CEN

WORKSHOP

AGREEMENT

CWA 15454

November 2005

English version

A Simple Query Interface Specification for Learning Repositories

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties, the constitution of which is indicated in the foreword of this Workshop Agreement.

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN Management Centre can be held accountable for the technical content of this CEN Workshop Agreement or possible conflicts with standards or legislation.

This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its Members.

This CEN Workshop Agreement is publicly available as a reference document from the CEN Members National Standard Bodies.

CEN members are the national standards bodies of Austria, Belgium, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: rue de Stassart, 36   B-1050 Brussels

Contents

Foreword

This CWA presents an Application Programmer Interface (API) for querying learning object repositories. The core is an abstract API; bindings to web services, Java and PHP have also been developed. Since one major design objective is to keep the specification simple and easy to implement, the interface is labelled Simple Query Interface (SQI).

In the context of this CWA, learning object repositories are defined as collections of educational material, courses, and learning objects with associated descriptions (referred to as "metadata"). Examples of repositories for learning are educational brokers, knowledge pools, streaming video servers, etc.

The decision for this work item was taken by the Learning Technologies Workshop at the 14th meeting on March 24/25, 2003. Work on the CWA actually started at the 17th meeting in April 2003.

The document has been developed through the collaboration of a number of contributing partners, representing a wide mix of interests, from universities to commercial companies. The names of the individuals and their affiliations that have expressed support for this CWA is available at the CEN/ISSS Secretariat. A list of contributors is at the end of this Foreword.

The work relates closely to the Memorandum of Understanding on "Multimedia Access to Education and Training in Europe".

The final review/endorsement round for this CWA was started on 2005-06-16/17 and closed on 2005-07-15.

The final text of this CWA was submitted to CEN for publication on 2005-09-06.

The CWA is technically the same as the Version 1.0 Beta (2005-04-13) that is on http://www.prolearn-project.org/lori

The CWA is part of three SQI related documents (see http://www.prolearn-project.org/lori):

i. "Learning Object Repository Interoperability Framework": this document provides the "big picture" and should probably be read first if you want to understand the context of this work;
ii. [this CWA] "Simple Query Interface Specification": this is the core of the specification for querying learning object repositories in an interoperable way;
iii. "Authentication and Session Management": this document focuses on specific issues related to authentication and session management.

Contributions from the following initiatives are acknowledged: Ariadne, Educanext, Celebrate, Edutella, Elena, EduSource, ProLearn, Universal, Zing.

This CEN Workshop Agreement is publicly available as a reference document from the National Members of CEN: AENOR, AFNOR, BSI, CSNI, CYS, DIN, DS, ELOT, EVS, IBN, IPQ, IST, LVS, LST, MSA, MSZT, NEN, NSAI, ON, PKN, SEE, SIS, SIST, SFS, SN, SNV, SUTN and UNI.

Comments or suggestions from the users of the CEN Workshop Agreement are welcome and should be addressed to the CEN Management Centre.

<u>List of Contributors and Editors</u>

Bernd Simon, Vienna University of Economics and Business Administration
 & EducaNext (Editor)
Christian Werner, Learning Lab Lower Saxony
Erik Duval, Katholieke Universiteit Leuven & Ariadne (Editor)
Daniel Olmedilla, Learning Lab Lower Saxony
Dan Rehak, Carnegie Mellon University
David Massart, European Schoolnet (Editor)
Frans Van Assche, European Schoolnet (Editor)
Griff Richards, Simon Fraser University
Gerhard Müller, IMC
Julien Tane, Universität Karlsruhe
Marek Hatala, Simon Fraser University
Matthew J. Dovey, Oxford University
Michel Arnaud, Université de Paris X Nanterre
Nikos Papazis, NCSR
Peter Dolog, Learning Lab Lower Saxony
Sascha Markus, IMC
Stefaan Ternier, Katholieke Universiteit Leuven (Editor)
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

Version	Changes
1.0 [this CWA]	This version introduces the following changes: • In order to allow, both, a stateful and a stateless implementation of SQI in synchronous mode, the methods synchronousQuery and getadditionalResults have been merged into one method. The implementation section has been extended by a subsection on that issue. • The method 'get Resource Description' and the associated fault 'NO_SUCH_RESOURCE' have been removed. • The java-like notation of the methods was replaced by a language-neutral description. • The term 'exception' was replaced by the more general concept of 'fault'. • A new fault named SQIFault was introduced. It replaces all the exceptions previously used in order to improve the simplicity and the extendibility of the API. A new sub-section was added to describe the fault mechanism. • A section on SQI Application Profiles has been introduced. • The triggers for the INVALID_START_RESULT fault have been specified. • Two new faults called METHOD_NOT_SUPPORTED and NO_MORE_RESULTS were introduced. • Section 1.1: Query Language and Results Format was rewritten. • Figures 1 and 2 have been updated. • Addition of a class diagram of the specification. • Minor textual improvements. • Changed parameter specifications from 'can be upper case, lower case, or capitalized' to parameter is case-insensitive. • The specification mentions now explicitly that the source location parameter of the Asynchronous Query method must be a URL. • Updated 'Open Work Items List' (partly based on ELENA Interoperability Case Study) • Added a section on SQI bindings.

0.8	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.

Additionally the following changes were made:

- 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 InvalidLocationException was removed since it is unlikely to be called in case of an invalid location error.

- All "NoValid...Exceptions" were changed to "Invalid… Exceptions".

- For setSourceLocation and asynchronousQuery the name of the exception that is thrown in case the asynchronous query is not supported has been changed from WrongQueryModeException to QueryModeNotSupportedException.

- Textual improvements of API description and introductory section.

| 0.7 | *At the European Schoolnet side David Massart replaces Frans Van Assche as editor of this document.* |

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.

- The methods setMaxResults and setMaxDuration are supported by both query methods (synchronousQuery and asynchronousQuery). setMaxResults has been split into two methods: setMaxQueryResults and setResultsSetSize. 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 setQueryMode has been removed. Instead the query mode specific methods return an exception in case the query mode is not supported.

- The requirement for a *sourceSessionID* in the asynchronous query mode has been removed. A *queryID*, issued by the source, has been introduced instead.

- *WrongQueryModeException* 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: setQueryLanguage, setQuerySchema, setSourceLocation and setResultsFormat. 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.

- getSupportedQueryModes 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 "getQueryResults" is now called "getAdditionalQueryResults".

- Default values for the methods setMaxQueryResults (100), setResultsSetSize (25), and setMaxDuration (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 corrected.

- 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	*From this version on, the specification is jointly edited by Bernd Simon, Erik Duval, and Frans Van Assche.* The following changes were introduced: • The SQI distinguishes now between an asynchronous query method (asynchronousQuery) and a synchronous query method (synchronousQuery), 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 query method and getQueryResults 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 *targetSessionID* and a *sourceSessionID* 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 getSupportedQueryLanguages and getSupportedSchemas 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	An application profile for a very simple query interface has been designed in Appendix B. → Thanks to the initiative of Erik Duval.
0.51	The presentation of the specification has been updated. Related work on EduSource added; XPATH included as a permissible value of the getSupportedQueryLanguages (→ based on input from Marek Hatala). Thanks to Peter Dolog the functionality overview also includes a UML activity diagram.

0.5	The following changes were introduced:
	• Besides the source-initiated (synchronous) query (mode) also a target-initiated (asynchronous) query (mode) was introduced (methods: queryResultsListener, setMaxDuration). → Thanks to a discussion with David Massart and Frans Van Assche.
	• Parameter SchemaReferences was introduced in the Query method providing a hook for data model mappings. Additionally, Method getSupportedSchemas was added. → Input from Zoltán Miklos.
	• 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	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.

1 Requirements and Design Principles

This paper presents an Application Program Interface (API) for querying learning objects repositories. Since one major design objective is to keep the specification simple and easy to implement, the interface is labelled <u>S</u>imple <u>Q</u>uery <u>I</u>nterface (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 application of the SQI specification in heterogeneous use cases.

> ➢ SQI supports, both, a stateful and a stateless implementation.

> ➢ SQI is based on a session management concept in order to separate authentication issues from query management.

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

In order to make use of SQI to implement full query functionality, SQI needs to be complemented with agreements about:

- the set of attributes and vocabularies that can be used in the query,

- the query language and its representation,

- the representation of list of learning objects that satisfy the query, and

- the representation of individual metadata instances on learning objects.

SQI is agnostic on these issues: Any agreement between two or more repositories is valid for SQI. Such agreements can, for example, be expressed by XML schemas or RDF schemas.

Although SQI does not directly 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 (Figure 1), 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 (Figure 2), 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 implement 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.
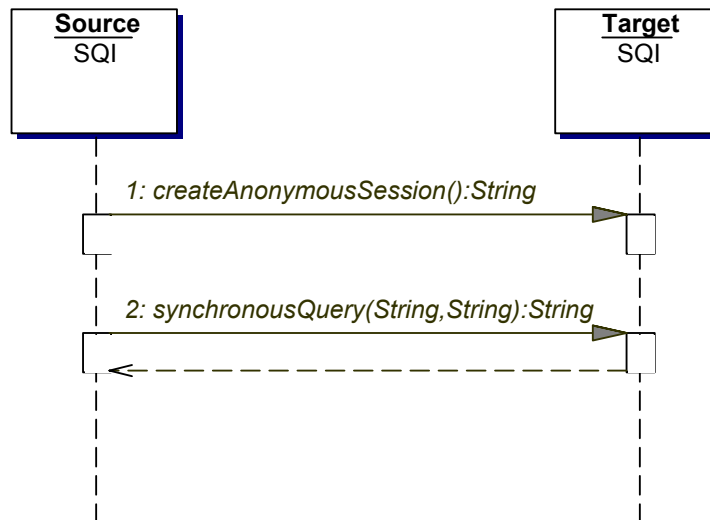

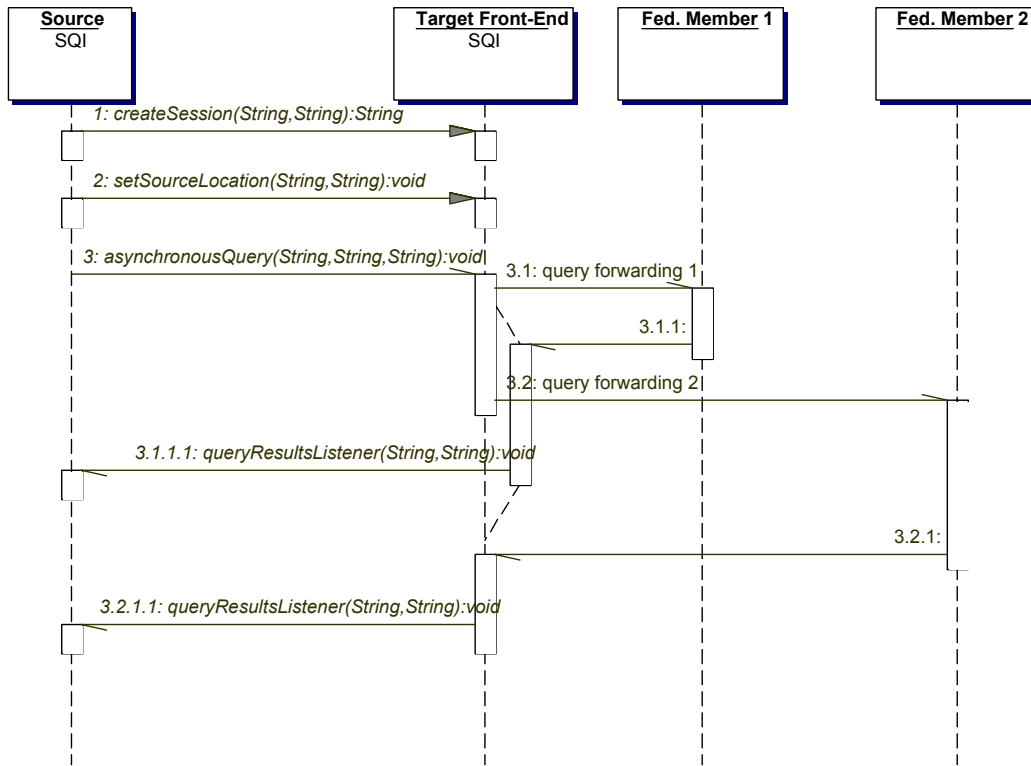**Figure 1: Synchronous Query Mode used for querying a single Repository**

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

A query interface operated in synchronous mode can perform multiple queries per session (even simultaneously). In case of an asynchronously operated query interface, the source provides a query ID that allows it to link incoming results to a submitted query (the source might query many targets and each target might answer to a query by returning more than one result to the source). Multiple queries can also be active within a session in asynchronous query mode.

1.3   Session Management

The interfaces introduced herein are based on a simple session management concept. It is assumed that a session has to be established before any further communication can take place. This specification separates query management and processing from authentication (and query policy management).

In case of a synchronously operated query interface, the source establishes a session at the target and uses the Session ID, which it obtained from the target, to identify itself during communication. Authentication does not need to be based on passwords, since also anonymous sessions can be created. Further details can be obtained from a separate document on the Session Management. The Session ID is also referred to as *targetSessionID*, since it is issued by the target to identify the source.

See the "Authentication and Session Management" document for details about the authentication and session management mechanism associated with SQI.

1.4   Stateful and Stateless Communication

Stateful and stateless are attributes that describe whether repositories are designed to keep track of one or more preceding events in a given sequence of interactions. Stateful means that the target repository keeps track of the state of interaction, for example, by storing the results of a previously submitted query in a cache. Stateless means that there is no record of previous interactions and that each interaction request can only be handled on the basis of the information that comes with it. The SQI specification allows implementers to opt for a stateful or a stateless approach.

1.5   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).

1.6   Simple Command Set and Extensibility

In order to make the interface easily extensible an approach, minimizing the number of parameters of the various methods rather than the number of methods is adopted. 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 needed. Hereby, backwards compatibility can be more easily maintained.

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, default values can be used for many of these query parameter configuration methods.

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 results[1] (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 are used before the first query is submitted, defaults are assumed.

Table 1: Overview of Simple Query Interface Methods

	Implemented at the target and called by the source	Implemented at the source and called by the target
Query Parameter Configuration		
setQueryLanguage	X	
setResultsFormat	X	
setMaxQueryResults	X	
setMaxDuration	X	
Synchronous Query Interface		
setResultsSetSize	X	
synchronousQuery	X	
getTotalResultsCount	X	
Asynchronous Query Interface		
asynchronousQuery	X	
setSourceLocation	X	
queryResultsListener		X

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, referred to as results set. The query is expressed in a query language identified

---

[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.

through a query parameter. In the query, reference to a common schema might be made. In synchronous mode the query results are directly returned by the synchronousQuery method. The getTotalResultsCount method returns the total number for matching metadata records found by the target operating. In case of an asynchronously operated query interface the queryResultsListener method is called by the target to forward the query results to the source.

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

In order to report abnormal situations (e.g., erroneous parameters or inability to carry out an operation), an *SQIFault* is provided, which can be thrown by all the SQI methods. A system of fault codes permits to document those abnormal situations.

2.2 Query Parameter 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* are case-insensitive.

Method name	**setQueryLanguage**
Return type	Void
Parameters	*Name* *Type* targetSessionID String queryLanguageID String
Fault	NO_SUCH_SESSION QUERY_LANGUAGE_NOT_SUPPORTED METHOD_FAILURE

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *TargetSessionID* is invalid;

15

- "QUERY_LANGUAGE_NOT_SUPPORTED" if the query language used in the request is not supported by the target; and

- "METHOD_FAILURE" if the operation fails for another reason.

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.

Method name	**setMaxQueryResults**
Return type	Void
Parameters	*Name* *Type* targetSessionID String maxQueryResults Integer
Fault	NO_SUCH_SESSION INVALID_MAX_QUERY_RESULTS METHOD_FAILURE

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *TargetSessionID* is invalid;

- "INVALID_MAX_QUERY_RESULTS" if an invalid number is provided for *maxQueryResults*; and

- "METHOD_FAILURE" if the operation fails for another reason.

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).

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "INVALID_MAX_DURATION" if an invalid number is provided for *maxDuration*; and

- "METHOD_FAILURE" if the operation fails for another reason.

Method name	**setMaxDuration**
Return type	Void
Parameters	*Name* *Type* targetSessionID String maxDuration Integer
Fault	NO_SUCH_SESSION INVALID_MAX_DURATION METHOD_FAILURE

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., the LOM XML Schema definitions files are available at http://standards.ieee.org/reading/ieee/downloads/LOM/lomv1.0/) or via pre-defined values that are case-insensitive.

Method name	**setResultsFormat**
Return type	Void
Parameters	*Name* *Type* targetSessionID String resultsFormat String
Fault	NO_SUCH_SESSION RESULTS_FORMAT_NOT_SUPPORTED METHOD_FAILURE

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "RESULTS_FORMAT_NOT_SUPPORTED" when the format provided via the *resultsFormat* parameter is not supported by the target; and

- "METHOD_FAILURE" if the operation fails for another reason.

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 a 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).

Method name	**setResultsSetSize**	
Return type	Void	
Parameters	*Name*	
	Type	
	targetSessionID	
	String	
	resultsSetSize	
	Integer	
Fault	NO_SUCH_SESSION	
	INVALID_RESULTS_SET_SIZE	
	QUERY_MODE_NOT_SUPPORTED	
	METHOD_FAILURE	

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "INVALID_RESULTS_SET_SIZE" if an invalid number is provided for *resultsSetSize*;

- "QUERY_MODE_NOT_SUPPORTED" in case the target does not support synchronous queries; and

- "METHOD_FAILURE" if the operation fails for another reason.

2.3.2   Synchronous Query

This method places a query at the target. The query statement is provided via the *queryStatement* parameter. Within a session identified via *targetSessionID* multiple queries can be submitted simultaneously. The method returns a set of metadata records matching the query. The *startResult* parameter identifies the start record of the results set. The index of the result set size starts with 1. The number of results returned is controlled by setResultsSetSize and its default value. A valid number for *startResult* can range from 1 to the total number of results. The total number of results produced is limited by setMaxQueryResults and its default value.

Method name	**synchronousQuery**		
Return type	String		
Parameters	*Name*		
	Type		
	targetSessionID		
	String		
	queryStatement		
	String		
	startResult		
	Integer		
Fault	NO_SUCH_SESSION		
	INVALID_ QUERY_STATEMENT		
	QUERY_MODE_NOT_SUPPORTED		
	METHOD_FAILURE		
	INVALID_START_RESULT		
	NO_MORE_RESULTS		
	METHOD_FAILURE		

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "QUERY_MODE_NOT_SUPPORTED" in case the target does not support synchronous queries;

- "INVALID_QUERY_STATEMENT" if the query statement does not comply with the syntax of the query language; and

- "INVALID_START_RESULT" if an invalid number is provided for *startResult*;

- "NO_MORE_RESULTS" if *startResult* is set to zero and no more results are available and

- "METHOD_FAILURE" if the operation fails for another reason.

2.3.3   Get Total Results Count

This method returns the total number of available results of a query. The *targetSessionID* identifies the session. The query is provided via the *queryStatement* parameter (see Section 3.1 on Stateful versus Stateless Implementation).

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "QUERY_MODE_NOT_SUPPORTED" in case the target does not support synchronous queries;

- "INVALID_QUERY_STATEMENT" if the query statement does not comply with the syntax of the query language; and

- "METHOD_FAILURE" if the operation fails for another reason.

Method name	**getTotalResultsCount**
Return type	Integer
Parameters	*Name*
	Type
	targetSessionID
	String
	queryStatement
	String
Fault	NO_SUCH_SESSION
	QUERY_MODE_NOT_SUPPORTED
	INVALID_QUERY_STATEMENT
	METHOD_FAILURE

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. The *sourceLocation* must be an URL.

Method name	**setSourceLocation**
Return type	Void
Parameters	*Name* *Type* targetSessionID String sourceLocation String
Fault	NO_SUCH_SESSION QUERY_MODE_NOT_SUPPORTED METHOD_FAILURE

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "QUERY_MODE_NOT_SUPPORTED" if the target does not support asynchronous queries; and

- "METHOD_FAILURE" if the operation fails for another reason.

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.

Method name	**asynchronousQuery**
Return type	Void
Parameters	*Name*
	Type
	targetSessionID
	String
	queryStatement
	String
	queryID
	String
Fault	NO_SUCH_SESSION
	QUERY_MODE_NOT_SUPPORTED
	NO_SOURCE_LOCATION
	INVALID_QUERY_STATEMENT
	METHOD_FAILURE

The following faults can occur:

- "NO_SUCH_SESSION" in case the given *targetSessionID* is invalid;

- "QUERY_MODE_NOT_SUPPORTED" if the target does not support asynchronous queries;

- "NO_SOURCE_LOCATION" in case no source location has been specified before submitting the query (via the method setSourceLocation*)*;

- "INVALID_QUERY_STATEMENT" if the query statement does not comply with the syntax of the query language; and

- "METHOD_FAILURE" if the operation fails for another reason.

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.

Method name	queryResultsListener
Return type	Void
Parameters	*Name* *Type* queryID String queryResults String
Fault	INVALID_QUERY_RESULTS NO_SUCH_QUERY METHOD_FAILURE

The following faults can occur:

- "INVALID_QUERY_RESULTS" in case the results set cannot be interpreted by the source;

- "NO_SUCH_QUERY" in case the given *queryID* is invalid; and

- "METHOD_FAILURE" if the operation fails for another reason.
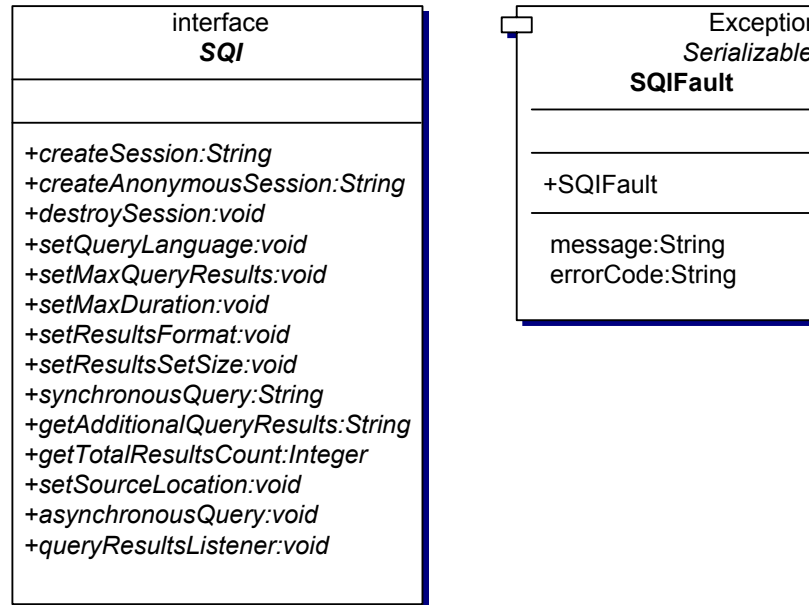
Fault Name	Fault Code	Description
UNDEFINED	SQI-00000	An undefined fault occurred. All the methods can throw an SQIFault with this code.
METHOD_FAILURE	SQI-00001	The method failed although it was correctly called, i.e., it is a problem at the callee side. The latter was not able to carry out the requested action. All the methods can throw a fault with this code.
NO_SOURCE_LOCATION	SQI-00002	No source location has been specified before submitting an asynchronous query.
INVALID_START_RESULT	SQI-00003	An invalid number was provided for the start Result.
INVALID_QUERY_STATEMENT	SQI-00004	The query statement does not comply with the syntax of the query language.
INVALID_RESULTS_SET_SIZE	SQI-00005	An invalid results set size was provided.
INVALID_MAX_DURATION	SQI-00006	An invalid duration was provided.
INVALID_MAX_QUERY_RESULTS	SQI-00007	An invalid maximum for the number of query results was provided.
INVALID_QUERY_RESULTS	SQI-00008	The results set cannot be interpreted by the source.
QUERY_MODE_NOT_SUPPORTED	SQI-00009	The target does not support the query mode required by the method.
RESULTS_FORMAT_NOT_SUPPORTED	SQI-00010	The results format provided is not supported by the target.
QUERY_LANGUAGE_NOT_SUPPORTED	SQI-00011	The query language used in a request is not supported by the target.
METHOD_NOT_SUPPORTED	SQI-00012	The method is not supported by the callee side.
NO_SUCH_SESSION	SQI-00013	The given session id is invalid.
NO_SUCH_QUERY	SQI-00014	The given query id is invalid.
WRONG_CREDENTIALS	SQI-00015	An invalid user id and/or password was provided.
NO_MORE_RESULTS	SQI-00016	Additional results are requested, but all results have already been provided.

Table 2: Overview of Simple Query Interface Faults

SQI provides only one fault named *SQIFault* (Figure 3), which is thrown by all methods. The *SQIFault* includes two properties:

- *message*: a free text describing the reason why the fault occurred, and
- *faultCode*: a fault code identifying the problem. The possible fault codes are part of the SQI specification. Faults SQI-00002 to SQI-0016 correspond to possible violations of preconditions of SQI methods whereas fault SQI-00001 corresponds to a failure of a method correctly called and fault SQI-00000 permits to report possible faults that are not yet supported by the specification. The complete list of SQI fault codes is presented in Table 2.

3 SQI Implementation Issues

3.1 Stateful versus Stateless Implementations

SQI in synchronous mode can be implemented, both, as a stateful as well as a stateless service. A stateless SQI service discards the query results as soon as they are transferred to the source. Whenever additional results are requested the query is resubmitted and reprocessed again by the target and the additional results are delivered.

A stateful SQI service keeps the state information of previous interactions. For example by caching the results, a reprocessing of a query is not needed in case additional results are requested. A stateful SQI service can use the query (or the hash of the query) to identify previously submitted queries.

For example, the method getAdditionalResultsCount would resubmit the query in a stateless implementation while only counting of the cached results could be done in a stateful implementation.

3.2 Bindings

An obvious way to implement SQI consists in using web-services. In order to ensure the interoperability between the different SQI implementations, a common WSDL binding for SQI is currently developed as a Sourceforge project (http://sqi-wsdl.sourceforge.net/).

A network consisting of object-oriented systems can implement the specific faults listed in Table 2 as sub-exceptions of the above described general SQIFault.

3.3 SQI Application Profiles

Application scenarios of this specification can be assumed, where only a sub-set of the methods proposed are used. In this case the methods that are not supported will throw an SQI fault (fault code: METHOD_NOT_SUPPORTED). For example a two-node SQI 'network' agrees on using only synchronous query methods and refrains from taking advantage of the query parameter configuration methods, since query parameters will be hard-coded.

An SQI Application Profile may also reference the query languages and results formats used by the systems.