

Specification

Table of Contents

1	Requirements and Design Principles.....	7
1.1	Query Language and Results Format	7
1.2	Synchronous and Asynchronous Query Mode.....	7
1.3	Command-Query Separation Principle	8
1.4	Simple Command Set and Extensibility.....	9
2	API Specification	10
2.1	Overview	10
2.2	Query Configuration	11
2.2.1	Set Query Language	11
2.2.2	Set Maximum Number of Query Results	11
2.2.3	Set Maximum Duration	12
2.2.4	Set Results Format	12
2.3	Synchronous Query Methods	12
2.3.1	Set Results Set Size	12
2.3.2	Synchronous Query	13
2.3.3	Get Additional Query Results	13
2.3.4	Get Total Results Count	14
2.4	Asynchronous Query Methods.....	14
2.4.1	Set Source Location	14
2.4.2	Asynchronous Query	14
2.4.3	Query Results Listener	15
2.5	Results Retrieval	15
2.5.1	Get Resource Description.....	15
3	Open Work Items	16

About this Section

Title:	Simple Query Interface Specification
Editor:	Bernd Simon (bernd.simon@wu-wien.ac.at) David Massart (david.massart@eun.org) Erik Duval (erik.duval@cs.kuleuven.ac.be)
Status:	Public Draft
Version (Date):	0.8 (2004-06-17)
Document Location:	http://rubens.cs.kuleuven.ac.be/vqwiki-2.5.5/jsp/Wiki?LorInteroperability

Acknowledgements

This work is partly sponsored by the CEN/ISSS Workshop on Learning Technologies. We also would like to acknowledge contributions from the following initiatives: Ariadne, Educanext, Celebrate, Edutella, Elena, EduSource, ProLearn, Universal, Zing.

List of Contributors

Bernd Simon, Vienna University of Economics and Business Administration & EducaNext
Christian Werner, Learning Lab Lower Saxony
Erik Duval, Katholieke Universiteit Leuven & Ariadne
Daniel Olmedilla, Learning Lab Lower Saxony
Dan Rehak, Carnegie Mellon University
David Massart, European Schoolnet
Frans Van Assche, European Schoolnet
Griff Richards, Simon Fraser University
Gerhard Müller, IMC
Julien Tane, Universität Karlsruhe
Marek Hatala, Simon Fraser University
Matthew J. Dovey, Oxford University
Peter Dolog, Learning Lab Lower Saxony
Sascha Markus, IMC
Stefaan Ternier, Katholieke Universiteit Leuven
Stefano Ceri, Politecnico Milano
Stefan Brantner, BearingPoint Infonova
Simos Retalis, University of Piraeus
Theo van Veen, Koninklijke Bibliotheek
Zoltán Miklós, Vienna University of Economics and Business Administration

Revision History

<i>Version</i>	<i>Changes</i>
0.8	<p>This document focuses now solely on the Query API. As a consequence the Related Work Section, the Session and Authentication Management, as well as the section describing the VSQI Profile were removed.</p> <p>Additionally the following changes were made:</p> <ul style="list-style-type: none">• Per default, time-out management is delegated to the target (i.e., default value for maximum query duration is 0, previously it was set to unrealistic 500 milliseconds).• The <code>InvalidLocationException</code> was removed since it is unlikely to be called in case of an invalid location error.• All “NoValid...Excepetions” were changed to “Invalid... Exceptions”.• For <code>setSourceLocation</code> and <code>asynchronousQuery</code> the name of the exception that is thrown in case the asynchronous query is not supported has been changed from <code>WrongQueryModeException</code> to <code>QueryModeNotSupportedException</code>.• Textual improvements of API description and introductory section.
0.7	<p><i>At the European Schoolnet side David Massart replaces Frans Van Assche as editor of this document.</i></p> <p>The following changes were introduced based on a meeting held between Daniel Olmedilla (who served as co-editor of this version), Stefaan Ternier, and Bernd Simon. Additionally, a ProLearn Workshop was held at Karlsruhe University at 13/2/2004, which helped to put this initiative better into context. Further input came from Frans Van Assche and Stefan Brantner.</p> <ul style="list-style-type: none">• The methods <code>setMaxResults</code> and <code>setMaxDuration</code> are supported by both query methods (<code>synchronousQuery</code> and <code>asynchronousQuery</code>). <code>setMaxResults</code> has been split into two methods: <code>setMaxQueryResults</code> and <code>setResultsSetSize</code>. The first controls the maximum number of results produced by a query. The later determines the default value for the number of results returned by a query with a single results-set and is only valid in the synchronous query mode.• The method <code>setQueryMode</code> has been removed. Instead the query mode specific methods return an exception in case the query mode is not supported.• The requirement for a <code>sourceSessionID</code> in the asynchronous query mode has been removed. A <code>queryID</code>, issued by the source, has been introduced instead.• <code>WrongQueryModeException</code> has been introduced for all query mode specific methods besides the results listener. All exceptions are now labelled with “Exception” at the end.• New methods have been added: <code>setQueryLanguage</code>, <code>setQuerySchema</code>, <code>setSourceLocation</code> and <code>setResultsFormat</code>. As a result the query methods have now fewer parameters. The parameters have been moved to the set methods. This approach gives easies backwards compatibility.• <code>getSupportedQueryModes</code> needs to become part of an SQI Profiling initiative. Therefore it has been removed from the current version of the specification.• In order to comply with the design principle “name follows function” the method “<code>getQueryResults</code>” is now called “<code>getAdditionalQueryResults</code>”.• Default values for the methods <code>setMaxQueryResults</code> (100), <code>setResultsSetSize</code> (25), and <code>setMaxDuration</code> (500) were defined. These values apply in case the set methods are not called before the first query execution.• At the SQI profile “Very Simple Query Interface” the requirement for the session management method was removed.• Some copy-paste errors at the method return values (String instead of Void) were

	<p>corrected.</p> <ul style="list-style-type: none"> • Introduction, requirements section and functionality description have been considerably extended. It has become more explicit that in asynchronous mode multiple queries can be active within a session while there can only one active query per session in synchronous mode.
0.6	<p><i>From this version on, the specification is jointly edited by Bernd Simon, Erik Duval, and Frans Van Assche.</i></p> <p>The following changes were introduced:</p> <ul style="list-style-type: none"> • The SQI distinguishes now between an asynchronous query method (<code>asynchronousQuery</code>) and a synchronous query method (<code>synchronousQuery</code>), the latter directly returns query results. → Revision triggered by Zoltán Miklos • The return format for the query results has become more explicit in both, the <code>query</code> method and <code>getQueryResults</code> method. Hereby, the target for mappings can be specified. → Revision triggered by Bernd Simon • Rational behind session management is now explained in more detail. Within a session only one active query may exist. Hence, there is no need anymore for a query ID, which makes the interface even simpler. A distinction between <code>targetSessionID</code> and a <code>sourceSessionID</code> has been introduced. → Revision triggered by David Massart and Daniel Ollmedia. • KEYWORD (search) was introduced as permissible value for “query language”. → Revision triggered by Erik Duval, Stefaan Ternier, and Frans Van Assche. • Query parameter SchemaReference: “UNKNOWN” is now also a valid argument. The references linking to local files have been replaced by a link to the LOM XML schema. → Revision triggered by Daniel Ollmedia. • The methods <code>getSupportedQueryLanguages</code> and <code>getSupportedSchemas</code> were removed from the specification, since it remains unclear how the system responses to the information gathered by these methods can look like. Future versions of this specification shall rather opt for some semantic descriptions of the interface (SQI repository profiles) rather than using multiple functions to find out more about the capabilities of the interface. → Revision triggered through various discussions. • Major text editing: Introduction was completely revised. Requirements and Scenarios Section as well as Limitations Section have been restructured and expanded. The Functionality Overview Section has been extended with a table. Restructuring of Syntax Section in order to better separate asynchronous and synchronous query interface commands and to clearly communicate, which commands are relevant in which query mode. Appendix B (Very Simple Query interface) has been completely revised and integrated into the document as Section 1.7 (A Very Simple Profile of SQI).
0.52	<p>An application profile for a very simple query interface has been designed in Appendix B. → Thanks to the initiative of Erik Duval.</p>
0.51	<p>The presentation of the specification has been updated. Related work on EduSource added; XPATH included as a permissible value of the <code>getSupportedQueryLanguages</code> (→ based on input from Marek Hatala). Thanks to Peter Dolog the functionality overview also includes a UML activity diagram.</p>
0.5	<p>The following changes were introduced:</p> <ul style="list-style-type: none"> • Besides the source-initiated (synchronous) query (mode) also a target-initiated (asynchronous) query (mode) was introduced (methods: <code>queryResultsListener</code>, <code>setMaxDuration</code>). → Thanks to a discussion with David Massart and Frans Van Assche. • Parameter <code>SchemaReferences</code> was introduced in the <code>Query</code> method providing a hook for data model mappings. Additionally, Method <code>getSupportedSchemas</code> was added. → Input

Version *Changes*

<i>Version</i>	<i>Changes</i>
	<p>from Zoltán Miklos.</p> <ul style="list-style-type: none">• Design assumptions and limitations were refined. → Input from Daniel Olmedilla and Stefano Ceri.• WrongCredentialsException replaced wrongUserID and wrongPasswordException in the Session Management Section. → Contribution from David Massart• “Source” and “Target” in Figure 1 were exchanged (aligned with ZING SRW).• * Related Work Section expanded by references to Edutella, CeLeBraTe and ZING SRW. → Input from Peter Dolog, David Massart, Matthew J. Dovey, Theo van Veen.
0.4	<p>The interface definition focuses now on metadata search. Free text search as “query language” is not further supported, because it requires a different type of interface (e.g. a return schema needs to be specified) and the advantages of metadata annotation are not seized by a free text search. Related work has been expanded, primitive means for session management introduced. The paper now includes a section on requirements and one on the limitations of the status quo.</p>

1 Requirements and Design Principles

In this paper an Application Program Interface (API) for querying learning objects repositories is presented. Since one major design objective is to keep the specification simple and easy to implement, the interface is labelled Simple Query Interface (SQI). The collaborative effort of combining highly heterogeneous repositories has led to the following requirements:

- SQI is neutral in terms of results format and query languages: The repositories connecting via SQI can be of highly heterogeneous nature: therefore, SQI makes no assumptions about the query language or results format.
- SQI supports Synchronous and Asynchronous Queries in order to allow heterogeneous networks to connect to each other.

The design of the API itself is based on following design principles:

- Command-Query Separation Principle,
- Simple Command Set and Extensibility.

The following sub-sections will describe each of the above mentioned items in more detail.

1.1 Query Language and Results Format

Any schema that two or more repositories have agreed on is valid for being used. An SQI schema serves two purposes: On the one hand, it identifies the set of attributes and vocabularies that can be used in the query, and on the other, it provides a format in which learning object descriptions are returned to the source. For example, both XML schemas and RDF schemas are valid in this context.

Since SQI is intended to be deployed by highly heterogeneous types of repositories (e.g., RDF engines, XML databases, relational databases, etc.) the interface cannot be based on one specific query language. Any query language chosen (e.g. XQUERY) would have a negative impact on the adoption of the interface by those repository types the language has not been designed for (e.g. RDF repositories in the case of XQUERY). Although the interface itself does not contribute to overcome the differences of the various paradigms in metadata management (Z39.50, XML-based approaches, RDF community), it aims to become an independent specification for all open educational repositories.

1.2 Synchronous and Asynchronous Query Mode

SQI can be deployed in two different scenarios.

1. In the *synchronous* scenario (fig. 3), the target returns the query results to the source. Results retrieval is therefore initiated by the source through the submission of the query and through other methods allowing the source to access the query results.
2. In the *asynchronous* scenario (fig. 4), results retrieval is target-initiated. Whenever a significant amount of matching results is found, these results are forwarded to the source by the target. To support this communication the source must a results listener. The source must be able to uniquely identify a query sent to a particular target (even if the same query is sent to multiple targets). Otherwise the source is not able to

distinguish the search results retrieved from various targets and/or queries previously submitted to a target.

Please note that the asynchronous query mode does not require an asynchronous handling on the messaging layer. It can also be implemented by two synchronous functions at the source and the target, respectively.

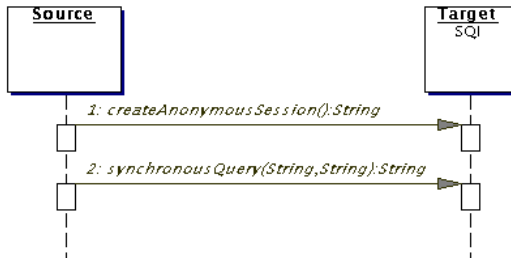


Figure 1: Synchronous Query Mode used for querying a single Repository

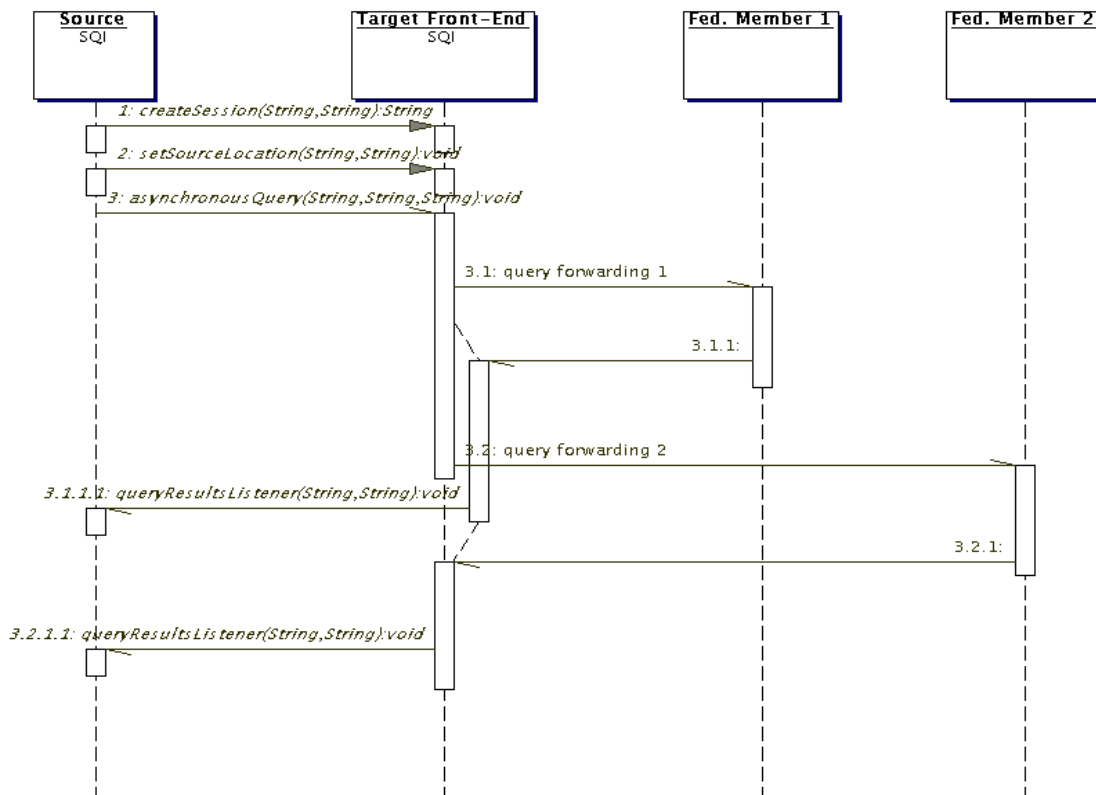


Figure 2: Asynchronous Query Mode used for performing a Federated Search

A query interface operated in synchronous mode can only have one active query per session. As a result, the session ID identifies the query uniquely. In case of an asynchronously operated query interface, the source issues a query ID that allows it to link incoming results to a submitted query (the source might query many targets). Multiple queries can be active within a session in asynchronous query mode.

1.3 Command-Query Separation Principle

SQI design is based on the "Command-Query Separation Principle". This principle states that every method should either be a command that performs an action, or a query that returns data to the caller, but not both. More formally, methods should return a value only if they are referentially transparent and hence cause no side-effects. This leads to a style of design that produces clearer and more understandable interfaces.

The Command-Query Separation (CQS) is a principle of object-oriented computer programming. It was devised by Bertrand Meyer a part of his pioneering work on the Eiffel programming language (Source: http://www.wordiq.com/definition/Command-Query_Separation).

1.4 Simple Command Set and Extensibility

In order to make the interface easily extensible we opted for an approach, which minimizes the number of parameters of the various methods rather than minimizing the number of methods. Variations of the interface (e.g. a separation between common query schema and common results format), can easily be introduced by adding a new function (e.g. `setSupportedQuerySchema`) while no change in the already implemented methods is need. Hereby, backwards compatibility can be more easily remained.

As a result, additional methods for setting query parameters like maximum duration and maximum number of returned search results were introduced. This design choice leads to simpler methods, but the number of interdependent methods is higher. However, for many of these query configuration and management methods, defaults can be used.

2 API Specification

2.1 Overview

First, the source needs to create a connection with the target. Once a session has been established (see Section 2), the query interface at the target awaits the submission of a search request.

A number of methods allow for the configuration of the interface at the target. Query parameters such as

- the query language (`setQueryLanguage`),
- the number of results returned within one results-set (`setResultsSetSize`),
- the maximum number of query results¹ (`setMaxQueryResults`),
- the maximum duration of query execution (`setMaxDuration`),
- and the results format (`setResultsFormat`)

can be set with the respective methods. The parameters set via these methods remain valid throughout the whole session or until they are set otherwise. If none of the methods is used before the first query is submitted, defaults are assumed.

	Implemented at the target and called by the source	Implemented at the source and called by the target
<i>Query Configuration</i>		
<code>setQueryLanguage</code>	X	
<code>setResultsFormat</code>	X	
<code>setMaxQueryResults</code>	X	
<code>setMaxDuration</code>	X	
<i>Synchronous Query Interface</i>		
<code>setResultsSetSize</code>	X	
<code>synchronousQuery</code>	X	
<code>getTotalResultsCount</code>	X	
<code>getAdditionalQueryResults</code>	X	
<i>Asynchronous Query Interface</i>		
<code>asynchronousQuery</code>	X	
<code>setSourceLocation</code>	X	
<code>queryResultsListener</code>		X
<i>Results Management</i>		
<code>getResourceDescription</code>	X	

Table 1: Overview of Simple Query Interface Methods

¹ While the size of results set determines the maximum number of results return by calling either `synchronousQuery` or `getAdditionalQueryResults`, the maximum number of results defines the total maximum number of results a query will return. Hence, it does not make sense to set the result set size bigger than the maximum number of results.

Then, the source submits a query, using either the `asynchronousQuery` or the `synchronousQuery` method. The query is then processed by the target and produces a set of records, also referred to as results set. The query is expressed in a query language identified through a query parameter. In the query, reference to a common schema is made.

To collect the query results from the source, two methods are offered, but their usage depends on the query mode the interface operates in.

- The `getAdditionalQueryResults` method can be used by the source to collect additional query results in case of a synchronously operated query interface. The `getTotalResultsCount` method returns the total number for matching metadata records found by the target.
- In case of an asynchronously operated query interface the `queryResultsListener` method is called by the target to forward the query results to the source. The `getResourceDescription` method allow for retrieving a learning object's full metadata record.

Table 1 provides an overview of the various methods and indicates whether they are implemented at the source or at the target.

2.2 Query Configuration

2.2.1 Set Query Language

This method allows the source to control the syntax used in the query statement by identifying the query language. Values for the parameter `queryLanguageID` can be provided in upper case, lower case, or capitalized.

```
Void setQueryLanguage (  
    String targetSessionID,  
    String queryLanguageID)  
throws  
    NoSuchSessionException,  
    QueryLanguageNotSupportedException;
```

The `NoSuchSessionException` is thrown in case the given `TargetSessionID` is invalid. `QueryLanguageNotSupportedException` is thrown if the query language used in the request is not supported by the target.

2.2.2 Set Maximum Number of Query Results

This method defines the maximum number of results, which a query will produce. The maximum number of query results is set to 100 by default, but can be controlled via this method. `maxQueryResults` must be 0 (zero) or greater. If the maximum number of query results is set to 0 (zero), the source does not want to limit the number of maximum query results produced.

```
Void setMaxQueryResults (  
    String targetSessionID,  
    Integer maxQueryResults)  
throws  
    NoSuchSessionException,  
    InvalidMaxQueryResultsException;
```

The *NoSuchSessionException* is thrown in case the given *TargetSessionID* is invalid. *InvalidMaxQueryResultsException* is thrown if an invalid number is provided.

2.2.3 Set Maximum Duration

This method enables the source to set a time-out for the query in case of an asynchronously operated query interface. The values of *maxDuration* must be 0 (zero) or greater. A source delegates the time out management of the query to the target by setting *maxDuration* to 0 (zero). The parameter *maxDuration* is interpreted in milliseconds. The default value is zero (i.e., time out management is delegated to the target).

```
Void setMaxDuration (  
    String targetSessionID,  
    Integer maxDuration)  
throws  
    NoSuchSessionException,  
    InvalidMaxDurationException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. *InvalidMaxDurationException* is thrown if an invalid number is provided.

2.2.4 Set Results Format

This method allows the source to control the format of the results returned by the target. The format according to which the results shall be formatted is specified in the *resultsFormat* parameter. The parameter is provided via a URI (e.g. “<http://ltsc.ieee.org/wg12/par1484-12-3/lom.xsd>” identifies the XML schema of the IEEE LOM standards) or via pre-defined values that can be provided in upper case, lower case, or capitalized.

```
Void setResultsFormat (  
    String targetSessionID,  
    String resultsFormat)  
throws  
    NoSuchSessionException,  
    ResultsFormatNotSupportedException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *ResultsFormatNotSupportedException* is thrown when the format provided via the *resultsFormat* parameter is not supported by the target.

2.3 Synchronous Query Methods

2.3.1 Set Results Set Size

This method defines the maximum number of results, which will be returned by single results-set. The size of the results-set is set to 25 records by default, but can be controlled via this method. *resultsSetSize* must be 0 (zero) or greater. A source asks for all results when the maximum number of results is set to 0 (zero).

```
Void setResultsSetSize (  
    String targetSessionID,  
    Integer resultsSetSize)  
throws  
    NoSuchSessionException,  
    InvalidResultsSetSizeException,  
    QueryModeNotSupportedException;
```

The *NoSuchSessionException* is thrown in case the given *TargetSessionID* is invalid. *InvalidResultsSetSizeException* is thrown if an invalid number is provided. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries.

2.3.2 Synchronous Query

This method places a query at the target. Since there can only be one active query per session, the *targetSessionID* also uniquely identifies the query. The query statement is provided via the *queryStatement* parameter. The resource attributes that have to be included in the results-set are defined in the query statement.

This method returns a set of metadata records matching the query. The number of results returned is controlled by `setResultsSetSize` and its default value. The total number of results produced is limited by `setMaxQueryResults` and its default value.

```
String synchronousQuery (  
    String targetSessionID,  
    String queryStatement)  
throws  
    NoSuchSessionException,  
    QueryModeNotSupportedException,  
    InvalidQueryStatementException;
```

The *NoSuchSessionException* is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries. The *InvalidQueryStatementException* is thrown if the query statement does not comply with the syntax of the query language.

2.3.3 Get Additional Query Results

After a query has been submitted and returned the first set of results, this method can be called in case the query has produced more results than delivered by the first results-set. The method returns a results-set consisting of a list of additional metadata records. The number of results included in the results-set is controlled by the `setResultsSetSize` method. The target returning the results ensures that the results comply with the results format specified.

The *targetSessionID* parameter identifies the previously defined query. The *startResult* parameter identifies the first record of the total results-set that will be returned.

The index of the result set size starts with 1. In case *startResult* is set to 0 (zero), the method delivers the “next” result set. For example, in case of the first call after the `synchronousQuery` call, `getAdditionalQueryResults` delivers a results-set starting with the record number ‘results-set size + 1’).

```
String getAdditionalQueryResults (  
    String targetSessionID,  
    Integer startResult)  
throws  
    NoSuchSessionException,  
    NoQuerySubmittedException,  
    QueryModeNotSupportedException,  
    InvalidStartResultException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *NoQuerySubmittedException* is thrown in case this method has been called without submitting a query beforehand. The *QueryModeNotSupportedException* is thrown in case the target does

not support synchronous queries. *InvalidStartResultException* is thrown if an invalid number is provided for *startResult*.

2.3.4 Get Total Results Count

This method returns the total number of available results of the previously submitted query. The *targetSessionID* identifies the previously submitted query (including the associated session).

```
Integer getTotalResultsCount (  
    String targetSessionID)  
throws  
    NoSuchSessionException,  
    QueryModeNotSupportedException,  
    NoQuerySubmittedException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support synchronous queries. *NoQuerySubmittedException* is thrown if no query has been submitted during the session.

2.4 Asynchronous Query Methods

2.4.1 Set Source Location

This method is required to be called before a query is submitted in asynchronous mode. The parameter *sourceLocation* specifies the location of the source's results listener in order for the target to be able to send the results.

```
Void setSourceLocation (  
    String targetSessionID,  
    String sourceLocation)  
throws  
    NoSuchSessionException,  
    QueryModeNotSupportedException;
```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support asynchronous queries.

2.4.2 Asynchronous Query

This method allows the source to submit a query to the target, while the results are returned in an asynchronous method. The query statement is provided via the *queryStatement* parameter. A query ID issued by the source is required in order to link the query results with the query, when they are later returned using the results listener. By using unique query IDs it is possible to submit an arbitrary number of queries per active session. The location of the source's results listener is needed and must be provided using *setSourceLocation* method.

Due to the asynchronous nature of this method, query results could still arrive from previous queries. The query is processed and results are forwarded within the timeframe specified in the *setMaxDuration* method.

```
Void asynchronousQuery (  
    String targetSessionID,  
    String queryStatement,
```

```

    String queryID)
throws
    NoSuchSessionException,
    NoSourceLocationException,
    QueryModeNotSupportedException,
    InvalidQueryStatementException;

```

NoSuchSessionException is thrown in case the given *targetSessionID* is invalid. The *QueryModeNotSupportedException* is thrown in case the target does not support asynchronous queries. The *NoSourceLocationException* is thrown in case no source location has been specified before submitting the query. The *InvalidSourceLocationException* is thrown if the location of the source has not been set (via the method `setSourceLocation`).

2.4.3 Query Results Listener

This target-initiated method forwards the results-sets to the source. The *queryID* parameter is used for linking the query results to previously submitted query, when they are later return using the results listener.

The *queryResults* holds a results-set consisting of a list of metadata records, which is formatted according to the schema specified in the query.

```

Void queryResultsListener (
    String queryID,
    String queryResults)
throws
    InvalidQueryResultsException,
    NoSuchQueryException;

```

InvalidQueryResultsException is thrown in case the results-set cannot be interpreted by the source. The *NoSuchQueryException* is thrown in case the given *queryID* is invalid.

2.5 Results Retrieval

2.5.1 Get Resource Description

This method returns the description of a learning resource. The parameter *resourceID* identifies the learning resource at target. The resource ID is provided in the results-set. The format according to which the results can be formatted is specified using the `setResultsFormat` method.

```

String getResourceDescription (
    String targetSessionID,
    String resourceID)
throws
    NoSuchSessionException,
    NoSuchResourceException;

```

The *NoSuchSessionException* is thrown in case the given *targetSessionID* is invalid. *NoSuchResourceException* is thrown if the learning resource with the provided *resourceID* does not exist at the source.

3 Open Work Items

- *API extension – Status Management:* Add methods supporting search status management (e.g. for cancellation of search, query status reporting).
- *API extension - Explain Method:* No method for retrieving the capabilities of an SQI node is provided. Here either one “explain” method (see for example SRW) could be provided, which delivers an SQI Profile Record. Alternatively many methods like `getSupportedQueryMode`, `getSupportedQueryLanguage`, etc. could be provided.
- *SQI Profiles:* The specification does not provide hints for defining SQI Profiles (Which query languages are supported? Which schemas are supported? Which query modes supported? In which format are results delivered?), nor is a profile format specified.
- *Enhance document with tutorial:* The document should provide a hands-on guide on how to connect a repository using SQI.
- *Enhance document with a detailed use case:* At least one typical use case with some code fragments should be added.
- *Enhance document by explaining `getResourceDescription` better:* It is not really clear what a resource description is and what a resource id is. An exemplifying use case would be beneficial.
- *Provide binding:* API bindings shall be provided where for example data types are specified at a higher level of detail.
- *Add a chapter on “Other uses of SQI”:* SQI can be used for exchanging other things than resource metadata. Currently the document is written entirely in the latter perspective. A chapter on "Other uses of SQI" should be added where we explain how SQI can be used to retrieve other things such as (language versions of) vocabularies up to evaluation data about training service providers. Even why not using SQI for performing a federated search into national school repositories?
- *Discuss:* In case of asynchronous querying, do we need to identify the target in the return Query ID?
- *Discuss:* In asynchronous query mode the target is calling a listener. However, there might be circumstances where it might be useful to do everything at the source side e.g. with polling.
- *Discuss:* In both documents, LOR Interoperability Framework and the SQI Specification, it is assumed that a clear separation between common query language and common schema does exist. Hereby, a scenario where a network of educational nodes is based on a common schema (e.g. an application profile of LOM) that can be query in different query languages (e.g. XQuery, QEL, etc.) can be realized. However, a close connection between query language and pre-defined attributes could exist, so that a distinction between query language and common schema can hardly be made. At the same time an additional distinction between common return format and common query schema might make sense, where the common query schema defines all the attributes that can be used in the query and the return schema the exact return format (including the attributes). If we separate query schema and results format, we will need an additional method for communicating the query schema.

- *Discuss*: We should still evaluate the pros and cons of the three options of the API formats more systematically before everything is carved in stone. The three options are roughly:
 - (1) many methods with a minimum set of parameters (Status Quo),
 - (2) A minimum set of methods with many parameters, and
 - (3) a minimum set of methods with a minimum set of parameters.The latter requires a more complex parameter such as an XML document as for example proposed by EduSource.