

OPC Unified Architecture

Specification

AMENDMENT X:  
ExtendedTranslateBrowsePathsToNodeIds

Draft 1.04.01

July 04, 2018

|  |  |  |  |
| --- | --- | --- | --- |
| Specification Type: | Industry Standard Specification | Comments: | Report or view errata: http://www.opcfoundation.org/errata |
|  |  |  |  |
| Title: | OPC Unified Architecture  Amendment X ExtendedTranslateBrowsePathsToNodeIds | Date: | July 04, 2018 |
|  |  |  |  |
| Version: | Draft 1.04 | Software: | MS-Word |
|  |  | Source: | OPC UA Amendment X Extended TranslateBrowsePathsToNdodeIds Draft 1.04.01.docx |
|  |  |  |  |
| Author: | OPC Foundation | Status: | Draft |
|  |  |  |  |

OPC Foundation

\_\_\_\_\_\_\_\_\_\_\_\_

UNIFIED ARCHITECTURE –

FOREWORD

This specification is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

**Copyright © 2006-2018, OPC Foundation, Inc.**

AGREEMENT OF USE

COPYRIGHT RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

OPC Foundation members and non-members are prohibited from copying and redistributing this specification. All copies must be obtained on an individual basis, directly from the OPC Foundation Web site  
H[TUhttp://www.opcfoundation.orgUT](http://www.opcfoundation.org)H.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OPC specifications may require use of an invention covered by patent rights. OPC shall not be responsible for identifying patents for which a license may be required by any OPC specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OPC specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

WARRANTY AND LIABILITY DISCLAIMERS

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OPC FOUDATION MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OPC FOUNDATION BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you.

RESTRICTED RIGHTS LEGEND

This Specification is provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor / manufacturer are the OPC Foundation,. 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

COMPLIANCE

The OPC Foundation shall at all times be the sole entity that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

Trademarks

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: H[TUhttp://www.opcfoundation.org/errataUT](http://www.opcfoundation.org/errata)H

Revision 1.04 Amendment 2 Highlights

The following table includes the Mantis issues resolved with this revision.

|  |  |  |
| --- | --- | --- |
| **Mantis ID** | **Summary** | **Resolution** |
|  |  |  |

OPC Unified Architecture Specification

AMENDMENT X: Extended TranslateBrowsePathsToNodeIds

**Part 4 – Services**

Add the following after 5.8.4. TranslateBrowsePathsToNodeIds:

5.8.5 ExtendedTranslateBrowsePathsToNodeIds

5.8.5.1 Description

This Service is used to request that the Server translates one or more browse paths to NodeIds. Each browse path is constructed of a starting Node and a RelativePath. The specified starting Node identifies the Node from which the RelativePath is based. The RelativePath contains a sequence of ReferenceTypes and query patterns.

A query pattern filters resulting nodes for each translate step. Like xpath predicates is it possible to use a second relative path to address another entity to decide whether the current node will be filtered out or not.

One purpose of this Service is to allow programming against type definitions. Since BrowseNames shall be unique in the context of type definitions, a Client may create a browse path that is valid for a type definition and use this path on instances of the type. For example, an ObjectType “Boiler” may have a “HeatSensor” Variable as InstanceDeclaration. A graphical element programmed against the “Boiler” may need to display the Value of the “HeatSensor”. If the graphical element would be called on “Boiler1”, an instance of “Boiler”, it would need to call this Service specifying the NodeId of “Boiler1” as starting Node and the BrowseName of the “HeatSensor” as browse path. The Service would return the NodeId of the “HeatSensor” of “Boiler1” and the graphical element could subscribe to its Value Attribute.

If a Node has multiple targets with the same BrowseName, the Server shall return a list of NodeIds. However, since one of the main purposes of this Service is to support programming against type definitions, the NodeId of the Node based on the type definition of the starting Node is returned as the first NodeId in the list.

5.8.5.2 Parameters

Table 43 defines the parameters for the Service.

Table 43 – TranslateBrowsePathsToNodeIds Service Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| **Request** |  |  |
| requestHeader | RequestHeader | Common request parameters (see 7.28 for *RequestHeader* definition). |
| browsePaths [] | BrowsePath | List of browse paths for which *NodeIds* are being requested. This structure is defined in-line with the following indented items. |
| startingNode | NodeId | *NodeId* of the starting *Node* for the browse path. |
| relativePath | ExtendedRelativePath | The path to follow from the *startingNode*.  The last element in the *extendedRelativePath* shall always have a *targetName* specified. This further restricts the definition of the RelativePath type. The *Server* shall return *Bad*\_*BrowseNameInvalid* if the *targetName* is missing.  The *extendedRelativePath* structure is defined in 7.27. |
|  |  |  |
| **Response** |  |  |
| responseHeader | ResponseHeader | Common response parameters (see 7.29 for *ResponseHeader* definition). |
| results [] | BrowsePathResult | List of results for the list of browse paths. The size and order of the list matches the size and order of the *browsePaths* request parameter. This structure is defined in-line with the following indented items. |
| statusCode | StatusCode | *StatusCode* for the browse path (see 7.34 for *StatusCode* definition). |
| targets [] | BrowsePathTarget | List of targets for the *relativePath* from the *startingNode*. This structure is defined in-line with the following indented items.  A *Server* may encounter a *Reference* to a *Node* in another *Server* which it cannot follow while it is processing the *RelativePath*. If this happens the *Server* returns the *NodeId* of the external *Node* and sets the *remainingPathIndex* parameter to indicate which *RelativePath* elements still need to be processed. To complete the operation the *Client* shall connect to the other *Server* and call this service again using the target as the *startingNode* and the unprocessed elements as the *relativePath*. |
| targetId | ExpandedNodeId | The identifier for a target of the *RelativePath*. |
| remainingPathIndex | Index | The index of the first unprocessed element in the *ExtendedRelativePath*.  This value shall be equal to the maximum value of *Index* data type if all elements were processed (see 7.13 for *Index* definition). |
| diagnosticInfos [] | DiagnosticInfo | List of diagnostic information for the list of browse paths (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *browsePaths* request parameter. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in processing of the request. |

5.8.4.3 Service results

Table 41 defines the Service results specific to this Service. Common StatusCodes are defined in 7.34.

Table 41 – TranslateBrowsePathsToNodeIds Service Result Codes

|  |  |
| --- | --- |
| Symbolic Id | Description |
| Bad\_NothingToDo | See Table 177 for the description of this result code. |
| Bad\_TooManyOperations | See Table 177 for the description of this result code. |

5.8.4.4 StatusCodes

Table 42 defines values for the operation level statusCode parameters that are specific to this Service. Common StatusCodes are defined in Table 178.

Table 42 – TranslateBrowsePathsToNodeIds Operation Level Result Codes

|  |  |
| --- | --- |
| Symbolic Id | Description |
| Bad\_NodeIdInvalid | See Table 178 for the description of this result code. |
| Bad\_NodeIdUnknown | See Table 178 for the description of this result code. |
| Bad\_NothingToDo | See Table 177 for the description of this result code.  This code indicates that the relativePath contained an empty list. |
| Bad\_BrowseNameInvalid | See Table 178 for the description of this result code.  This code indicates that a TargetName was missing in a RelativePath. |
| Uncertain\_ReferenceOutOfServer | The path element has targets which are in another server. |
| Bad\_TooManyMatches | The requested operation has too many matches to return.  Users should use queries for large result sets. *Servers* should allow at least 10 matches before returning this error code. |
| Bad\_QueryTooComplex | The requested operation requires too many resources in the server. |
| Bad\_NoMatch | The requested relativePath cannot be resolved to a target to return. |

Add the following as 7.27. ExtendedRelativePath:

**7.27 ExtendedRelativePath**

The components of this parameter are defined in Table 169.

**Table 169 – ExtendedRelativePath**

|  |  |  |
| --- | --- | --- |
| Name | page164image5853696Typepage164image5856400page164image5856608 | Description |
| ExtendedRelativePath | structure | Defines a sequence of *References* and *BrowseNames* to follow. |
| elements [] | ExtendedRelativePath Element | A sequence of *References* and *query patterns* to follow. This structure is defined in-line with the following indented items. Each element in the sequence is processed by finding the targets and then using those targets as the starting nodes for the next element. The targets of the final element are the target of the *ExtendedRelativePath*. |
| referenceTypeId | NodeId | The type of reference to follow from the current node. The current path cannot be followed any further if the referenceTypeId is not available on the Node instance. If not specified then all *References* are included and the parameter includeSubtypes is ignored. |
| isInverse | Boolean | Only inverse references shall be followed if this value is TRUE. Only forward references shall be followed if this value is FALSE. |
| includeSubtypes | Boolean | Indicates whether subtypes of the *ReferenceType* should be followed. Subtypes are included if this value is TRUE. |
| browseNamePattern | QualifiedName | Browsename included with wildcards “\*”, “?”. & is escape character:  ‘\*’: Zero, or more characters  ‘?’: One character  ‘&’ escape character |
| predicates[] | PredicatePathStep | Predicates path starting from the current target node, See 7.28 |

An *ExtendedRelativePath* can be applied to any starting *Node*. The targets of the *ExtendedRelativePath* are the set of *Nodes* that are found by sequentially following the elements in *ExtendedRelativePath*.

The PredicatePath for a current target node is a filter that decides whether this current node will be removed or remain in the set of target nodes for the next translate step.

The PredicatePath is an array of PredicateSteps (predicates[]).

A text format for the *ExtendedRelativePath* can be found in Clause A.2. This format is used in examples that explain the *Services* that make use of the *ExtendedRelativePath* structure.

Samples for BrowseNamePattern:

|  |  |
| --- | --- |
| Boiler | All TargetNodes with the BrowseName Boiler |
| Boiler\* | All TargetNodes with the BrowseName starting with Boiler |
| Bo?ler | All strings that Starts with “Bo” followed by one character and than ends with ”ler”. |
| Bo&?ler | Bo?ler |
| Bo&&ler | Bo&ler |

**7.28 PredicatePathStep**

Each PredicateStep contains a processing instruction and additional parameters depending on the processing instruction. In difference to the RelativePath each PredicateStep should result in zero or one target nodes.

Possible Processing instructions are:

|  |  |  |
| --- | --- | --- |
| Name | page164image5854112Parameter | Description |
| Follow | Reference | Follow a reference for the next step |
| And | None | Split predicate path to a left and right side and combine the results using a logical and:  Store the current result, compute the next steps until an And or Or instruction or the end is reached. Than compute an logical and on both results. |
| Or | None | Split predicate path to a left and right side and combine the results using a logical or:  Store the current result, compute the next steps until an Or instruction or the end is reached. Than compute an logical or on both results. |
| BracketStart | None | Call a new predicate Path procession beginning with the next predicate path step. (this call recursive the predicate path processing) |
| BracketEnd | None | End the current predicate path processing, calculate the result and return this as processing result to the caller. |
| Expression | Expression | Attribute Comparative Operator Regular Expression or Number |

The components of the PredicatePathStep are defined in Table 170.

## U MonitoringMode

The *MonitoringMode* is an enumeration that specifies whether sampling and reporting are enabled or disabled for a *MonitoredItem*. The value of the publishing enabled parameter for a *Subscription* does not affect the value of the monitoring mode for a *MonitoredItem* of the *Subscription*. The values of this parameter are defined in Table 146.

Table 146 – MonitoringMode Values

|  |  |
| --- | --- |
| Value | Description |
| DISABLED\_0 | The item being monitored is not sampled or evaluated, and *Notifications* are not generated or queued. *Notification* reporting is disabled. |
| SAMPLING\_1 | The item being monitored is sampled and evaluated, and *Notifications* are generated and queued. *Notification* reporting is disabled. |
| REPORTING\_2 | The item being monitored is sampled and evaluated, and *Notifications* are generated and queued. *Notification* reporting is enabled. |

**Table 170 – PredicatePathStep**

|  |  |  |
| --- | --- | --- |
| Name | page164image5853696page164image5854112Typepage164image5856400page164image5856608 | Description |
| Processing Instruction | enum | An enumeration that specifies the processing instruction for this predicate step. It has the following values:  FOLLOW\_0 Follow the reference.  AND\_1 logical and  OR\_2 logical or  BRACKET\_START\_3 Start a subprocessing  BRACKET\_END\_4 End of subprocessing  EXPRESSION\_5 Compute expression |
| referenceTypeId | NodeId | Direction to follow beginning from the current target node or the result of the last step. Used for Processing instruction FOLLOW\_0. |
| isInverse | Boolean | Only inverse references shall be followed if this value is TRUE. Only forward references shall be followed if this value is FALSE. . Used for Processing instruction FOLLOW\_0. |
| includeSubtypes | Boolean | Indicates whether subtypes of the ReferenceType should be followed. Subtypes are included if this value is TRUE. . Used for Processing instruction FOLLOW\_0. |
| AttributeId | UInt32 | Attribute of the current predicate node to compare. (for attribute ids of Attributes, see Part 6) . Used for Processing instruction EXPRESSION\_5. |
| Comparator | enum | An enumeration that specifies the processing instruction for this predicate step. It has the following values:  LESS\_THAN\_0 “<”, for Numbers only  GREATER\_THAN\_1 “>”, for Numbers only LESS\_OR\_EQUAL\_2 “<=”, for Numbers only GREATER\_OR\_EQUAL\_3 “>=”, for Numbers only EQUAL\_4 “=”  NOT\_EQUAL\_5 “!=”  Used for Processing instruction EXPRESSION\_5. |
| RegularExpression | String | Regular expression that is operated on the attribute value. Used for Processing instruction EXPRESSION\_5. |
| Number | Number | Number the attribute value copmpared to. Used for Processing instruction EXPRESSION\_5. |

Remarks:

Whether the attribute is compared to a regular expression or a number depends on the type of the attribute. Attributes of type string are computed using a regular expression. Attributes of type Number are computed using a number.

If the type of the attribute don’t match the comparator or the “right side” an error is returned.

Exchange the following as A.2. RelativePath:

**BNF of ExtendedRelativePath**

A *ExtendedRelativePath* is a structure that describes a sequence of *References* and *Nodes* to follow. This annex describes a text format for a *RelativePath* that can be used in documentation or in files used to store configuration information.

The components of an Extended*RelativePath* text format are specified in Table A.1.

**Table A.1 – ExtendedRelativePath**

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| / | The forward slash character indicates that the *Server* is to follow any subtype of *HierarchicalReferences*. |
| . | The period (dot) character indicates that the *Server* is to follow any subtype of a *Aggregates ReferenceType*. |
| <[#!ns:]ReferenceType> | A string delimited by the ‘<’ and ‘>’ symbols specifies the *BrowseName* of a *ReferenceType* to follow. By default, any *References* of the subtypes the *ReferenceType* are followed as well. A ‘#’ placed in front of the BrowseName indicates that subtypes should not be followed.  A ‘!’ in front of the BrowseName is used to indicate that the inverse *Reference* should be followed. The *BrowseName* may be qualified with a namespace index (indicated by a numeric prefix followed by a colon). This namespace index is used specify the namespace component of the *BrowseName* for the *ReferenceType.* If the namespace prefix is omitted then namespace index 0 is used. |
| [ns:]BrowsePattern‘[‘PredicatePath‘]‘ | A string that follows a ‘/’, ‘.’ or ‘>’ symbol specifies the pattern of a *BrowseName* of a target *Node* to return or follow. This BrowsePattern may be prefixed by its namespace index. If the namespace prefix is omitted then namespace index 0 is used. The BrowsePattern can be followed by a PredicatePath in brackets “[“,”]”. A PredicatePath can filter the set of target nodes.  Omitting the final Browse*Pattern and PredicatePath* from a path is equivalent to a wildcard operation that matches all *Nodes* which are the target of the *Reference* specified by the path.  Syntax of the pattern is defined in 7.27. Syntax of the PredicatePath is defined in A.2 |
| & | The & sign character is the escape character. It is used to specify reserved characters that appear within a *pattern*. A reserved character is escaped by inserting the ‘&’ in front of it. Examples of *patterns* with escaped characters are:  Received browse path name “&/Name\_1” “&.Name\_2” “&:Name\_3”  “&&Name\_4”  “&”Name\_5”  Resolves to “/Name\_1” “.Name\_2” “:Name\_3” “&Name\_4” “”Name\_5” |

Table A.2 provides *RelativePaths* examples in text format.

OPC Unified Architecture, Part 4 163 Release 1.04

**Table A.2 – *BNF of PredicatesPath***

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| PredicatePath | PredicatePathStep |  PredicatePathStep PredicatePath |
| PredicatePathStep | Reference |  Expression |
| Reference | <[#!ns:]ReferenceType> |
| Expression | AttributeName Comparator RegularExpression |  AttributeName Comparator Number |
| AttributeName | ‚@‘ String  String ist the name oft he attribute to use |
| Comparator | ‚<‘,‘>‘,‘<=‘,‘>=‘,‘=‘,‘!=‘ |
| Regular Expression | ‚“‘ Regular Expression ‚“‘  ToDo find reference to standardization body for regular expressions. <https://en.wikipedia.org/wiki/Regular_expression>  The regular expression has to be wrapped in quotes (“). |
| Number | A floating point value, see part 6 |
| <[#!ns:]ReferenceType> | A string delimited by the ‘<’ and ‘>’ symbols specifies the BrowseName of a ReferenceType to follow. By default, any References of the subtypes the ReferenceType are followed as well. A ‘#’ placed in front of the BrowseName indicates that subtypes should not be followed.  A ‘!’ in front of the BrowseName is used to indicate that the inverse Reference should be followed. The BrowseName may be qualified with a namespace index (indicated by a numeric prefix followed by a colon). This namespace index is used specify the namespace component of the BrowseName for the ReferenceType. If the namespace prefix is omitted then namespace index 0 is used. |
| & | The & sign character is the escape character. It is used to specify reserved characters that appear within a pattern. A reserved character is escaped by inserting the ‘&’ in front of it. Examples of patterns with escaped characters are:  Received browse path name “&/Name\_1” “&.Name\_2” “&:Name\_3”  “&&Name\_4”  “&”Name\_5”  Resolves to “/Name\_1” “.Name\_2” “:Name\_3” “&Name\_4” “”Name\_5” |

**Table A.3 – Extended*RelativePath* Examples**

|  |  |
| --- | --- |
| **Browse Path** | **Description** |
| “/2:Block&.Output” | Follows any forward hierarchical *Reference* with target *BrowseName* = “2:Block.Output”. |
| “/3:Truck.0:Node\*” | Follows any forward hierarchical *Reference* with target *BrowseName* = “3:Truck” and from there a forward *Aggregates Reference* to a target with any *BrowseName beginning with* “0:Node”. |
| “<1:ConnectedTo>1:Boil\*/1:HeatSensor” | Follows any forward Reference with a *BrowseName* = ‘1:ConnectedTo’ and finds targets with *BrowseName beginning with Boild in namespaceindex 1*. From there follows any hierarchical *Reference* and find targets with *BrowseName* = ‘1:HeatSensor’. |
| “<0:HasChild>2:Wheel” | Follows any forward Reference with a *BrowseName* = ‘HasChild’ and qualified with the default OPC UA namespace. Then find targets with *BrowseName* = ‘Wheel’ qualified with namespace index ‘2’. |
| “<!HasChild>Truck” | Follows any inverse Reference with a *BrowseName* = ‘HasChild’. Then find targets with *BrowseName* = ‘Truck’. In both cases, the namespace component of the *BrowseName* is assumed to be 0. |
| “<1:ConnectedTo>1:Boiler\*[<1:hasFixatation>  1:MountMode@value=”Screwed”]/  1:CurrentTemperature” | Follows any forward Reference with a *BrowseName* = ‘1:ConnectedTo’ and finds targets with *BrowseName* begins with ‘1:Boiler’. All found targets filtered for having a forward reference with BrowseName = ‘1:hasFixatation’ that point to a node with the BrowseName = ‘1:MountMode and the value attribute = “Screwed”. From the resulting nodes it finds all targets of hierarchical *References which have the BrowseName = ‘1:CurrentTemperature’* |
| “<!hasTypeDefinition>[<1:MountedTo>  <1:hasFixatation>1:Fixatation@value=”Screwed”]” | Follows any reverse Reference with a BrowseName = ‘hasTypeDefinition’. All found targets filtered for having a forward reference with BrowseName = ‘1:MountedTo’ that point to a node having a forward reference with the BrowseName = ‘1:hasFixatation’ that point to a node with the BrowseName = ‘1:Fixatation’ and the value attribute = “Screwed”.  This ExtendedRelativePath returns all instances of the starting Typedefinition Node which have a MountedTo Reference, and that targets have a hasFixatation reference with a Fixatation node and the value Screwed. |
| “<0:HasChild>”  page183image37244624 | Finds all targets of forward References with a BrowseName = ‘HasChild’ and qualified with the default OPC UA namespace. |

The following BNF describes the syntax of the Extended*RelativePath* text format.

<relative-path> ::= [<reference-type>] [<browse-name>] [<predicates-path>] [relative-path] [[1]](#footnote-1)

<reference-type> ::= '/' | '.' | '<' ['#'] ['!'] <browse-name> '>'

<browse-name> ::= [<namespace-index> ':'] <name>

<namespace-index> ::= <digit> [<digit>]

<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<name> ::= (<name-char> | '&' <reserved-char> | '.' | '\*') [<name>]

<predicates-path> ::= '[' <predicates-step> ']'

<predicates-step> ::= [<reference-type>] [<expression>] [<predicates-step>]

<expression> ::= '@' <name> <comparator> <regular-expression-string> | <Number>

<comparator> ::= '<' | '>' |'<=' | '>=' |'!=' | '='

<Number> ::= ['+' | '-'] [<digit-sequence>] [.] [<digit-sequence>]

<digit-sequence> ::= <digit> [<digit-sequence>]

<regular-expression-string> ::= '”' < regular-expression > '”'

<regular-expression> ::= t.b.d. from Regexp definition

<reserved-char> ::= '/' | '.' | '<' | '>' | ':' | '#' | '!' | '&'| '\*'

| '['| ']'

<name-char> ::= All valid characters for a String (see Part 3) excluding reserved-chars.

1. All elements of the <relative-path> are optional, but it hast o contain at least one optional element [↑](#footnote-ref-1)