3.20 RegistryPackageSelector

The canonical query RegistryPackageSelector allows clients to create a Subscription to a remote server to replicate a remote RegistryPackage as well as all its member objects and the AssociationType instances that relate the members of the RegistryPackage to it. This query MAY be used as Selector query within the Subscription for the replication as defined in the object replication feature.

3.20.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
registryPackageId	Matches rim:Registry/@id. Does not allow wildcards.	string		1

3.20.2 Query Semantics

- The server MUST return the specified RegistryPackageType instance, all RegistryObjectType instances that are members of the specified RegistryPackage as well as all "HasMember" AssociationType instances between the RegistryPackageType instance and its members. that are descendants of that ClassificationScheme.
- The member RegistryObjectType instances MUST NOT be returned as nested elements inside the RegistryPackage. Instead they MUST be returned as sibling elements with the RegistryPackage and Associations within the RegistryObjectList element of the QueryResponse.

3.21 Query Functions

A server MAY support any number of pre-defined functions known as *Query Functions*, that may be used within a query expression or query parameter. Query functions are similar in concept to functions in SQL. Query functions may be used within the query expression of a parameterized query as well as within its invocation parameter values. Query functions enable parameterized queries to use reusable specialized search algorithms to augment their capabilities.

This specification defines a number of canonical functions that are standard functions that MUST be supported by a server. Profiles, implementations and deployments may define additional query functions beyond the canonical functions defined by this specification.

3.21.1 Using Functions in Query Expressions

A parameterized query stored as a rim:QueryDefinition instance MAY have a rim:QueryExpression which defines defines a query expression within its sub-nodes. A client MAY submit a rim:QueryDefinition such that its query expression may use any number of query functions supported by the server any where within the query expression where it is syntactically correct to use value returned by the function.

If a query expression contains one or more function invocations then the query expression MUST delimit the parts of the query expression that are not a function invocation with the leading characters "#@" and trailing characters "@#". This is similar in syntax to a java multi-line comment syntax where a comment is delimited by leading characters "/*" and trailing characters "*/". The delimiters serve the following purposes:

- Allows a parser to recognize the non-function parts of the query expression that MUST be preserved *as is*
- Allows implementations to optimize by skipping function parsing and evaluation if the special delimiter characters are not present in query expression

The following is an example of an SQL query expression which uses the subClassificationNode function to match all RegistryObjects that are targets of Association with specified sourceObject and type that a subnode of AffiliatedWith node upto a depth of 2 levels in the descendant hierarchy. The delimiter characters are in bold font while the function invocations is in bold and italic font below:

```
--example of a query expression with query functions
#@SELECT targetObject.* FROM
RegistryObjectType targetObject, AssociationType a WHERE

  a.sourceObject = :sourceObject AND
  a.type IN (@# subClassificationNode('urn:oasis:names:tc:ebxml-
regrep:AssociationType:AffiliatedWith', 2, ",") #@) AND
  targetObject.id = a.targetObject@#
```

3.21.2 Using Functions in Query Parameters

A client MAY use query functions supported by a server within parameter values specified when invoking a parameterized query. A client MAY invoke a parameterized query using the Query protocol such that its query parameter values may use any number of query functions supported by the server any where within the query parameter where it is syntactically correct to use value returned by the function.

If a query parameter value contains one or more function invocations then the query expression MUST delimit the parts of the query parameter that are not a function invocation with the leading characters "#@" and trailing characters "@#". If a query parameter value only has function invocations and contains no non-function parts then it must include at least one leading or trailing "#@@#" delimiter token pair to allow optimized parsing and evaluation of query functions only when needed.

The following is an example of a query expression that has no query functions. Its two parameters are shown in bold font:

```
--Following is the query expression within the server
--This time it has no query functions as they are in the query parameters
SELECT targetObject.* FROM
RegistryObjectType targetObject, AssociationType a WHERE

  a.sourceObject = :sourceObject AND
  a.type IN ( :types ) AND
  targetObject.id = a.targetObject
```

The following is an example of invocation of a parameterized query that uses the above query expression and uses the subClassificationNode function from previous example within the value of the *types* parameter. Note the trailing "#@@#" delimiter tokens are present as required.

```
<query:QueryRequest maxResults="-1" startIndex="0" ...>
    <rs:ResponseOption returnComposedObjects="true"
returnType="LeafClassWithRepositoryItem"/>
    <query:Query queryDefinition="urn:acme:ExampleQuery">
        <rim:Slot name="sourceObject">
```

```
            <rim:ValueList>
                <rim:ValueListItem xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                    <rim:Value>urn:test:Person:Danyal</rim:Value>
                </rim:ValueListItem>
            </rim:ValueList>
        <rim:Slot name="types">
            <rim:ValueList>
                <rim:ValueListItem xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                    <rim:Value>subClassificationNode('urn:oasis:names:tc:ebxm
l-regrep:AssociationType:AffiliatedWith', 2, ",")#@@#</rim:Value>
                </rim:ValueListItem>
            </rim:ValueList>        </rim:Slot>
    </query:Query>
</query:QueryRequest>
```

3.21.3 Function Processing Model

A server MUST meet the following function processing requirements during the processing of a QueryRequest:

- When processing a query expression elements (rim:QueryDefinition/rim:QueryExpression) the server SHOULD NOT perform function processing if the special delimiter sequences of “#@” and “@#” are not found in the query expression
- When processing query invocation parameter elements (query:QueryRequest/query:Query/rim:Slot/rim:ValiueList/rim:ValueListItem) the server SHOULD NOT perform function processing if the special delimiter sequences of “#@” and “@#” are not found in the query expression
- When processing a query expression element if the special delimiter sequences of “#@” and “@#” are found then the server MUST process query expression elements to replace all function invocations with the value returned when the function is invoked with specified parameters
- When processing query invocation parameter elements if the special delimiter sequences of “#@” and “@#” are found then the server MUST process each query parameter element to replace all function invocations with the value returned when the function is invoked with specified parameters
- When invoking a function that has another function invocation as its parameter the inner most functions MUST be invoked first so that the outer function can be invoked with the value returned by the inner function invocation
- When processing a query expression or query parameter the special delimiter characters “#@” and “@#” MUST be removed and the value contained within them MUST be preserved without any change

3.21.4 Function Processor BNF

The following BNF grammar normatively describes the grammar for query expressions and query invocation parameters with embedded function invocations. The ***start*** production describes the grammar for query expressions and query invocation parameters with embedded function invocations.

```
<DEFAULT> SKIP : {
" "
| "\t"
| "\r"
| "\n"
}

<DEFAULT> TOKEN : {
<S_NUMBER: <FLOAT> | <FLOAT> (["e","E"] (["-","+"])? <FLOAT>)?>
| <#FLOAT: <INTEGER> | <INTEGER> ("." <INTEGER>)? | "." <INTEGER>>
| <#INTEGER: (<DIGIT>)+>
| <#DIGIT: ["0"-"9"]>
| <BOOLEAN: "true" | "false">
}

<DEFAULT> TOKEN : {
<S_IDENTIFIER: (<LETTER>)+ (<DIGIT> | <LETTER> | <SPECIAL_CHARS>)*>
| <#LETTER: ["a"-"z","A"-"Z"]>
| <#SPECIAL_CHARS: "_">
| <S_CHAR_LITERAL: "\'" (~["\'"])* "\'" ("\'" (~["\'"])* "\'")*>
| <S_QUOTED_IDENTIFIER: "\"" (~["\n","\r","\""])* "\"">
| <OPENPAREN: "(">
| <CLOSEPAREN: ")">
| <COMMA: ",">
| <COLON: ":">
| <DELIMITED_TEXT: "#@" (~["@"])* "@#">
}

start  ::=    ( textOrFunctionCall )+ <EOF>
text   ::=    ( ( <DELIMITED_TEXT> ) )
textOrFunctionCall  ::=    ( text | FunctionCall )
FunctionCall ::=   FunctionReference <OPENPAREN> FunctionArgumentList
<CLOSEPAREN>
FunctionReference   ::=    <S_IDENTIFIER> <COLON> <S_IDENTIFIER>
FunctionArgumentList   ::=   FunctionArgument ( <COMMA> FunctionArgument )*
FunctionArgument    ::=    ( FunctionCall | <S_CHAR_LITERAL> |
<S_QUOTED_IDENTIFIER> | <S_NUMBER> | <BOOLEAN> )
```

3.22 Canonical Functions

This section defines a set of standard canonical queries that MUST be supported by all servers. A client MAY use these functions within a query expression or within the value of a parameter to a parameterized query. A server MUST process the functions according to their behavior as specified in this section. The function processing model is specified in Function Processing Model.

A client MUST use the “ebrs:” namespace prefix when using a canonical function defined by this profile.

Function Name	Semantics
ebrs:subClassificationNodes	Returns descendant classification nodes of specified node up to specified depth
ebrs:superClassificationNodes	Returns ancestor classification nodes of specified node up to specified depth

Table 3: Canonical Functions Defined By This Profile

3.22.1 Canonical Function: subClassificationNode

This canonical function takes an ClassificationNode's id as parameter and returns all ClassificationNode's that are descendants of the specified ClassificationNode and within the specified number of generations as indicated by the depth parameter.

3.22.1.1 Parameter Summary

Parameter	Description	Data Type
classificationNodeId	The value of this parameter specifies the id of a ClassificationNodeType instance	string
depth	Specifies how many generations deep to match descendants	integer
delimiter	The value of this parameter specifies the delimiter string to be used as separator between the tokens representing the ids matched by the function	string

3.22.1.2 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The string MUST be “ebrs:null” if no ClassificationNode is found that is a descendant of specified ClassificationNode within the specified depth or if specified ClassificationNode does not exist
- The string MUST consist of a set of ClassificationNode's ids separated by the appropriate delimiter character when any ClassificationNode's are found hat are descendant of specified ClassificationNode within the specified depth
- A depth of N where N > 0 matches the Nth generation descendants of the specified ClassificationNode. For example a depth of 1 matches the immediate children of the specified ClassificationNode while a depth of 2 matches the grandchildren of the specified ClassificationNode
- A depth of -1 matches all descendants of the specified ClassificationNode

3.22.2 Canonical Function: superClassificationNode

This canonical function takes an ClassificationNode's id as parameter and returns all ClassificationNode's that are ancestors of the specified ClassificationNode aand within the specified number of generations as indicated by the depth parameter.

3.22.2.1 Parameter Summary

Parameter	Description	Data Type
classificationNodeId	The value of this parameter specifies the id of a ClassificationNodeType instance	string

depth	Specifies how many generations deep to match ancestors	integer
delimiter	The value of this parameter specifies the delimiter string to be used as separator between the tokens representing the ids matched by the function	string

3.22.2.2 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The string MUST be “ebrs:null” if no ClassificationNode is found that is a ancestor of specified ClassificationNode within the specified depth or if specified ClassificationNode does not exist
- The string MUST consist of a set of ClassificationNode's ids separated by the appropriate delimiter character when any ClassificationNode's are found hat are ancestor of specified ClassificationNode within the specified depth
- A depth of N where N > 0 matches the Nth generation ancestors of the specified ClassificationNode. For example a depth of 1 matches the immediate parents of the specified ClassificationNode while a depth of 2 matches the grandparents of the specified ClassificationNode
- A depth of -1 matches all ancestors of the specified ClassificationNode