INTERNATIONAL STANDARD

ISO/IEC 24730-1

First edition
2006-02-15

Information technology — Real-time locating systems (RTLS) —

Part 1: Application program interface (API)

Technologies de l'information — Systèmes de localisation en temps réel (RTLS) —

Partie 1: Interface de programmation d'application (API)

Reference number
ISO/IEC 24730-1:2006(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 24730-1 was prepared by Technical Committee ISO/TC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

ISO/IEC 24730 consists of the following parts, under the general title *Information technology — Real-time locating systems (RTLS)*:

— *Part 1: Application program interface (API)*

— *Part 2: 2,4 GHz air interface protocol*

The following part is under preparation:

— *Part 3: 433 MHz air interface protocol*

Introduction

ISO/IEC 24730 defines two air interface protocols and a single application program interface (API) for real-time locating systems (RTLS) for use in asset management and is intended to allow for compatibility and to encourage interoperability of products for the growing RTLS market.

This part of ISO/IEC 24730, the RTLS application program interface, establishes a technical standard for RTLS. To be fully compliant with this standard, RTLS must comply with this part of ISO/IEC 24730 and at least one air interface protocol defined in ISO/IEC 24730.

Real-time locating systems are wireless systems with the ability to locate the position of an item anywhere in a defined space (local/campus, wide area/regional, global) at a point in time that is, or is close to, real time. Position is derived by measurements of the physical properties of the radio link.

Conceptually there are four classifications of RTLS:

— Locating an asset via satellite (requires line-of-sight) - accuracy to 10 m.

— Locating an asset in a controlled area, e.g., warehouse, campus, airport (area of interest is instrumented) - accuracy to 3 m.

— Locating an asset in a more confined area (area of interest is instrumented) - accuracy to tens of centimetres.

— Locating an asset over a terrestrial area using a terrestrial mounted receiver over a wide area, cell phone towers for example – accuracy 200 m.

There are a further two methods of locating an object which are really RFID rather than RTLS:

— Locating an asset by virtue of the fact that the asset has passed point A at a certain time and has not passed point B.

— Locating an asset by virtue of providing a homing beacon whereby a person with a handheld can find an asset.

The method of location is through identification and location, generally through multi-lateration. The different types are

- Time of Flight Ranging Systems,
- Amplitude Triangulation,
- Time Difference of Arrival (TDOA),
- Cellular Triangulation,
- Satellite Multi-lateration,
- Angle of Arrival.

This part of ISO/IEC 24730 defines an API needed for utilizing an RTLS.

An API is a boundary across which application software uses facilities of programming languages to invoke services. These facilities may include procedures or operations, shared data objects and resolution of identifiers. A wide range of services may be required at an API to support applications. Different methods may be appropriate for documenting API specifications for different types of services.

The information flow across the API boundary is defined by the syntax and semantics of a particular programming language, such that the user of that language may access the services provided by the application platform on the other side of the boundary. This implies the specification of a mapping of the functions being made available by the application platform into the syntax and semantics of the programming language. An API specification documents a service and/or service access method that is available at an interface between the application and an application platform.

This API describes the RTLS service and its access methods, to enable client applications to interface with the RTLS. This RTLS service is the minimum service that must be provided by an RTLS to be API compatible with this standard.

Information technology — Real-time locating systems (RTLS) —

Part 1:
Application program interface (API)

1 Scope

This part of ISO/IEC 24730 enables software applications to utilize a real-time locating system (RTLS) infrastructure to locate assets with RTLS transmitters attached to them. It defines a boundary across which application software uses facilities of programming languages to collect information contained in RTLS tag blinks received by the RTLS infrastructure.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 19762-1, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 1: General terms relating to AIDC*

ISO/IEC 19762-3, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 3: Radio frequency identification (RFID)*

ISO/IEC 9075:2003 (all parts), *Information technology — Database languages — SQL*

IETF RFC 2616: June 1999, *Hypertext Transfer Protocol — HTTP/1.1* (http://www.ietf.org/rfc/rfc2616.txt)

Extensible Markup Language (XML) 1.0, (Third Edition) W3C Recommendation, World Wide Web Consortium (W3C), 4 February 2004. (http://www.w3.org/TR/REC-xml/)

XML Schema Part 1: Structures, W3C Recommendation, World Wide Web Consortium (W3C), Cambridge Massachusetts, 2 May 2001. (http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/)

XML Schema Part 2: Datatypes, W3C Recommendation, World Wide Web Consortium (W3C), Cambridge Massachusetts, 2 May 2001. (http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/)

SOAP Version 1.2 Part0: Primer, W3C Recommendation, World Wide Web Consortium (W3C), 24 June 2003, (http://www.w3.org/TR/2003/REC-soap12-part0-20030624/)

SOAP Version 1.2 Part1: Messaging Framework, W3C Recommendation, World Wide Web Consortium (W3C), 24 June 2003. (http://www.w3.org/TR/2003/REC-soap12-part1-20030624/)

SOAP Version 1.2 Part2: Adjuncts, W3C Recommendation, World Wide Web Consortium (W3C), 24 June 2003. (http://www.w3.org/TR/2003/REC-soap12-part2-20030624/)

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762-1, ISO/IEC 19762-3 and the following apply.

3.1
RTLS transmitters
active (battery powered) radio transmitters that are attached to assets (items, people, etc.)

NOTE RTLS transmitters send messages containing a unique ID for the asset, and may provide certain status information about the RTLS transmitter (e.g. battery level) and the asset (e.g. temperature).

3.2
tag blink
series of transmissions communicating a single asset movement or status change

NOTE Within the API, a tag blink consists of a "TagID" property that uniquely identifies the RTLS transmitter, as well as numerous other properties. Within the API, each property is also referred to as a field.

3.3
field
element of a data record in which information is stored, which may contain one or more properties of a tag blink, and which has a unique XML tag associated with it

NOTE Each tag blink property has a single, corresponding field in each TagBlink *XML* message. Conversely, within this API, each field may have one or more properties.

3.4
XML tag
marker that qualifies content in an XML document

3.5
SOAP
lightweight protocol intended for exchanging structured information in a decentralized, distributed environment

NOTE See references to SOAP Version 1.2 in the bibliography for additional information.

3.6
service
software program provides responses to requests from other software programs, which are frequently on other remotely connected computers

3.7
persistent connection
network connection between a webserver and a client that is kept open even after sending error responses

3.8
real-time
actively monitored with sub-minute updates for asset movement or status change

4 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in ISO/IEC 19762-1, ISO/IEC 19762-3 and the following apply.

HTTP	HyperText Transfer Protocol
RPC	Remote Procedure Call
RTLS	Real Time Locating System
XML	eXtensible Markup Language

5 The service

5.1 Purpose

The RTLS software service shall be a Web Service. It receives tag blinks from the RTLS infrastructure, and delivers those blinks as the response to client requests, over standard Internet protocols. In addition, it allows filtering and field selection on the contents of those blinks.

5.2 Specification summary

— An RTLS service shall support message exchange using the SOAP Version 1.2 encoding.

— An RTLS service should listen and respond to client requests using the HTTP 1.1 network protocol. If this network protocol is used, then the service shall conform to the IETF RFC 2616. It should listen on port 80, or port 443 (for secure communication).

— An RTLS service may take advantage of the persistent connections feature, which is available in HTTP 1.1. This would yield significant performance benefits for repeated calls to the RTLS Service, as described in Clause 7.4.

— An RTLS service shall maintain state of the last blink of all tags in the RTLS infrastructure, since the service was started. It may maintain state, from before its start time, by using a repository.

— An RTLS service shall support filtering and field selection, as described in Clause 7.2.5 and Clause 7.2.6. An RTLS service shall eliminate extraneous spaces in the content of query parameters, such as filters, as described in Clause 7.2.5.

— An RTLS service shall be able to create a session, on request from the client, as described in Clause 7.3.

— Each newly, created session shall keep state of the query parameters, such as the filter criteria, for utilization in subsequent requests to that session.

— Each request to that session shall return all tag blinks that match the filter criteria, and which occurred subsequent to the last request, as described in the Clause 7.4.

— If no tag blinks match the criteria, an RTLS service may delay response to session-based requests, until such tag blinks are received, or some pre-determined period has passed.

— An RTLS service shall remove the session state, when the CloseSession request is received, as described in Clause 7.5.

— An RTLS service may close the session, when a QuerySession request is not received after some predetermined period of time. This optional, predetermined period (ExpirationSeconds) may be defined by each vendor, in the SessionResponse structure, as described in Clause 8.3.

— An RTLS Service shall limit the length of all string values to a 1000 characters.

5.3 Security

The API uses SOAP Version 1.2, which typically assumes standard Internetworking protocols (such as HTTP 1.1 and TCP/IP). Security issues regarding RTLS message exchange can be addressed using existing, security standards and technologies at the communication layer. Therefore, security of the RTLS message exchange is beyond the scope of this standard.

6 The Application Program Interface (API)

6.1 Purpose

The Application Program Interface (API) provides a standard mechanism for accessing location-enriched, tag blinks from the RTLS Service from a client application.

6.2 Language Independence

This API specifies a language independent interface to the RTLS Service. It does so by using an industry standard protocol, SOAP Version 1.2, to communicate to the RTLS service.

6.3 Architecture

Figure 1 describes all the API message exchanges between a client application and the RTLS Service. The RTLS service shall not keep state regarding a stateless RPC request (Query), but it shall keep state of a session-based RPC request (OpenSession).

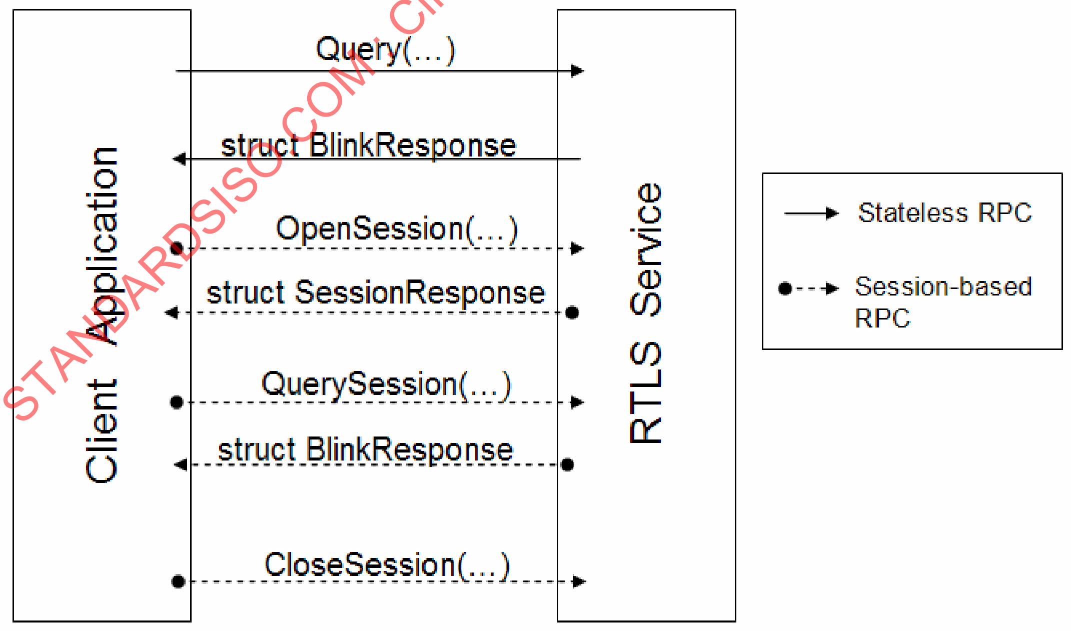

Figure 1 — Architecture of SOAP-RPC

6.4 Nomenclature and conventions

The API standard declares namespaces for RTLS in its XML Schema. All fields in the message exchange fall under one of the namespaces described in those schemas. See the Annex A for the schemas.

7 Subroutine calls

7.1 Overview of SOAP-RPC

a) struct QueryResponse Query(char* queryName, struct FilterType filters, struct FieldType fields, struct SortType SortBy)

b) struct SessionResponse OpenSession(char* queryName, struct FilterType filters, struct FieldType fields)

c) struct QueryResponse QuerySession (char* SessionID)

d) void CloseSession(char* SessionID)

"SOAP-RPC" refers to the remote methods provided to client applications by the RTLS Service.

The function signatures shown above are in C language syntax, merely to describe corresponding SOAP-RPC methods concisely. The SOAP-RPC may be bound to any programming language, with corresponding subroutines. The client implementation of each subroutine shall create a well-formatted *SOAP* message, and should make an *HTTP-POST* request to the RTLS Service.

Responses to a request may be a *SOAP* fault structure, a QueryResponse structure, or a SessionResponse structure. These structures may be deserialized, only if no error occurred. If an error occurs, the server shall return a *SOAP* fault structure, and the client may deserialize that fault structure.

7.2 Remote Procedure Call: Query

7.2.1 Purpose

This client request retrieves the last blink received by the RTLS Service for all tags that match the criteria of the query. At most 1 blink per tag is returned. This is a stateless query. No state about the query is kept in the RTLS service.

7.2.2 Synopsis

The RPC function signature in C language syntax would look like:

struct QueryResponse Query(char* queryName, struct FilterType filters, struct FieldType fields, struct SortType SortBy)

Input parameters:

- QueryName
- Filters
- Fields
- SortBy

Output parameters:

— QueryResponse structure

An underlying requirement is that parameters follow the Query method *SOAP* schema, as described in the Annex A, Clause A.2.

7.2.3 Description

The client application shall construct a *SOAP* request message, which will be deserialized at the service to invoke the function above. The *SOAP* request schema for Query RPC is described in the Annex A.

The client application is responsible for opening a connection (typically, using HTTP 1.1), and sending the *SOAP* request message - Query, with its associated parameters. The response will consist of either a QueryResponse structure or a *SOAP* fault structure. The client application, or a library that implements client side proxy functions, shall parse the response accordingly.

All the parameters discussed below refer to fields of a tag blink. These fields are described in the QueryResponse data structure, in Clause 8.2, and in the XML Schema defined in Annex A.

7.2.4 Parameter: queryName

This is the simplest parameter. This parameter opens up the opportunity for creating other queries to the RTLS system, in the future.

Requirements:

1) The RTLS Service shall accept “RTLS_Blinks” as the value for the query name.

EXAMPLE

<QueryName>RTLS_Blinks</QueryName>

7.2.5 Parameter: filters structure

This is the most complex parameter.

Requirements:

1) The service shall return only those tag blinks that match the filters criteria, in the QueryResponse structure.

2) If no filters are needed, the filter parameter shall be passed as follows: <FilterBy/>.

3) The filters shall be expressed as a sum-of-products Boolean expression, with Boolean operators acting on relational expressions, which in turn contain TagBlink fields and their values.

4) The filters parameter shall use a subset of Boolean and relational operators, as standardized in: ISO/IEC 9075:2003 (all parts), *Information technology — Database languages — SQL*. See Table 1 below.

5) Filters shall not include fields with no values, such as parent fields. Examples of parent fields are <VendorSection>, <States/> and <TagBlink/>.

6) The service shall ignore extraneous blank spaces in non-string field values.

7) The right-hand side of all relational expressions, except those containing ‘=’ only, shall be inside a CDATA section. E.g. <TagID><![CDATA[<> 600]]></TagID>.

Table 1 — Operators

Relational operators	Boolean operator
>, <, >=, <=, =, <>	OR

EXAMPLE 1

```
<FilterBy>
      <TagID><![CDATA[>50]]></TagID>
      <TagID><![CDATA[<= 1001]]></TagID>
      <OR/>
      <BatteryLow>=true</BatteryLow>
      <Blinking>=true</Blinking>
</FilterBy>
```

This filter is equivalent to “all blinks which have TagIDs greater than 50 **AND** less than or equal to 1001, **OR** any tag with a BatteryLow flag set to true **AND** has a Blinking flag set to true.” Note that this is a sum of products Boolean expression. A sequence of relational expressions which are not separated by an “</OR>” operator, are assumed to be ANDed together.

EXAMPLE 2

The following is an example of an **invalid** filter. It includes a parent field <VendorSection>, which has no values by itself.

```
<FilterBy>
      <VendorSection>
             <CustomField>=foobar</CustomField>
     </VendorSection>
</FilterBy>
```

The following is the **valid** version of the filter above.

```
<FilterBy>
        <CustomField>=foobar</CustomField>
</FilterBy>
```

7.2.6 Parameter: fields structure

A tag blink can potentially have a large number of fields, not all of which may be interesting to the client application. By specifying a list of fields, the number of fields per blink returned can be reduced, thereby reducing unnecessary network traffic.

Requirements:

1) The list of fields shall be expressed as a space delimited sequence of fields.

2) The list shall consist of either TagBlink properties or a parent *XML* tag containing TagBlink properties.

3) If a parent tag is specified, with no mention of any child tags, then all child elements for that parent tag shall be returned in the response. If child tags are mentioned, then only the fields corresponding to those tags shall be returned.

If all fields are desired, the fields parameter shall be either:

1) <Fields/>

or

2) <Fields>TagBlink</Fields>

Note that the second alternative works, because the <TagBlink/> tag is the parent field of all tag blink fields.

EXAMPLE

```
<Fields>
    TagID BatteryLow
</Fields>
```

The response shall contain tag blinks with only 2 fields: TagID and BatteryLow.

7.2.7 Parameter: SortBy structure

Specifying the sort order in the query can control the order in which tag blinks appear in the QueryResponse structure.

Requirements:

1) The SortBy parameter is optional. If omitted, the tag blinks shall be sorted by the RTLSBlinkTime field. If omitted, the parameter shall be passed as follows: <SortBy/>.

2) The SortBy parameter shall consist of a single TagBlink property, followed by a sort order. It shall not contain a parent field such as <Location>, since that field does not contain a value to sort by.

3) The sort order shall be either “asc” for ascending, or “desc” for descending.

EXAMPLE

```
<SortBy>
    <Field>TagID</Field>
    <Order>asc</Order>
</SortBy>
```

In this example, the response shall contain tag blinks in ascending order of the TagID field.

7.2.8 Output parameter: QueryResponse structure

QueryResponse structure is described in Clauses 8.1 and 8.2. Its *XML* schema is defined in the Annex A, in Clauses A.6 and A.7.

7.2.9 Faults

A general description and structure for faults is described in Clause 8.4. The schema is defined in the Annex A, in Clause A.9. Table 2 contains the error codes and related information that shall be included in a *SOAP* fault response to a Query RPC.

Table 2 — Error codes

Error code	Error message	SOAP fault code	SOAP fault subcode
1000	The request is not compliant with the RTLS schema standard.	Sender	rpc:ProcedureNotPresent
1001	The request is not compliant with the RTLS schema standard.	Sender	rpc:BadArguments
1010	This request is not compliant with the *SOAP* schema standard.	Sender	(optional)
9999	There is an internal application error.	Receiver	(optional)

7.2.10 Examples

The examples below are *SOAP* request-response messages that follow the corresponding XML Schema described in the Annex A.

EXAMPLE 1

Method Request:

```
<env:Envelope
      xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body >
    <Query xmlns="http://www.autoid.org/iso24730-1/RTLS-schema">
            <QueryName>RTLS_Blinks</QueryName>
            <FilterBy>
                  <TagID><![CDATA[= 1234]]></TagID>
                  <OR/>
                  <BatteryLow>=true</BatteryLow>
            </FilterBy>
            <Fields>TagID Location BatteryLow</Fields>
            <SortBy>
                  <Order>asc</Order>
                  <Field>TagID</Field>
            </SortBy>
    </Query>
  </env:Body>
</env:Envelope>
```

EXAMPLE 2

Method Response:

```
<env:Envelope
      xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Body>
      <QueryResponse xmlns="http://www.autoid.org/iso24730-1/RTLS-schema"
                             xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
            <rpc:result>QueryResult</rpc:result>
            <QueryResult>
                  <NumItems>1</NumItems>
                  <TagBlinks>
                        <TagBlink>
                              <TagID>1234</TagID>
                              <Location>
                                    <X>23</X>
                                    <Y>34</Y>
                                    <Z>45</Z>
```

```
                    </Location>
                    <States>
                        <BatteryLow>true</BatteryLow>
                    </States>
                </TagBlink>
            </TagBlinks>
        </QueryResult>
    </QueryResponse>
  </env:Body>
</env:Envelope>
```

7.3 Remote Procedure Call: OpenSession

7.3.1 Purpose

This client request starts a session in the RTLS Service, provides query parameters to that session, and returns with an unique identifier for that session. The session in the service maintains the query parameters, so that repeated requests to that session do not require the query parameters to be sent repeatedly.

Unlike the Query RPC, this request creates state in the Service, and QuerySession RPC uses that state, as described in Clause 7.4.

7.3.2 Synopsis

The RPC function signature in C language syntax would look like:

```
struct SessionResponse OpenSession(char* queryName, struct FilterType filters,
struct FieldType fields)
```

Input parameters:

— QueryName

— Filters

— Fields

Output parameters:

— SessionResponse structure

An underlying requirement is that parameters follow the OpenSession method *SOAP* schema, as described in the Annex A, Clause A.3.

7.3.3 Description

The client application shall construct a *SOAP* request message, which will be deserialized at the service to invoke the function above. The *SOAP* request shall contain sections for each input parameter.

The client application is responsible for opening a connection (typically, using HTTP 1.1), and sending the *SOAP* request message. The response will consist of either a SessionResponse structure, or a *SOAP* fault structure. The client application or a library that implements client side proxy functions shall parse the response accordingly.

7.3.4 Input parameters

The input parameters are listed below. The parameters are identical to the parameters for the Query RPC request, defined in Clause 7.2. Unlike the Query RPC, OpenSession RPC does not have a SortBy parameter. QueryName - See Clause 7.2.4,

— Filters — See Clause 7.2.5,

— Fields — See Clause 7.2.6

7.3.5 Output parameter: SessionResponse structure

The SessionResponse structure is described in Clause 8.3. Its XML Schema is defined in the Annex A, in Clause A.8.

7.3.6 Faults

A general description and structure for faults is described in Clause 8.4. The XML Schema is defined in the Annex A, in Clause A.9. Table 3 contains the error codes and related information that shall be included in a *SOAP* fault response to a OpenSession RPC.

Table 3 — Error codes

Error code	**Error message**	**SOAP fault code**	**SOAP fault subcode**
1000	The request is not compliant with the RTLS schema standard.	Sender	rpc:ProcedureNotPresent
1001	The request is not compliant with the RTLS schema standard.	Sender	rpc:BadArguments
1010	This request is not compliant with the *SOAP* schema standard.	Sender	(optional)
2000	The RTLS service is unable to open additional sessions, until some sessions are closed.	Receiver	(optional)
9999	There is an internal application error.	Receiver	(optional)

7.3.7 Examples

The examples below are *SOAP* request-response messages that follow the corresponding XML Schema described in the Annex A.

EXAMPLE 1

Method Request

```
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body >
    <OpenSession xmlns="http://www.autoid.org/iso24730-1/RTLS-schema" >
        <QueryName>RTLS_Blinks</QueryName>
        <FilterBy>
                <TagID><![CDATA[< 5000]]></TagID>
                <OR/>
                <BatteryLow>=true</BatteryLow>
        </FilterBy>
```

```
            <Fields />
            <SortBy>
                        <Order>asc</Order>
                        <Field>TagID</Field>
            </SortBy>
       </OpenSession>
  </env:Body>
</env:Envelope>
```

EXAMPLE 2

Method Response

```
<env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Body>
      <SessionResponse xmlns="http://www.autoid.org/iso24730-1/RTLS-schema"
                                    xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
                  <rpc:result>SessionID</rpc:result>
                  <SessionID>ABJ12300KO</SessionID>
                  <MaxNumBlinks>5000</MaxNumBlinks>
                  <ExpirationSeconds>1800</ExpirationSeconds>
            </SessionResponse>
  </env:Body>
</env:Envelope>
```

7.4 Remote Procedure Call: QuerySession

7.4.1 Purpose

This client request is used to retrieve real-time, tag blinks on a remote computer. The ability to retrieve blinks in real-time is dependent on communication constraints between the client and the remote RTLS Service. This client request retrieves all tag blinks that match the criteria of the session, and which were received by the RTLS Service since the last QuerySession request to this session. Unlike the Query request, more than 1 blink per RTLS transmitter may be returned. By making repeated QuerySession requests, client applications can obtain near real-time data from the RTLS Service.

7.4.2 Synopsis

The RPC function signature in C language would look like:

struct QueryResponse* QuerySession(char* SessionID)

Input parameters:

— SessionID

Output parameters:

— QueryResponse structure

An underlying requirement is that parameters follow the QuerySession method *SOAP* schema, as described in the Annex A, Clause A.4.

7.4.3 Description

The client application shall construct a *SOAP* request message, which will be deserialized at the service to invoke the function above. The *SOAP* request schema for QuerySession RPC is described in the Annex A.

The client application is responsible for opening a connection (typically, using HTTP 1.1), and sending the *SOAP* request message - QuerySession, with its associated parameters. The response shall consist of either a QueryResponse structure or a *SOAP* fault structure. The client application, or a library that implements client side proxy functions, shall parse the response accordingly.

7.4.4 Input parameter: SessionID

Requirements:

1) The client application shall use the same SessionID in the QuerySession RPC, that it received from OpenSession RPC.

2) For each session, there shall be only one client.

3) The client application shall send the SessionID to the same host/service, from which it received the SessionID.

7.4.5 Output parameter: QueryResponse structure

This is the same output parameter that is returned by the Query RPC defined in Clause 7.2.8. Unlike the Query request, however, more than 1 tag blink per RTLS transmitter may be returned. This is because the session returns all tag blinks that match the session's criteria, and that occurred since the last QuerySession request for that session.

QueryResponse structure is described in Clause 8.1 and 8.2. Its *XML* schema is defined in the Annex A in clauses A.6 and A.7.

7.4.6 Faults

A general description and structure for faults is described in Clause 8.4. The XML Schema is defined in the Annex A, in Clause A.9. Table 4 contains the error codes and related information that shall be included in a *SOAP* fault response to a QuerySession RPC.

Table 4 — Error codes

Error code	Error message	SOAP fault code	SOAP fault subcode
1000	The request is not compliant with the RTLS schema standard.	Sender	rpc:ProcedureNotPresent
1001	The request is not compliant with the RTLS schema standard.	Sender	rpc:BadArguments
1010	This request is not compliant with the *SOAP* schema standard.	Sender	(optional)
3000	No session with this SessionID has been found.	Receiver	(optional)
9999	There is an internal application error.	Receiver	(optional)

7.4.7 Examples

The examples below are *SOAP* request-response messages that follow the corresponding XML Schema described in the Annex A.

EXAMPLE 1

Method Request

```
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Body>
    <QuerySession xmlns="http://www.autoid.org/iso24730-1/RTLS-schema">
         <SessionID>
              ADJ909123LOPQ
         </SessionID>
    </QuerySession>
  </env:Body>
 </env:Envelope>
```

EXAMPLE 2

Method Response

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body >
    <QueryResponse xmlns="http://www.autoid.org/iso24730-1/RTLS-schema"
                      xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
        <rpc:result>QueryResult</rpc:result>
        <QueryResult>
            <NumItems>1</NumItems>
            <TagBlinks>
                <TagBlink>
                    <TagID>1234</TagID>
                    <Location>
                        <X>23</X>
                        <Y>34</Y>
                        <Z>45</Z>
                    </Location>
                    <States>
                        <BatteryLow>true</BatteryLow>
                    </States>
                </TagBlink>
            </TagBlinks>
        </QueryResult>
    </QueryResponse>
  </env:Body>
</env:Envelope>
```

7.5 Remote Procedure Call: CloseSession

7.5.1 Purpose

This client request's sole purpose is to inform the RTLS Service that the Session and its associated state are no longer needed by the client application, and that all resources used by that Session may be released.

7.5.2 Synopsis

The RPC function signature in C language would look like:

void CloseSession(char* SessionID)

Input Parameter:

— SessionID

Output Parameter:

— None

An underlying requirement is that parameters follow the CloseSession method *SOAP* schema, as described in the Annex A, Clause A.5.

7.5.3 Description

The client application is responsible for opening a network connection (typically, using HTTP 1.1), and sending the *SOAP* request message. Unlike the other RPC, the output parameter is an empty parameter. In other words, if the session has been closed successfully, the service shall return an empty response. In case of an error, it shall return a *SOAP* fault structure. A client application may ignore either response.

7.5.4 Input parameter: SessionID

Requirements:

1) The client application shall use the same SessionID that it received from OpenSession RPC.

2) The client application shall send the SessionID to the same host and service, from which it received the SessionID.

7.5.5 Output parameter: (none)

This request does not have an output parameter. In terms of the *SOAP* request-response message exchange, this translates to an empty *SOAP* response.

Requirements:

1) If the session has been closed successfully, the service shall respond with a *SOAP* message that has an empty body.

EXAMPLE

```
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Body />
</env:Envelope>
```

7.5.6 Faults

A general description and structure for faults is described in Clause 8.4. The XML Schema is defined in the Annex A, in Clause A.9. Table 5 contains the error codes and related information that shall be included in a *SOAP* fault response to a CloseSession RPC.

Table 5 — Error codes

Error code	Error message	SOAP fault code	SOAP fault subcode
1000	The request is not compliant with the RTLS schema standard.	Sender	rpc:ProcedureNotPresent
1001	The request is not compliant with the RTLS schema standard.	Sender	rpc:BadArguments
1010	This request is not compliant with the *SOAP* schema standard.	Sender	(optional)
3000	No session with this SessionID has been found.	Receiver	(optional)
4010	The session could not be closed in a timely manner.	Receiver	(optional)
9999	An internal application error has occurred.	Receiver	(optional)

7.5.7 Examples

The examples below are *SOAP* request-response messages that follow the corresponding XML Schema described in the Annex A.

EXAMPLE 1

Method Request

```
<env:Envelope
           xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Body >
        <CloseSession  xmlns=" http://www.autoid.org/iso24730-1/RTLS-schema ">
            <SessionID>
                ADJ909123LOPQ
            </SessionID>
        </CloseSession>
    </env:Body>
 </env:Envelope>
```

EXAMPLE 2

Method Response

As per Clause 5.2, a RTLS service may close the session, when a QuerySession request is not received after a vendor-specified period of time (ExpirationSeconds) . If a session is removed in such a way, then the service may return the fault shown below.

```
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:value>env:Receiver</ env>
            </env:Code>
            <env:Reason>
                <env:Text xml:lang="en-US">Application error</env:Text>
            </env:Reason>
```

```
                <env:Detail>
                    <RTLSFaultDetail xmlns ="http://www.autoid.org/iso24730-1/RTLS-schema">
                        <ErrorCode>
                            3000
                        </ErrorCode>
                        <ErrorMessage>
                            No session with this SessionID has been found.
                        </ErrorMessage>
                    </RTLSFaultDetail>
                </env:Detail>
            </env:Fault>
        </env:Body>
</env:Envelope>
```

8 Data structures and data types

The XML Schema for all data structures described below are detailed in the Annex A.

8.1 TagBlink structure

Table 6 contains the list of fields and their description in the TagBlink data structure. The fields are categorized into Required or optional fields. Every RTLS service shall support the Required fields for any Tag Blink. An RTLS Service may omit Boolean fields, when their value is False. For example, instead of sending <BatteryLow>false</BatteryLow>, the service may choose to omit that field in the TagBlink structure.

The unit of length measurement for X, Y, Z, and Distance fields shall be in meters.

The fields, RTLSBlinkTime and LocateTime, shall be in Coordinated Universal Time (UTC), and contain difference of time with UTC.

Table 6 — TagBlink data structure

TagBlink fields	**Required or optional**	**Description**
TagID	Required	Tag identifier. Type: string.
CoordRef	Optional	This provides an identifier for the coordinate reference of this tag. Type: string.
Location	Required	A parent *XML* tag which contains a subset of the following location elements: NoLocate, X, Y, Z, ZoneID, Bearing, and Distance.
NoLocate	Optional [a]	The RTLS System received a tag blink, but was unable to calculate its location. Type: boolean
X	Optional [a]	Location along the X-axis as a float. The Y *XML* tag shall follow it. Type: float.
Y	Optional [a]	Location along the Y-axis as a float. It shall be preceded by X, and may be followed by a Z *XML* tag. Type: float.
Z	Optional [a]	Location along the Z-axis. The X and Y *XML* tags shall precede it. Type: float.
ZoneID	Optional [a]	Zone identifier. Type: string.
Bearing	Optional [a]	Angle from the Reader. It shall be in degrees. It shall be followed by a Distance *XML* tag. Type: float.
Distance	Optional [a]	Distance from the Reader. Type: float.
RTLSBlinkTime	Required	The time when this tag blink was formed. Type: timestamp.
LocateTime	Optional	The last time this RTLS transmitter was located. Type: timestamp.

TagBlink fields	Required or optional	Description
TagModel	Optional	Model of the RTLS transmitter. Type: string.
ResourceType	Optional	The type of asset, the transmitter is associated with. Type: string.
ReaderID	Optional	The Reader that located the tag blink. Type: string.
States	Optional	A parent *XML* tag which contains various states of the RTLS transmitter.
General	Optional	A generic *XML* tag that contains any telemetry data, from the RTLS transmitter. Type: string.
Buttons	Optional	A binary string such as 1011. This example represents 4 buttons, and their states. 1 stands for on, and 0 stands for off. Type: binary string.
ExciterID	Optional	The identifier of a device that causes the RTLS transmitter to blink, when the transmitter passes its vicinity. Type: string.
Motion	Optional	If a tag is moving, this element may be set to true. Type: Boolean.
BatteryLow	Optional	If a tag is running low on battery, this element may be set to true. Type: Boolean.
Blinking	Optional	If the RTLS system determines that the transmitter is blinking, this element may be set to true. Type: Boolean.
Registered	Optional	If the RTLS system determines that the transmitter is pre-registered with it, then this element may be set to true. Type: Boolean.
VendorSection	Optional	A parent *XML* tag that can contain any vendor specific elements. A client application may ignore *XML* tags in this section.

[a] This field appears under the Location parent field. Even though it is shown as optional, the Location parent field shall have at least one of the fields. In addition, the descriptions of some fields above contain requirements regarding other fields.

8.2 QueryResponse structure

Table 7 contains the list of fields and their descriptions in the QueryResponse data structure.

Table 7 — QueryResponse data structure

QueryResponse fields	Required or optional	Description
VendorID	Optional	Identifies the vendor that is delivering RTLS Blinks. Type: string.
QueryResult	Required	A parent *XML* tag that contains the NumItems and TagBlinks array.
NumItems	Required	Number of tag blinks in the response structure. Type: integer.
TagBlinks	Required	It contains an array of TagBlink structures, as described in Clause 8.1.
TagBlink	Required	A single TagBlink structure, as described in Clause 8.1.

8.3 SessionResponse structure

Table 8 contains the list of fields and their descriptions in the SessionResponse data structure.

Table 8 — SessionResponse data structure

SessionResponse fields	Required or optional	Description
SessionID	Required	This is an identifier that uniquely specifies the session created by the QuerySession RPC, on a host. Type: string.
MaxNumBlinks	Optional	The maximum number of blinks that this session will store, before it starts dropping blinks. Type: integer.
ExpirationSeconds	Optional	If a client does not make a request in this time, the service may close the session. Type: integer.
VendorSection	Optional	A parent *XML* tag that can contains any vendor specific elements. A client application may ignore *XML* tags in this section.

8.4 Fault structure

8.4.1 Description

SOAP Version 1.2 defines that "a SOAP Fault element contains two mandatory sub-elements, `env:Code` and `env:Reason`, and (optionally) application-specific information in the `env:Detail` sub-element."[1] The RTLS *SOAP* faults consist of all 3 sub-elements. See Table 9 for a description of this fault structure. See Annex A, Clause A.9, for the schema defining the Detail portion of the fault.

The requirements below are with regard to the faults defined for each API function in this document. Table 9 contains the list of fields and their descriptions in a RTLS fault structure.

Requirements:

— An RTLS Service shall use the *SOAP* fault codes defined for each RPC, when the corresponding error occurs.

— Some fault codes defined for each RPC have subcodes associated with them. The service shall provide the fault subcode, if it has been specified for a given fault code. When a fault subcode is listed as "(optional)", the RTLS Service may provide an arbitrary subcode that qualifies the corresponding fault code.

— The service shall provide the mandatory env:Reason sub-element in the fault response. As per SOAP Version 1.2, this sub-element shall contain one or more env:Text sub-elements. The contents of the env:Text shall be an arbitrary, human readable, generic explanation of the fault. For example: "Internal Application Error". A more specific explanation will be provided in the Detail element of the *SOAP* fault.

— In the Detail sub-element of the *SOAP* fault, the service shall provide the ErrorCode and may provide the corresponding ErrorMessage fields.

— The service shall use the ErrorCode values described for each API function in this standard. The service may provide more specific and descriptive ErrorMessages, by appending additional text to the ErrorMessage text that corresponds with each ErrorCode.

1) *SOAP Version 1.2 Part0: Primer,* W3C Recommendation.

— In addition to faults defined in this standard, the service may provide more specific and descriptive application-specific faults. Any such errors shall have an unique ErrorCode value that is greater than 10000.

Table 9 — RTLS Fault data structure

Fields	**Required or optional**	**Description**
Code	Required	*SOAP* Fault Code. Type: See SOAP Version 1.2 specification
Subcode	Optional	*SOAP* Fault Subcode. Type: See SOAP Version 1.2 specification
Reason	Required	*SOAP* Reason parent *XML* Tag.
Text	Required	*SOAP* Reason Text (one for each locale). See SOAP Version 1.2 specification.
RTLSFaultDetail	Required	A parent *XML* tag that contains the fault parameters below.
ErrorCode	Required	This is a unique identifier of an error condition. Type: integer.
ErrorMessage	Optional	This is a textual description of an error condition. Type: string

8.4.2 Examples

EXAMPLE 1

Below is an example of the fault structure returned when an error occurs. It describes an internal application error, and shows how additional text may be appended to the ErrorMessage.

```
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:value>env:Receiver</ env>
            </env:Code>
            <env:Reason>
                <env:Text xml:lang="en-US">Application error</env:Text>
            </env:Reason>
            <env:Detail>
                <RTLSFaultDetail xmlns ="http://www.autoid.org/iso24730-1/RTLS-schema">
                    <ErrorCode>
                        9999
                    </ErrorCode>
                    <ErrorMessage>
                        An internal application error has occurred.  Unable to connect to the database XYZ.
                    </ErrorMessage>
                </RTLSFaultDetail>
            </env:Detail>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

EXAMPLE 2

Below is an example of the fault structure returned when an error occurs. Such fault may be returned if there is an error in the filter rule of the QueryRequest RPC. Note the additional description "Invalid filter rule" which is appended to the ErrorMessage field. The fault below also exemplifies a *SOAP* RPC fault that has a subcode and an arbitrary reason text – "Request Error".

```
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope"
        xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:Value>env:Sender</env:Value>
                <env:Subcode><env:Value>rpc:BadArgument</env:Value></env:Subcode>
            </env:Code>
            <env:Reason>
                <env:Text xml:lang="en-US">Request Error</env:Text>
            </env:Reason>
            <env:Detail>
                <RTLSFaultDetail xmlns =" http://www.autoid.org/iso24730-1/RTLS-schema">
                    <ErrorCode>1001</ ErrorCode>
                    <ErrorMessage>
                        The request is not compliant with the RTLS schema standard. Invalid filter rule.
                    </ErrorMessage>
                </RTLSFaultDetail>
            </env:Detail>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Annex A
(normative)

XML Schema for Remote Procedure Calls

Any validation schema shall ensure compliance with the XML Schema below.

A.1 RTLS API Schema

The XML Schema below refers to files containing other schemas. These other schemas are also defined in this Annex.

```
<?xml version="1.0" encoding="utf-8" ?>
  <xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace=”http://www.autoid.org/iso24730-1/RTLS-schema"
   version="1.0">
      <xsd:annotation>
            <xsd:documentation>
                  This schema includes the schemas for the methods and their
                  responses. It also defines the RTLS namespace.
            </xsd:documentation>
      </xsd:annotation>
      <xsd:include schemaLocation="Query.xsd"/>
      <xsd:include schemaLocation="QueryResponse.xsd"/>

      <xsd:include schemaLocation="OpenSession.xsd"/>
      <xsd:include schemaLocation="SessionResponse.xsd"/>
      <xsd:include schemaLocation="QuerySession.xsd"/>
      <xsd:include schemaLocation="CloseSession.xsd"/>

      <xsd:include schemaLocation="FaultDetails.xsd" />

  </xsd:schema>
```

A.2 SOAP Request Schema: Query

```
<?xml version="1.0" encoding="utf-8" ?>
  <xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:rtls="http://www.autoid.org/iso24730-1/RTLS-schema"  version="1.0">
    <xsd:element name="Query">
          <xsd:complexType>
          <xsd:sequence>
                        <xsd:element name="QueryName" type="xsd:string"/>
                              <xsd:element name="FilterBy" type="FilterType"/>
                              <xsd:element name="Fields" type="FieldsType" />
                              <xsd:element name="SortBy" type="SortType" />
                          </xsd:sequence>
              </xsd:complexType>
    </xsd:element>
```