 for latest version of the grammar parser specification.
2. SpecValue is further constrained - see Special Operands
3. The value of a Validation Expression MUST conform to the Exp syntax in the grammar above, except for RESTRICT_PICKLIST and SET_PICKLIST expressions, whose value MUST conform to the List syntax. Any expression with keyword starting with “X-” MAY have a List value as well.
4. The text in CharValue must not include the (single or double) quote used to delimit the value.
5. TimeValue must be expressed in the constrained ISO8601 format described below and is enclosed in hashmarks(#) (ex. #2018-09-11T14:30:00#).

   ISO 8601, RFC 3339 and the W3C note provide for additional constraints to the formats. Based on common usage patterns, this standard applies the following additional constraints to improve interoperability and compatibility. The representation of the time offset UTC character 'Z' and the date-time separator character 'T' MUST be upper case.

   The time-secfrac is limited to one digit only. The date and time representations are intended for machine processing, therefore, no whitespace is expected in any of the atoms. Examples of the format are similar to that of the W3C note, for example, 2018-07- 16T19:20: 30.4+01:00 or 2018-07-16T18:20:30.4Z. Servers and Clients MUST treat the time-offset 'Z' and "+00:00" as identical times. Servers and Clients MAY use the interpretation of RFC 3339 section 4.3 Unknown Local Offset Convention where the time-offset "-00:00" is semantically different from "+00:00" and represents a known UTC time but unknown local time.
6. A FieldName is a bracketed name of a field belonging to the same class as the field to which this expression is attached, and has a type of that field specified by the metadata. If used with the LAST keyword, its value is the value of the field as it was in the database before the current updates took place. If used without LAST, the updated value of the field MUST be used.

7. A TimeValue has TIME type.
8. A CharValue has CHAR type.
9. A IntValue has INT type.
10. A FloatValue has FLOAT type.
11. FieldName can be expressed with or without brackets by the RCP 61 spec.

The SpecValue uses the tokens defined in Section 2.7.8.4 Special Operands. See that section for valid values. These values may change between versions of the standard.

New Section: 2.7.11.3 Atoms and Primitives

Token	Definition
ALPHA	::= %x41-5A | %x61-7A 1; A-Z | a-z
CHAR	::= %x01-7F ; ANY 7-BIT US-ASCII CHARACTER, ; EXCLUDING NULL
CTL	::= %x00-1F | %x7F ; controls
DIGIT	::= %x30-39 ; 0-9
HEXDIG	::= DIGIT | "A" | "B" | "C" | "D" | "E" | "F"
OCTET	:= %x00-FF ; any 8-bit sequence of data
BOOLEAN	::= TRUE | FALSE
TRUE	::= "1"
FALSE	::= "0"
LAST	::= "LAST"
RETSNAME	::= 1*64IDALPHANUM
IDALPHANUM	::= ALPHANUM | "_"
ALPHANUM	::= ALPHA | DIGIT
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)>
QUOTE	::= %x22
NULL	::= <no character>
CRLF or	::= CR LF; special character U+21B5
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
OPTNONNEGATIVENUM	::= NULL | NONNEGATIVENUM ; null or >= 0
OPTPOSITIVENUM	::= NULL | POSITIVENUM ; null or >= 1
NONNEGATIVENUM	::= "0" | POSITIVENUM ; also known as cardinal numbers or counting numbers ; consisting of integers greater than 0

NONZERODIGIT	::= %x31-39 ; 1-9
PLAINTEXT	::= <any OCTET except CTL+S>
POSITIVENUM	::= NONZERODIGIT *DIGIT ; > 0
SERIAL	::= "-1" | NONNEGATIVENUM;
TEXT	::= <any OCTET except CTL+S, but including LWS >
TOKENCHAR	::= <any CHAR except CTL s or TSPECIALS >
TOKEN	::= 1*TOKENCHAR
TSPECIALS	::= "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | <"> | "/" | "[" | "]" | "?" | "=" | "}" | "{" | SP | HT
quoted-string	::= (<"> *(QDTEXT) <">)
QDTEXT	::= <any TEXT except <">>
RETSDATETIME	::= date-time | partial-date-time
RETSTIME	::= full-time | partial-time
DATE	::= <Date using the format defined in RFC 2616 as rfc1123-date>
OR	::= DOT "OR" DOT
AND	::= DOT "AND" DOT
NOT	::= DOT "NOT" DOT
CONTAINS	::= DOT "CONTAINS" DOT
IN	::= DOT "IN" DOT
MOD	::= DOT "MOD" DOT
LPAREN	::= %x28 ; ASCII left parenthesis character (
RPAREN	::= %x29 ; ASCII right parenthesis character)
SQUOTE	::= %x27 ; ASCII single quote character '
QUOTE	::= %x22 ; ASCII double quote character "
DOT	::= %x2E ; ASCII period character .
COMMA	::= %x2C ; ASCII comma character ,
HASH	::= %x23 ; ASCII hash character #
PIPE	::= %x7C ; ASCII pipe character |
PLUS	::= %x2B ; ASCII plus character +
MINUS	::= %x2D ; ASCII minus character -
LBRACKET	::= %x5B ; ASCII left bracket [
RBRACKET	::= %x5D ; ASCII right bracket]
SLASH	::= %x2F ; ASCII forward slash /
ASTERISK	::= %x2A ; ASCII asterisk *
EQ	::= %x3D ; ASCII equals =
EXCLAMATION	::= %x21 ; ASCII exclamation mark !
GT	::= %x3E ; ASCII less than >

LT	::= %x3C ; ASCII less than <
CTRL_S	::= %x13 ; ASCII CTRL+S
LTE	::= LT EQ ; "<="
GTE	::= GT EQ ; ">="
NE	::= EXCLAMATION EQ ; "!="
CONCAT	::= PIPE ; "|"
PLAINTEXT	::= *(~[CTRL_S]) ; Zero or more characters, no CTRL_S

New Section: 2.7.11.4 Special Operands

Token	Data Type	Description
.EMPTY.	EMPTY	A value that matches an empty or all-blank field. Supplies an empty (zero-length) field when used in a SET expression.
.TRUE.	BOOLEAN	Boolean value of TRUE (1)
.FALSE.	BOOLEAN	Boolean value of FALSE (0)
.TODAY.	TIME	The current date.
.NOW.	TIME	The current time.
.ENTRY.	TYPEOF (FIELD)	The current field text.
.OLDVALUE.	TYPEOF (FIELD)	The value that was in the field as returned from the host in the search operation. If the field is new, . OLDVALUE. is .EMPTY.
.USERID.	CHAR	The value of the user-id field returned by the function on the resource, see Section 2.7.9.3
. USERCLASS.	CHAR	The value of the user-class field returned by the function on the resource, see Section 2.7.9.3.
. USERLEVEL.	CHAR	The value of the user-level field returned by the function on the resource, see Section 2.7.9.3.
. AGENTCODE.	CHAR	The value of the agent-code field returned by the function on the resource, see Section 2.7.9.3.
. BROKERCODE.	CHAR	The value of the broker-code field returned by the function on the resource, see Section 2.7.9.3.
. BROKERBRANCH.	CHAR	The value of the broker-branch field returned by the function on the resource, see Section 2.7.9.3.
. UPDATEACTION.	CHAR	Name of the UpdateAction for which this validation is performed. (See **UpdateAction** table below, and Section 2.7 Update in the Web API 1.1 document)
.any.	See Description	If the name of the *SpecValue* (stripped of the first and last dot) is equal to a name of one of the *info-token-keys* returned as part of the function on the Resource, then the type and value of this *SpecValue* is defined by that *info-token-key*. If no such *info-token-key* exists, the value is **ERROR**.

New Section: 2.7.11.5 Time-Specific Atoms

Token	Definition
date-fullyear	::= 4DIGIT
date-month	::= 2DIGIT ; 01 - 12
date-mday	::= 2DIGIT ; 01 - 28, 01-29, 01-30, 01-31, based on month/year

time-hour	::= 2DIGIT ; 00 - 23
time-minute	::= 2DIGIT ; 00 - 59
time-second	::= 2DIGIT ; 00 - 58, 00 - 59, 00 - 60 based on leap second rules
time-secfrac	::= "."1DIGIT
time-numoffset	::= ("+"|"-") time-hour ":" time-minute
time-offset	::= "Z" | time-numoffset
partial-time	::= time-hour ":" time-minute ":" time-second [time-secfrac]
full-date	::= date-fullyear " - " date-month " - " date-mday
full-time	::= partial-time time-offset
date-time	::= full-date "T" full-time
partial-date-time	::= full-date "T" partial-time

Notes:

1. The definitions for ALPHA , CHAR , CTL , DIGIT , HEXDIG and OCTET are derived from RFC 2234.
2. ISO 8601, RFC 3339 and the W3C note provide for additional constraints to the formats. Based on common usage patterns, this standard applies the following additional constraints to improve interoperability and compatibility. The representation of the time offset UTC character 'Z' and the date-time separator character 'T' MUST be upper case.
3. The time-secfrac is limited to one digit only. The date and time representations are intended for machine processing, therefore, no whitespace is expected in any of the atoms. Examples of the format are similar to that of the W3C note, for example, 2018-07- 16T19:20: 30.4+01:00 or 2018-07-16T18:20:30.4Z. Servers and Clients MUST treat the time-offset 'Z' and "+00:00" as identical times. Servers and Clients MAY use the interpretation of RFC 3339 section 4.3 Unknown Local Offset Convention where the time-offset "-00:00" is semantically different from "+00:00" and represents a known UTC time but unknown local time.

New Section: 2.7.11.6 Functions and Operators

A Validation Expression may use Functions. We define <FuncExp> as a function with parameters. The following functions are defined:

List of Functions

Function	Parameter Types	Type	Comments
BOOL	BOOLEAN, CHAR	BOOLEAN	
CHAR	Exp, TYPEOF(Exp) NE FLOAT	CHAR	
CHARF	FLOAT, INT	CHAR	The CHARF function converts a Float number, and in the second parameter specifies how many decimal digits MUST appear after the point.
TIME	TIME, CHAR	TIME	
DATE	TIME, CHAR	TIME	
INT	INT, FLOAT, BOOL, CHAR	INT	
FLOAT	INT, FLOAT, BOOL, CHAR	FLOAT	
SUBSTR	CHAR, INT, INT	CHAR	The SUBSTR function returns a substring of its first parameter. Second parameter is a starting position of the substring, third parameter is the ending position of the substring. Positions are 1-based.
STRLEN	CHAR	INT	The STRLEN function returns the length if its parameter.
LOWER	CHAR	CHAR	The LOWER function returns its parameter lower-cased.
UPPER	CHAR	CHAR	The UPPER function returns its parameter upper-cased.
IIF	BOOLEAN, Exp, Exp	TYPEOF (Exp)	The IIF function returns the value of its second parameter if the first parameter evaluates to true, or the value of its third parameter otherwise. Types of second and third parameter must be same, and it is the type of the result. These parameters are also known as *CondExp*, *TrueExp*, and *FalseExp*, respectively.
YEAR	TIME	INT	

MONTH	TIME	INT	
DAY	TIME	INT	
WEEKDAY	TIME	INT	
TYPEOF	Exp	CHAR	The input parameter, *Exp,* can be any valid expression in the grammar. This function will return a CHAR representation of the given expression's type, one of: {BOOLEAN, CHAR, FLOAT, INT, TIME}
MATCH	CHAR, Exp	BOOLEAN	Takes a Regular Expression *regex* as a CHAR and an expression Exp, and returns True if expression matches *regex* and False otherwise.

Notes:

1. The BOOL, CHAR, TIME, DATE, INT and FLOAT functions are used just to change a type of expression. The DATE and TIME functions are synonyms. Note that any of these functions may fail (return an ERROR value) if the parameter can not be transformed to the appropriate type.
2. In conversion from BOOLEAN to INT or FLOAT, .TRUE. is converted to 1 and .FALSE. is converted to 0.
   Casting FLOAT to INTEGER discards the fractional part.
3. When converting to CHAR, BOOL values are represented as “0” and “1”, TIME values are represented using format defined in RFC 1123 with digital timezone, INT values are represented with no leading zeros.
4. When converting from CHAR to BOOL, values “0”,”1”,”YES”,”NO”,”TRUE” and “FALSE” (no matter what the case) MUST be understood.
5. When converting from CHAR to TIME, any RFC 1123 –compliant format MUST be understood. A leading and/or trailing # MUST be removed before conversion.
6. When converting from CHAR to INT or FLOAT, usual formats MUST be understood. Scientific format (with exponent) MUST NOT be understood. FLOAT numbers with empty integral part (.5, -.4) MUST be understood as long as there is at least one digit after the decimal point.
7. The YEAR, MONTH, DAY and WEEKDAY parse the date part of TIME value. They return values ranging from 1 to the appropriate maximum. WEEKDAY returns 1 for Sunday, 2 for Monday etc.
8. Other functions may be defined later (HOUR and MINUTE are first candidates). If a server uses a function the client does not recognize, the client MUST evaluate it as ERROR.
9. A Validation Expression may have Operations applied to the parameters. The Operators may be applied on certain types that determine the result type. The input types and resulting output type is defined in the Validation Expression **List of Operators** Table (below).

List of Operators

Operator	Left Operand	Right Operand	Result	Meaning
.MOD.	INT	INT	INT	Arithmetic MODULO operation
/, *	INT	INT	INT	Integer division and multiplication
/, *	INT	FLOAT	FLOAT	Division and multiplication
/, *	FLOAT	INT	FLOAT	Division and multiplication
/, *	FLOAT	FLOAT	FLOAT	Division and multiplication
+,-	INT	INT	INT	Integer addition and subtraction.
+,-	INT	FLOAT	FLOAT	Addition and subtraction.
+,-	FLOAT	INT	FLOAT	Addition and subtraction.
+,-	FLOAT	FLOAT	FLOAT	Addition and subtraction.
+	FLOAT	TIME	TIME	Time shift.
+	TIME	FLOAT	TIME	Time shift.
-	TIME	FLOAT	TIME	Time shift.
-	TIME	TIME	FLOAT	Time shift.
||	CHAR	CHAR	CHAR	String concatenation.
.CONTAINS.	CHAR	CHAR	BOOLEAN	String containment. The operation is TRUE if the left operand contains the right operand as a substring anywhere within it.
.IN.	Any	List of operands, all of the same type as the left operand	BOOLEAN	List inclusion. The operation is TRUE if the left operand is equal to any member of the list.
.AND.	BOOLEAN	BOOLEAN	BOOLEAN	

				A Boolean operator that takes two Boolean operands, and whose value is TRUE if and only if both of its operands are TRUE.
.OR.	BOOLEAN	BOOLEAN	BOOLEAN	A Boolean operator that takes two Boolean operands, and whose value is TRUE if either of its operands is TRUE.
.NOT.	BOOLEAN	BOOLEAN	BOOLEAN	A Boolean operator that takes a single Boolean operand and returns its inverse.
=, !=	Any	Same as left	BOOLEAN	Equality.
<,>,<=,>=	INT,FLOAT	INT,FLOAT	BOOLEAN	Numeric comparison.
<,>,<=,>=	TIME	TIME	BOOLEAN	Date and time comparison
<,>,<=,>=	BOOLEAN	BOOLEAN	BOOLEAN	Boolean comparison (TRUE > FALSE).

Notes:

1. Arithmetic operations between dates use number of days as the FLOAT parameter (or result); e.g. 0.25 represents a time span of 6 hours.
2. Any operation with an ERROR argument MUST evaluate to ERROR. An EMPTY value may be compared (=,!=) against any value.
3. Appropriate casting functions (BOOL, CHAR, TIME, INT, FLOAT) MUST be applied to the parameters. If a function or an operator is applied to a data type different than shown in the above tables, the expression MUST evaluate to ERROR.

New Section: 2.7.11.7 Constraint Types

New Section: 2.7.11.7.1 Resource Constraints

OData provides a mechanism for constraining Resources by annotating the Resource. Examples of constraints are property data type, property ranges or limits and control the values that a property within a Resource may take. These constrains may also be expressed using Validation Expressions, so either MAY be used. Ideally vendors would choose a single, standard way to *transport* validation rules until OData has the ability to express both Resource and Process constraints.

New Section: 2.7.11.7.2 Process Constraints

OData *does not* provide a mechanism for process constraints on a Resource at this time. The Validation Expression language MUST be used to represent rules that define business processes, role constrained validations, sequence constraints or previous value constrains.

TODOs:

- [ ] Review and use Odata BNF atoms as they exists - ensure that we don't redefine.
- [ ] Review and remove atoms that are not used.

Please add any other items that need to be taken care of to the bottom of the list.

Impact

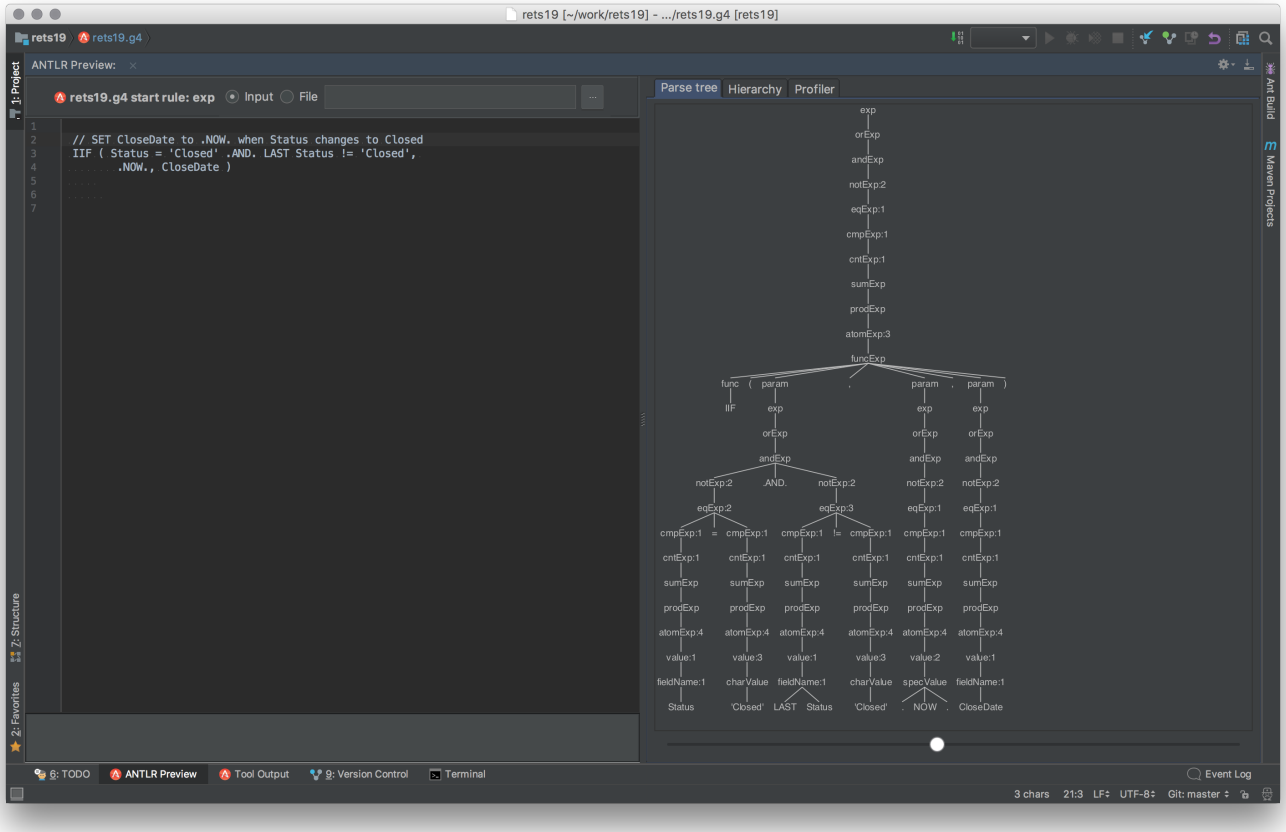
This change proposal only impacts the Upsert (Add/Modify a Resource Entity) functionality. Since this functionality did not exist in previous versions, this is not a backward-compatible change, but it will have no impact on previous versions of the standard.

Additional Information

Attached is a zip containing a working minimal implementation of this BNF in an ANTLR .g4 file, along with a Parser generated in Java. There is a README included with some example expressions.

RETS-19.zip

Here is a screenshot showing a sample expression and a parse tree:



Certification Impact

Here are some ideas for testing rules:

Rule Testing

With all the various ways to process rules, it's likely important to create a Rules testing reference implementation so servers, clients, and “fancy” clients would all have a single source of truth to check their rules implementations against. In my view this is somewhat critical so that all clients agree on what rules mean, regardless of their particular implementations.

What does testing look like? Perhaps creating a set of common rules that have some degree of interdependence and ordering.

Rule Testing Should:

- Exercise simple independent rule validation on a payload with .SET_REQUIRED. operations and only constant-valued expressions. For instance, set 10 fields to be .SET_REQUIRED., .TRUE.
- Add independent rules with .USERLEVEL. constraints, for instance, set 10 fields as required, and 5 as required if the user is not an admin.
- Add independent rules which are to be conditionally executed based on .UPDATEACTION.
- Add initialization rules using the .SET_DEFAULT. action for .UPDATEACTION. = "ADD"
- Add independent SET rules for fields not already referenced in any previous rules, and not to be referenced afterwards.
- Add dependent SET rules based on some of the fields that were initialized previously.
- Add conditionally dependent SET rules (using IIF).
- Add computations with 3-5 dependencies in the rule set, some conditional.
- Add reciprocal calculations -- tests proper reevaluation and sequencing.