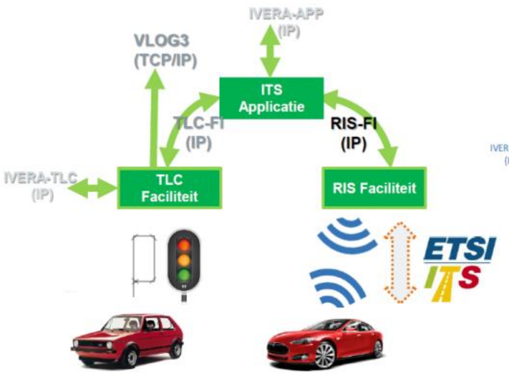
Intelligente Verkeers Regel Installatie (iVRI) – Fase 2

Deliverable 1ab: IDD Generic-FI

Interface Design Description Generic-FI



Datum: 2 december 2016
Versie: 1.1

VOORWOORD

In mei 2016 is opdracht verstrekt door het Ministerie van Infrastructuur en Milieu via het Beter Benutten Vervolg (BBV) programma aan vier VRA leveranciers om te komen tot een gezamenlijke definitie van VRA standaarden ten behoeve van connected en coöperatieve functionaliteit.

Dit document vormt Deliverable 1ab van de afgesproken leverdelen in de opdrachtverstrekking, omschreven als “IDD Generic-FI”.

Deze deliverable beschrijft in het Engels het koppelvlak van het verkeersregeltoestel naar de verschillende mogelijke C-ITS-applicaties.

Dit document is tot stand gekomen door samenwerking van de vijf leveranciers in de werkgroep bestaande uit:

Inge Fløan
Eric Koenders

Peter Smit

Wim Nouwens
Jeroen Hiddink

Benno Geels

NB. De rest van dit document is geschreven in het Engels om internationale uitwisseling te ondersteunen.

The rest of this deliverable has been written in English to facilitate international exchange.

DOCUMENT CONTROL SHEET

Document versions:

Version	Date	Author	Comment
0.1	2016-06-06	WG2	Initial draft
0.2	2016-06-22	WG2	Rework after review
0.3	2016-06-30	WG2	Rework after review
0.4	2016-07-05	WG2	Rework after review
0.5	2016-07-08	WG2	Rework after review
0.9	2016-07-14	WG2	Final draft
0.91	2016-08-09	WG2	Rework of 0.9 comments
0.92	2016-08-16	WG2	Rework after review meeting
1.0	2016-08-25	WG2	Rework after review
1.1	2016-11-15	WG2	§4.2 : clarified relations between TCP-ports and security option §4.3.2 : clarified serverside certificate verification §4.5: added JSON minimum supported message size §4.6. Figure 2: correct presentation/session-layer §5.5 Clarified replies and events sent in actions Table 1: added conditions 'RegistrationTimeout' and 'Already Registered' Table 2: condition "RegistrationRequest.username = Appl.username" removed, and added action "Send DeregistrationReply" §5.7: clarified timing descriptions, added 'Successful registration interval' §5.9: back-off: define successful connection, added reset-condition §6.3 : Changed ApplicationUsername description. §6.5 : added additional information to SessionEvent. Added ranges to SessionEventCode. §9.1: added exception 'socket error' §9.2: removed 10 sec delay after incorrect login §9.2: Exception 1: added reference to back off algorithm §9.2: exception "No Registration" : concretize alive timeout reference §9.2: added exception "deregistration from a not registered ITS-A" §9.5: various exceptions: added 'Close Connection' action. Removed "mandatory attribute" exception. Fixed typographical errors

Approval:

	Who	Date	Version
Prepared			
Reviewed			
Approved			

Publication level: Public

Version filename: iVRI2_del_1ab_IDD_Generic-FI_v1.1.docx

CONTENT

Introduction

1.1 Overview

The iTLC architecture [Ref 1] defines several interfaces of the iTLC. Two of these interfaces have common features, these interfaces are the

- Traffic Light Controller Facilities Interface (TLC-FI), used to interact with a Traffic
1 Light Controller for instance for acquiring detector status and request actuation of output signals. (see [Ref 2].)
- Roadside-ITS-Station Facilities Interface (RIS-FI). Used to interact with a RIS for instance for obtaining positions of C-ITS Vehicles and to distribute events to C-ITS stations in the range of the RIS.

These two interfaces are shown in the following figure:

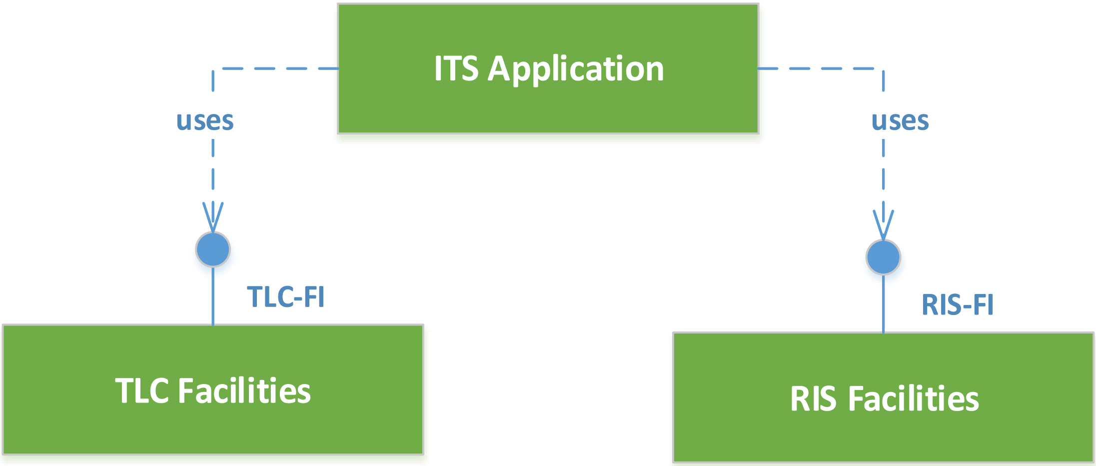


Figure 1 System overview Facilities Interfaces

The RIS-FI and TLC-FI are robust interfaces between (external) ITS Applications and the respective facilities.

The TLC- and RIS-FI share common technical requirements and as ITS Applications will communicate with both, it is chosen to design the interfaces on common technological base, such as transport protocols and security as well as on a common information transaction model.

These common functions and interactions are the subject of this document while separate documents for the TLC ([Ref 4]) and RIS ([Ref 5]) provide domain specific information description and functional use-cases. The intention is that the specific documents can be described communication technology agnostic.

In the remainder of this document, when generic traits of RIS-FI and TLC-FI are described, the Facilities Interface is called the X-FI. Furthermore, Facilities denotes both RIS Facilities and TLC Facilities.

1.2 Purpose and scope

This document describes the interface design of the X-FI with respect to

- Underlying technologies,
- Information transaction model,

- Generic protocol and transaction methods,
- Generic objects,
- Use-cases / interactions and
- Error / exception handling.

1.3 Advise for the reader

It is advised that the reader understands the iTLC Architecture as described in *iTLC Architecture WG3 (Deliverable F) v 1.2, jan. 2016* ([Ref 1]) as well as the requirements in *Beter Benutten Vervolg, project iVRI, Deliverable G2, IRS TLC Facilities Interface v1.2, jan 2016* ([Ref 2]) and *Beter Benutten Vervolg, project iVRI, Deliverable G1, IRS RIS Facilities Interface v1.2, jan 2016* ([Ref 3]).

1.4 Document conventions

To identify an Object and its attributes, the following format is used:

<Object type name>.<attribute name>

For instance for the *AliveObject*, which has an attribute *tick* is identified as *AliveObject.tick*

This document contains decision tables to describe logic, these tables are formatted as follows:

CONDITIONS	condition 1	N	Y	Y
	condition 2	-	Y	Y
	condition 3	-	N	Y
ACTIONS	ERROR: failure 1 encountered		√	
	ERROR: failure 2 encountered			√
	Execute action			√

Several CONDITIONS are used to indicate which conditions must be valid for any number of ACTIONS.
Boolean CONDITIONS are used.

- Y = Yes, the condition is valid
- N = No, the condition is not valid
- \- = Conditions doesn't matter for the actions

The ACTIONS taken are indicated with a checkmark (√)

References

ID	Reference
[Ref 1]	*iTLC Architecture WG3 (Deliverable F) v 1.2, jan. 2016*
2 *[Ref 2]*	*Beter Benutten Vervolg, project iVRI, Deliverable G2, IRS TLC Facilities Interface v1.2, jan 2016*
[Ref 3]	*Beter Benutten Vervolg, project iVRI, Deliverable G1, IRS RIS Facilities Interface v1.2, jan 2016*
[Ref 4]	*Beter Benutten Vervolg, project iVRI – fase 2, Deliverable 1a IDD TLC Facilities Interface v1.1, dec 2016*
[Ref 5]	*Beter Benutten Vervolg, project iVRI – fase 2, Deliverable 1b IDD RIS Facilities Interface v1.0, dec 2016*
[Ref 6]	*JSON-RPC 2.0 Specification* *http://www.jsonrpc.org/specification*
[Ref 7]	*The Transport Layer Security (TLS) Protocol Version 1.2* *RFC 5246, https://tools.ietf.org/html/rfc5246*
[Ref 8]	*The JavaScript Object Notation (JSON) Data Interchange Format* *RFC 7159, https://tools.ietf.org/html/rfc7159*
[Ref 9]	*IRS Security v1.1, oct 2016*
[Ref 10]	*Uniform Resource Identifier (URI): Generic Syntax, RFC 3986* *https://www.ietf.org/rfc/rfc3986.txt*

Acronyms, abbreviations and concepts

Acronyms and abbreviations

3

C-ITS	Cooperative ITS functionality for exchange of data between in-vehicle and/or road side devices making use of either cellular or short range wireless communication
IDD	Interface Design Description
IRS	Interface Requirements Specification
iTLC	Intelligent TLC performing traffic light controller functions and allowing for ITS applications
ITS	Intelligent Transport Systems
ITS Station	Functional entity specified by the ITS station reference architecture (see [Ref 1])
ITS-A	ITS Application
ITS-CLA	ITS Control Application
ITS-CRA	ITS Consumer Application
ITS-PRA	ITS Provider Application
iVRI	See iTLC
RIS	Roadside ITS Station
TLC	Traffic Light Controller; controls signals of one or more intersections
UTC	Coordinated Universal Time

Concepts

Traffic Control Application	Application which implements a traffic control algorithm and is able to request signal group states
ITS Control Application	A Traffic Control Application which uses TLC- and/or RIS-interfaces
ITS Application	An application which supports one or more ITS use-cases. Range of possible ITS Applications include an ITS Control Application
TLC Facilities	Component providing facilities of a TLC to users (internal and/or external). Includes amongst others: • Access to information from the TLC • Services to trigger actuators
RIS Facilities	Component providing facilities of a RIS to users (internal and/or external).

4 Technical description

4.1 Introduction

The X-FI allows ITS Applications to access data stored in a TLC or RIS through an Internet Protocol based network. Multiple ITS Applications can interact with the TLC concurrently.

4.2 Network connections

The X-FI protocol is based on a bi-directional connection using TCP/IP. The Facilities offers a TCP port at which it listens to socket connections. ITS Applications are aware of the TLC IP address and TCP port prior to deployment.

Depending on the specific site security implementation (see [Ref 9]), communication over the TCP port may or may not need to be secured.

Default TCP ports for the different Facilities are listed in the following table:

Facilities	Port
TLC Facilities (TLS)	11001
TLC Facilities (no security)	11501
RIS Facilities (TLS)	12001
RIS Facilities (no security)	12501

4.3 Network security

The security of connections between an ITS Application and the Facilities and therefore the privacy, authenticity and integrity of the data exchanged using the X-FI is ensured through means of a (Virtual) Private Network and/or Transport Layer Security (TLS).

Which of the methods is used depends on the situation and security requirements.

4.3.1 Private network

The ITS Application(s) and the Facilities are placed within a private network (See [Ref 9]).

VPN ensures confidentiality (secure against eavesdropping) and integrity (secure against data manipulation) against systems outside the private network when using unsafe underlying networks to communicate.

Data exchanged within the private network is not confidential nor guarded against manipulations, so it is required that the ITS Applications used within one VPN are well-behaved / certified and that there is no need to protect these applications from other applications within the private network.

4.3.2 TLS

The TLS protocol creates a secured channel between a single ITS Application and the Facilities. The channel created is confidential and the data integrity is assured against other applications communicating with the Facilities within the same private network.

At a minimum TLS version, 1.2 is used, with server-side certificate verification: the ITS Application can verify the authenticity of the Facilities with a certificate.
This implies:

- private key handling
- certificate deployment

There is no need for the Facilities to authenticate the ITS Application in this stage as this is done separately within the application layer (username, password) after the secured channel has been established, please refer to 8.1 for the use-case describing this procedure. Chosen security shall be based on recommendations in RFC7525 (https://tools.ietf.org/html/rfc7525).

The following cipher suites are recommended by RFC7525. The Facilities determines which cipher is used.

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

4.4 Data encoding - JSON

All data exchanged is encoded using JavaScript Object Notation (JSON) (see http://www.json.org/), which is a flexible lightweight, versatile data-interchange format easy to read for humans and easy to parse by computers.

“An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, Boolean, null, object, or array.” (See [Ref 8])

For the following types (case insensitive) as used in the Object-definitions, the addition to the JSON definition in [Ref 8] is:

Float

[minus] int [frac]

Integer

[minus] int

4.5 Data transport

JSON-RPC is used as transport protocol, this is defined in chapter 11.
The minimum supported JSON-RPC message size is 32kBytes.

4.6 JSON-RPC usage for X-FI

JSON-RPC is an Application layer protocol. It assumes an underlying stream connection between two peers.

The ITS Application and Facilities will both act as a JSON-RPC client and server using one TCP session.

(1) The ITS Application client accesses data and sets subscriptions in the Facilities server. The server in the Facilities responds to these requests, possibly with accompanying data. Protocol handling such as authentication and authorisation and alive checking takes place.

(2) The Facilities client sends notifications with data the ITS Application needs to receive and executes protocol handling such as alive checking.

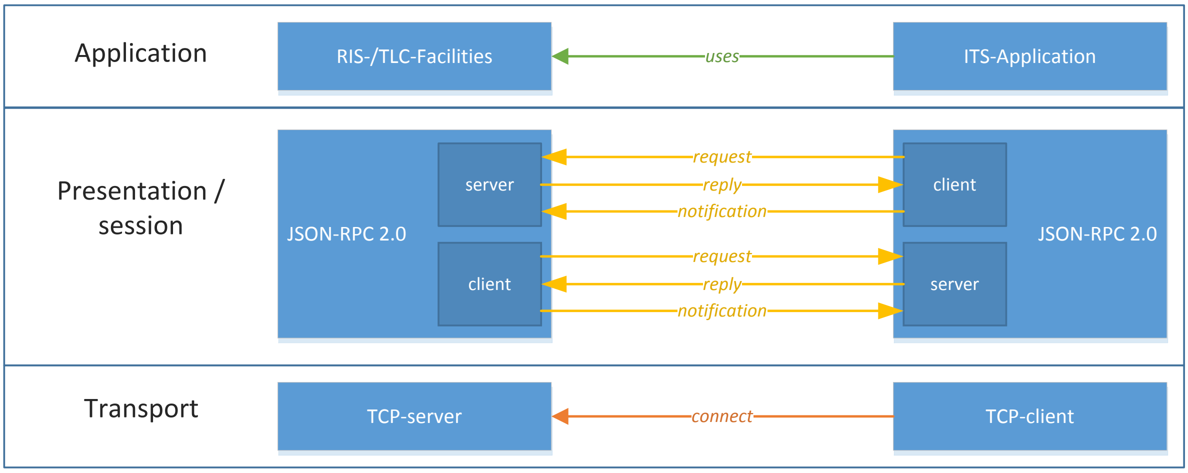


Figure 2 Network layers

Functional description

5.1 Objects
The RIS and TLC Facilities exchange different types of information with ITS Applications. The information is exchanged as TLC and/or RIS Objects.

This document uses the name **FI Object** to define objects common for TLC and RIS as well as to describe traits common for a TLC or RIS Object.

There are **two categories** of FI Objects:
- **FI State objects**. These objects describe physical or logical entities and their states. The objects are uniquely identifiable and typically exists throughout the lifetime of the RIS or TLC instance. Examples of such objects are signal groups and detectors containing states such as external signal group state and detection input state.
- **FI Event objects**. These objects convey the occurrence of a specific event related to a specific FI State object. These objects can be seen as generated by FI State Objects. Such an event can for instance contain a vehicle message (KAR or C-ITS) or a speed and length detected by a loop detector.

5.2 Time reference
When State Objects are being synchronized or Event Objects are sent, a time reference object is always sent. Both the Facilities and the ITS-A sends this object. The time reference contains the current relative time-tick of the sender.

The relative **time-tick** is an ever-increasing unsigned integer value. Every increment of 1 of the timer-tick corresponds with 1ms incremented time of the sender.

The time-tick is a 32-bit unsigned integer
- Range: 0 to 4294967295
- Always incrementing
- On overflow, the value takes actual interval into account and wraps properly back to a value making sure that it is always possible to deduct the previous interval.

A running time-tick is independent of updates of the calendar time.

5.3 Calendar time (UTC)
Both an ITS-A and Facilities maintain a notion of the calendar time, the calendar time is expressed as the UTC time. The **UTC time** gives the actual UTC time of the sender (in ms resolution). There may be jumps in this time as it is being kept up-to-date by for instance NTP.

5.4 Method categories
The following categories of methods exist within the X-FI:
1. Protocol methods
2. Data access methods
3. Data subscriptions and notifications

5.4.1 Protocol methods
The protocol methods manage and support the protocol connections, such as listing protocol methods and object types, authentication and authorisation methods, activation methods for ITS Control Applications and alive checking.

5.4.2 Data access methods
These are methods used to read, update, create and delete data of the Facilities. TLC Objects are accessed, such as signal group status, intersection modes and detection data. RIS Objects are accessed, such as Events and ITS Stations.

5.4.3 Data subscriptions and notifications
Subscription methods are used to manage subscriptions to changes of FI State Objects and/or generation of FI Event Objects. The notification methods convey the changed FI State objects or the generated FI Event Objects.

When an ITS Application owning a subscription is disconnected from the Facilities or an Alive error occurs, the subscriptions will be removed by the Facilities and the ITS Application needs to subscribe to the requests again when it reconnects.

5.5 Session States

An ITS Application can connect to the Facilities to create a session. The following diagram shows the states of such an application.

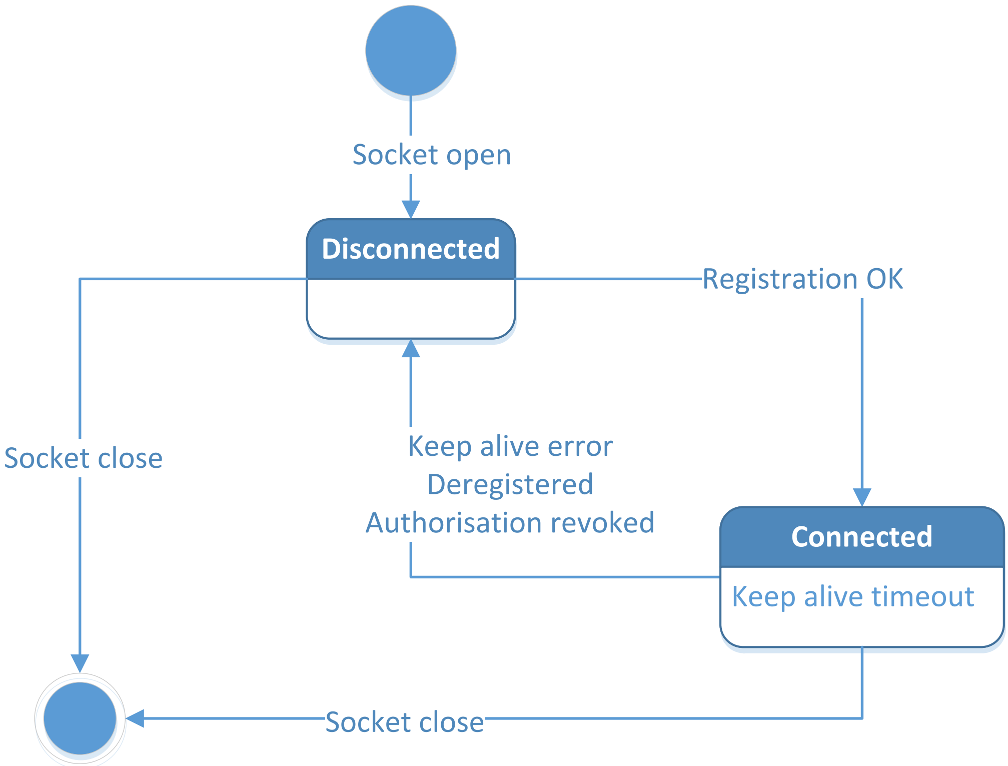


Figure 3 Session state diagram

The transition of an Application object between these states is described by the decision tables below:

Table 1 Application in session state Disconnected - Facilities decision table

CONDITIONS	Application session state = **Disconnected**	N	Y	Y	Y	Y	Y	Y	Y	Y
	RegistrationRequest Received	-	N	Y	Y	Y	Y	Y	Y	N
	Application registration timeout									Y
	RegistrationRequest.version = supported	-	-	N	Y	Y	Y	Y	Y	-
	RegistrationRequest.username = Application.username	-	-	-	N	Y	Y	Y	Y	-
	Application.username already registered							N	Y	-
	RegistrationRequest.password = Application.password	-	-		-	N	Y	Y	-	-
	RegistrationRequest.type = Application.type	-	-	-	-	-	N	Y	-	-
ACTIONS	invalid protocol Send error response with ProtocolErrorCode = InvalidProtocol			√						
	unknown application username Send error response with ProtocolErrorCode = NotAuthorised				√					
	invalid password Send error response with ProtocolErrorCode = NotAuthorised					√				
	invalid role Send error response with ProtocolErrorCode = NotAuthorised						√		√	
	Create unique session identifier							√		
	Set Application session state = Connected							√		
	Send RegistrationReply							√		
	Close socket			√	√	√	√		√	√
	Log the state transition.							√		
	Log error situation			√	√	√	√		√	√

Table 2 Application in session state Connected - Facilities decision table

CONDITIONS	Application session state = **Connected**	N	Y	Y	Y	Y	Y
	Alive check = OK	-	N	Y	-	Y	Y
	Authorisation revoked	-	-	N	Y	N	N
	RegistrationRequest received	-	-	N	-	Y	N
	DeregisterRequest received	-	-	N	-		Y
ACTIONS	Alive check failed		√				
	authorisation revoked Send SessionEvent with SessionEventCode = Deregistered				√		
	application is already Connected Send error response with ProtocolErrorCode = NotAuthorised					√	
	Send DeregisterReply						√
	Set Application session state = Disconnected						√
	Log the state transition.						√
	Terminate session, close socket		√		√	√	
	Log error situation		√		√	√	

5.6 Alive checking

The Alive checking mechanism is part of the Protocol methods. An ITS Application which has been registered must continuously send an Alive message and it assumes that it receives a similar alive message from the Facilities with the same interval. The required timing is defined in 0.

It is the responsibility of the sender of the Alive messages that the underlying system is alive and functioning properly. When the underlying system is not functioning properly to handle the requests at the interface that the Alive message should not be sent even if the Alive generation procedure can function in isolation.

If any of the involved parties fails to receive the message from the other within 2.5 * interval, the verifying party will assume the connection broken and may break off the connection.

The Alive check starts after an application has successfully logged in.

As part of the Alive checking, the sender provides the following objects:

- Actual **UTC time** (in milliseconds)
- Actual **time-tick** (in milliseconds)

5.7 Timing

This section contains timing parameters.

Table 3 Timing parameters

Item	Time	Description
Alive interval ITS-CLA (both directions)	2s	Time between alive messages between an ITS-CLA and the Facilities
Alive interval ITS-A (both directions)	10s	Time between alive messages between a non-ITS-CLA application and the Facilities
Successful registration interval	42s	Time between successful Registration requests. When an ITS-A is disconnected, it shall wait this time before trying to Register again.

5.8 Protocol versions

The TLC-FI and the RIS-FI both implement a certain version of the X-FI protocol definition. The versioning scheme of the protocol definition is as follows: major.minor.revision

	Description
major	Used to indicate major technical or functional change to the protocol. Each IDD of a TLC-FI or RIS-FI shall identify compatibility of the protocol, methods and objects. All ITS-A's shall be able to communicate with Facilities using the same major version. Possibly breaking backwards compatibility.
minor	Increasing values are used to indicate minor changes to the protocol-definition. Compatibility is guaranteed
revision	Minor changes to specification, clarifications and typographical errors. Compatibility is guaranteed

An ITS Application is responsible for using a proper protocol version when communicating with the Facilities. In 9.3 exception handling for differences in the protocol versions are handled.

An X-FI implementation shall reject connection by ITS-A's using unsupported protocol versions. The X-FI defines which version of this Generic-FI document is used and which versions are supported / unsupported.

5.9 Back off procedure

It is possible that an ITS-A cannot connect (failure during TCP socket connect, TLS-negotiation or RegistrationError) with the Facilities, in this case it will follow a back off strategy when trying to reconnect. This strategy shall involve a relatively quick retry mechanism the first times it fails and the time between connection attempts shall increase until a maximum. The minimum time an ITS-A shall wait to reconnect is defined in the following table:

count	minimal retry-timeout
1 .. 5	1 sec
6..10	2 sec
11..20	5 sec
21..25	30 sec
>25	60 sec

The minimal retry-timeout will be reset to the value belonging with 'count 1...5' when a successful connection (Registration succeeded) is created."

6 Objects

This chapter describes Object-definitions of types used in both TLC-FI and RIS-FI.

6.1 Template FI Object definition

Definition of FI Objects are described by using the following standard notations.

<OBJECTNAME>

Descriptive name	Short name of object type	
Definition	Definition of the object; where applicable including usage of the object and its attributes	
Representation	One of the standardized types (see section 0)	optional additional description
Range	From X to Y	range may include keywords like “ENUM” (see section 6.1.1)
Unit	unit, where applicable	e.g. ‘m/s’ or ‘second’

<COMPOSITE_OBJECT>

Descriptive name	CompositeObject	
Definition	Text describing the composite object, in this case this would be something like "A general object containing other objects".	
Access	Defines access rights to the Object for each application type. Access right is defined by R = Read, the application type is allowed to read this object. Actual access restriction may apply for each attribute. W = Write, the application type is allowed to write this object. Actual access restriction may apply for each attribute.	

Representation	{
	meta {
	ObjectId id
	}
	state {
	Objectname a R/W
	}
	Objectname b
	
	Objectname c[]
	
	}
		Representation may include keywords like "CHOICE" or "ENUM".
		
		Keywords 'meta' and 'state' are optional and can be used according to section 6.1.1.
		
		With each attribute, access rights (R/W) may be defined. An application type allowed to write an object may only write attributes that are labelled W
		
		an attribute may contain an array of objects, this is indicated with the square brackets '[' and ']'

Range	N/A	
Unit	N/A	

6.1.1 Keywords

<u>Meta</u>

The meta keyword defines the scope within an Object-definition in which meta-attributes are defined, for example:

```
Meta {
        ObjectID    id
        String      name
        …
}
```

The meta-data is returned as reply after explicit ReadMeta request.

During JSON-encoding, the keyword "meta' will not be included in the JSON-stream.

State

The state-keyword defines the scope within an Object-definition in which state-attributes are defined, for example:

```
State {
SignalGroupState        requestedState…
FaultState              fault
}
```

The attributes within the state-scope will be returned as part of an ObjectStateUpdate notification.

During JSON-encoding, the keyword “state’ will not be included in the JSON-stream.

ENUM

attribute is of type Integer, with range according to specified enumeration

CHOICE

attribute of one of the type Objects as specified in the choice-scope

abstract

Referred object cannot be instantiated directly, use CHOICE-keyword to indicate possible concrete object types

<OPT>

All object attributes are mandatory, except for attributes marked with the keyword <OPT>, they may be omitted.

E.g. the elevation attribute is optional:

```
{
Float   latitude
Float   longitude
Float   elevation       <OPT>
}
```

If for a mandatory attribute the value is not known, this must be indicated by using the attribute values indicating “unknown” when it exists. Otherwise the JSON value null shall be used.

<Object-Type>

The description contains generic types which contains reference to a specific instance of an ObjectType. To explicitly define which object-type the attribute must contain, the keyword <object-type> is added to the attribute type.

E.g. the intersection attribute of the following definition must contain Object identifiers (ObjectID) referencing an object of type *Intersection.*

```
{
Meta {
        ObjectID                    id
        ObjectID<Intersection> intersection
}
}
```

6.2 Base

Length

Descriptive name	Length
Definition	Length. The value shall be set to null if the information is unavailable.
Representation	Float
Range	0 to 429496729.5
Unit	meter

Location

Descriptive name	A geographical location
Definition	This object describes a WGS84 location

Representation	{
	Float latitude
	Float longitude
	Float elevation <OPT>
	}

Range	latitude from -90.000000 to 90.000000 longitude from -180.000000 to 180.000000 elevation from -100.000 to 8000.000
Unit	latitude in degrees longitude in degrees elevation in meters

ObjectData

Descriptive name	Object update
Definition	An object describing the data of one or more objects. The ObjectData is the contents of the Object except the Meta{} scope. The update of all objects mentioned in objects is atomic. The ticks attribute defines the tick at which the data update is sent.

Representation	{
	ObjectReference objects
	abstract ObjectDataContent data[]
	Ticks ticks
	}

Range	N/A
Unit	N/A

ObjectDataContent (abstract)

Descriptive name	Object data
Definition	Abstract object type to group all data of objects. The contents is defined by the object itself containing all attributes except the *Meta{}* scope.
Representation	N/A
Range	N/A
Unit	N/A

ObjectEvent

Descriptive name	Object event update object	
Definition	An object describing an update containing events generated from one or more objects. ObjectEventContent is an abstract type which can contain events generated by all objects. The ticks is the time at which the data of the events is detected.	
Representation	{	
	ObjectReference	objects
	abstract ObjectEventContent	events[]
	Ticks	ticks
	}	
Range	N/A	
Unit	N/A	

ObjectEventContent (abstract)

Descriptive name	Object event data
Definition	Abstract object type that is used to group all event data objects can generate. The contents is defined by the object itself.
Representation	N/A
Range	N/A
Unit	N/A

ObjectID

Descriptive name	Object Identifier
Definition	A unique identifier for an object instance per ObjectType. Recommendation is to use functional names of the objects, for instance "D02" for a detector, signal group "FC02"
Representation	String
Range	Allowed characters: 'a-z' (ASCII 97 through 122), 'A-Z' (ASCII 65 through 90), '0-9', '_' (underscore, ASCII 95) and '–' (hyphen, ASCII 45).
Unit	N/A

ObjectMeta

Descriptive name	Object Meta data	
Definition	An object describing the Meta data one or more objects. The ObjectMeta is the contents of the Meta{} scope identifier. Of the Object The ticks attribute defines the tick at which the data update is sent.	
Representation	{	
	ObjectReference	objects
	abstract ObjectMetaContent	meta[]
	Ticks	ticks
	}	
Range	N/A	
Unit	N/A	

ObjectMetaContent (abstract)

Descriptive name	Object meta data
Definition	Abstract object type to group all meta data of objects. The contents is defined by the *Meta{}* scope identifier of the object
Representation	N/A
Range	N/A
Unit	N/A

ObjectReference

Descriptive name	Object reference
Definition	A reference to a number of objects of the same type

Representation	{
	abstract ObjectType type
	ObjectID ids[]
	}

Range	N/A
Unit	N/A

ObjectStateUpdate

Descriptive name	Object state update
Definition	An object describing a state update of one or more objects. The ObjectState is the contents of the State{} scope of the objects. The update of all objects mentioned in objects is atomic.

Representation	{
	ObjectReference objects
	abstract ObjectStateUpdateContent states[]
	}

Range	N/A
Unit	N/A

ObjectStateUpdateContent (abstract)

Descriptive name	Object state
Definition	Abstract object type to group all states of objects. The contents is defined by the *State{}* scope identifier of the object.
Representation	N/A
Range	N/A
Unit	N/A

ObjectStateUpdateGroup

Descriptive name	Group Object state update
Definition	This object is used to define a group of object state updates. The different state updates are in the update attribute. The ticks attribute defines the tick from which the states in the update are valid.

Representation	{
	ObjectStateUpdate update[]
	Ticks ticks
	}

Range	N/A
Unit	N/A

ObjectType (abstract)

Descriptive name	Object type
Definition	Abstract object type to group the types of objects supported by a Facilities Interface. Each Facilities Interface implements its own types.
Representation	N/A
Range	N/A;
Unit	N/A

ProtocolErrorCode

Descriptive name	Error code
Definition	Error code used for protocol requests, this is an extension of JSON –RPC errors being passed as the code attribute of an error object. See section 11.
Representation	Integer

Range	ENUM {
	Error (0)
	NotAuthorised (1)
	NoRights (2)
	InvalidProtocol (3)
	AlreadyRegistered (4)
	UnknownObjectType (5)
	MissingAttribute (6)
	InvalidAttributeType (7)
	InvalidAttributeValue (8)
	InvalidObjectReference (9)
	}
	0 through 999 : Generic Error codes
	1000 through 1999 : TLC-FI Error codes
	2000 through 2999 : RIS-FI Error codes

Unit	N/A

SessionID

Descriptive name	Session Identifier
Definition	An identifier unique for a session with the Facilities. This is a specific type of ObjectID used only between two peers, other ITS-A cannot use this ID to obtain information about the session.
Representation	See ObjectID
Range	See ObjectID
Unit	See ObjectID

Speed

Descriptive name	Speed
Definition	Speed value in meters per second. When the information is not available, the value shall be set to null.
Representation	Float
Range	0.0 to 99.0
Unit	meter / second

Ticks

Descriptive name	A time represented as a number of ticks
Definition	A tick is the basic unit of relative time for an application and a Facilities Interface, per session (values between sessions are not related). The value wraps around when the maximum is reached.
Representation	Integer
Range	From 0 to 4294967295
Unit	1 millisecond

Timestamp

Descriptive name	A time stamp
Definition	The number of milliseconds since 1-1-1970 00:00:00 UTC
Representation	Integer
Range	From 0 to 18446744073709551615
Unit	1 millisecond

6.3 Registration

RegistrationRequest

Descriptive name	A registration request
Definition	This object describes the contents of a registration request

Representation	{
	ApplicationUsername username
	ApplicationPassword password
	ApplicationType type
	ProtocolVersion version
	ApplicationURI uri
	}

Range	N/A
Unit	N/A

RegistrationReply

Descriptive name	A registration reply
Definition	This object describes the contents of a registration reply. The sessionid is a unique identifier created by the Facilities to identify this session. All session communication uses this identifier. facilities : reference to the Facilities with which this session is active

Representation	{
	SessionID sessionid
	ObjectReference facilities
	ProtocolVersion version
	}

Range	N/A
Unit	N/A

ApplicationPassword

Descriptive name	Application password
Definition	Definition of an application's password
Representation	String
Range	Values 32 through 126 from the ASCII character set, except ' " ' (double quotes, ASCII 34) and ",“ (comma, ASCII 44)
Unit	N/A

ApplicationURI

Descriptive name	Application uniform resource identifier.
Definition	Gives information of an application.
Representation	String
Range	Values 32 through 126 from the ASCII character set, except ' " ' (double quotes, ASCII 34) and ",“ (comma, ASCII 44) AND Further limited by the characters allowed by the URI generic syntax in [Ref 10]
Unit	N/A

ApplicationUsername

Descriptive name	Application username
Definition	Defines the username of an application. Is used by registration to create a session. The username is not case-sensitive.
Representation	String
Range	Allowed characters: 'a-z' (ASCII 97 through 122), 'A-Z' (ASCII 65 through 90), '0-9', '_' (underscore, ASCII 95) and '–' (hyphen, ASCII 45). A username always starts with a letter.
Unit	N/A

ApplicationType

Descriptive name	Application types
Definition	Consumer: is allowed to read and subscribe to changes of FI Objects Provider: has the same rights as a consumer, but can in addition provide data to the Facilities through the X-FI Control: has the same rights as a Provider, but can in addition control exclusive resources of the Facilities through the X-FI
Representation	Integer

Range	ENUM {
	Consumer (0)
	Provider (1)
	Control (2)
	}

Unit	N/A

ProtocolVersion

Descriptive name	Protocol version
Definition	Structure containing the protocol version.

Representation	{
	Integer major
	Integer minor
	Integer revision
	}

Range	major: 0 – 1000 minor: 0 – 1000 revision: 0 - 1000
Unit	N/A

6.4 Deregistration

DeregistrationRequest

Descriptive name	A de-registration request
Definition	This object describes the contents of a de-registration request.
Representation	{ }
Range	N/A
Unit	N/A

DeregistrationReply

Descriptive name	A deregistration reply
Definition	This object describes the contents of a deregistration reply. The result is empty.
Representation	{ }
Range	N/A
Unit	N/A

6.5 Session

SessionEvent

Descriptive name	A session event
Definition	This object describes an event generated within a session.

Representation	{
	SessionEventCode code
	SessionEventInformation info <OPT>
	}

Range	N/A
Unit	N/A

SessionEventCode

Descriptive name	Session event code
Definition	Code defining an event for the Session.
Representation	Integer

Range	ENUM {
	Deregistered (0)
	FacilitiesStopping (1)
	}
	0 through 999 : Generic codes
	1000 through 1999 : TLC-FI codes (see explanations in [Ref 4])
	2000 through 2999 : RIS-FI codes (see explanations in [Ref 5])

Unit	N/A

SessionEventInformation

Descriptive name	A session event information object
Definition	This object describes additional information related to a session event. Which object and attribute caused the event. Attribute contains the string representing the attribute.

Representation	{
	ObjectType type
	ObjectID id
	String attribute
	}

Range	N/A
Unit	N/A

6.6 Alive

AliveObject

Descriptive name	An alive object	
Definition	This describes an Alive object	
Representation	{	
	Ticks	ticks
	Timestamp	time
	}	
Range	N/A	
Unit	N/A	

7 Methods

7.1 Register

This method is used to register an Application with the Facilities.

Request:

Method: Register		
Parameter name	**Type**	**Description**
params	RegistrationRequest	Registration object containing login information

Result:

Parameter name	**Type**	**Description**
result	RegistrationReply	Result of the registration request

Error:

Parameter name	**Type**	**Description**
code	ProtocolErrorCode	Error code
message	String	optional message

7.2 Deregister

This method is used by an ITS Application to de-register from the Facilities.

Request:

Method: Deregister		
Parameter name	**Type**	**Description**
params	DeregistrationRequest	Deregistration object containing logoff information

Result:

Parameter name	**Type**	**Description**
result	DeregistrationReply	Result of the de-registration

Error:

Parameter name	**Type**	**Description**
code	ProtocolErrorCode	Error code
message	String	optional message

7.3 Alive

This method is used by an ITS Application and the Facilities to send Alive messages to the peer.

Request :

Method: Alive		
Parameter name	**Type**	**Description**
Params	AliveObject	Alive object

Result:

Parameter name	**Type**	**Description**
result	AliveObject	Alive object received is returned to the sender

Error:

Parameter name	**Type**	**Description**
code	ProtocolErrorCode	Error code
message	String	optional message

8 Functional use-cases

8.1 Establish connection with the Facilities

Name	**Establish stable connection with the Facilities**
Description / context	An ITS Application is started and initiates connection with the Facilities, methods and objects exchanged are described.
Actor	ITS Application
Goal	The ITS Application is authenticated and authorised to be connected with the Facilities.
Pre-condition(s)	ITS Application is configured with - Facilities connection details ITS Application and Facilities is configured with - Application username - Application password - application type - (Optional) TLS certificate for the Facilities
Trigger	ITS Application connects with the Facilities TCP port
ITS Application functions	When the connection requires TLS, the ITS-A checks the authenticity of the Facilities as part of the TLS negotiation. ITS-A registers with the Facilities using the *Register* method - Passes a RegistrationRequest object Waits for RegistrationResponse object *(Optional) ITS-A provides meta-data relevant for the Facilities* After connection success, - Stores session identifier - Executes the connection health use-case (see 8.4).
Facilities functions	Waits for connection requests from ITS Applications. When a TCP connection is initiated AND the connection must be secured with TLS: - Initiates the TLS negotiation - manages the TLS session creation. Waits for *Registration* request by the ITS Application. Checks Registration of the ITS Application against configured information according to decisions in Table 1 and Table 2 When successful registration - Sets Application session state = Connected - Creates a session identifier - Sends *RegistrationResponse* - Starts connection health use-case (see 8.4)
Post-conditions	*Application session state= Connected*
Exceptions	Facilities rejects ITS Application provided credentials and/or type - Facilities provides failure in response to registration request - Facilities terminates session

	The Application username is already used by an active session - Reject connection attempt *Note: In case a lingering connection is present, the keep alive use-case will remove the dead application.*
End result	ITS-A has created a session and can start to access the FI Objects.

8.2 Break connection with the Facilities

Name	**Break connection with the Facilities**
Description / context	An ITS Application has a session with the Facilities, it needs to terminate the session.
Actor	ITS Application
Goal	The ITS Application is deregistered and disconnected from the Facilities.
Pre-condition(s)	Application session state = Connected
Trigger	ITS Application internal logic - Sends a Deregister request
ITS Application functions	Waits for response from the Facilities Response = OK - Terminates TLS and TCP sessions OR Response = ERROR - Terminates TLS and TCP sessions - Logs error
Facilities functions	Received DeregisterRequest - Executes decisions in Table 2
Post-conditions	*Application session state = Disconnected*
Exceptions	ITS-A receives no response - Terminates TLS and TCP sessions - Logs error
End result	ITS Application has no session with the Facilities

8.3 Revoke ITS Application authorisation

Follows decision table for the Connected state, see Table 2

8.4 Check connection health

Follows decision table for the Connected state, see Table 2

Exception handling

This chapter focuses on exceptions which can occur and describes how ITS-A and/or Facilities shall detect the exception and respond to it. This chapter does not address exceptions caused by a specific protocol implementation, but addresses implementation-independent exceptions only.

9

9.1 Network

ID	Title	Description
1	IP network problems	Both the Facilities and ITS-A shall detect problems in the network connection and disconnect the connection if a network problem is detected. ITSA shall take the initiative to re-connect. Examples of TCP/IP network problems: - the connection is lost; - a read or write operation on the TCP socket reports an error; - data is delayed; - One peer has disconnected but the other peer assumes the connection is still alive.
2	Message bursts	Both the Facilities and an ITS-A may send and receive a burst of messages. The sending entity is responsible for sending messages in the proper order. A time tick is sent with each message from the Facilities and ITS-A, this tick can be used to handle timing.
3	Multiple sockets	A network host may host more than one ITS-A, giving them all the same source IP address. The Facilities X-FI implementation shall - not close an existing socket when a peer tries to create a new connection with an IP address of an already connected peer - allow minimal 10 concurrent TCP-session in total
4	Socket error	If the Facilities detects a socket error for an established session it shall immediately deregister an ITS-A when the TCP socket of this ITS-A is closed.

9.2 Session

ID	Title	Description
1	Registration by already registered Application	The Facilities shall have one and only one session with an ITS-A. It may be possible that a previous session is still seen as active by the Facilities while an ITS-A tries to reconnect the session after a failure. The Facilities shall: - Accept only one session per Application username. - not close an existing session when a peer tries to register a new session with the same Application username - Report rejection to the peer trying to register the new session (Response ProtocolErrorCode = NotAuthorised)

		- Close socket connection with this peer The ITS-A shall: Implement the back off algorithm as described in section 5.9 when it is refused connection (i.e. the time between registration attempts shall increase as the number of failures increases)
2	Login with incorrect credentials	A peer may provide incorrect credentials, i.e. an unknown Application username or incorrect password not matching the Application known to the Facilities . The Facilties shall not allow sessions with peers providing incorrect credentials. The Facilities shall: - Provide feedback on the incorrect login with the NotAuthorised ProtocolErrorCode. - Close socket connection with this peer - Add messages to security log / alarm
3	Alive check fails	When alive check fails, the network or processing has failed to recover within the expected time. Both the ITS-A and the FI monitors the alive objects from the peer and regards the session as lost. The Facilities shall: - reset the session states - close the sockets - Log error situation The ITS-A shall - close the sockets - re-establish the session if needed. - Log error situation Back off algorithm: The ITS-A tries to initiate a new session following the back off procedure (see 5.9).
4	Facilities restart (soft)	During active sessions it is possible that the Facilities needs to restart the interface in a soft way. All existing sessions must be disconnected. The Facilities shall: - notify the ITS-A's of the imminent restart generating a SessionEvent with FacilitiesStopping code - deregister all ITS-A's - discard (silently) any new registration attempts The ITS-A shall: - handle proper deregistration - try to re-connect to the Facilities and follow normal back off mechanism as defined in section 5.9
5	No Registration	A peer may connect to the Facilities, but fail to provide a Registration request. The Facilities shall - Wait for the ITS-A Alive timeout defined in 5.7 - Terminate the connection

6	Registration within active session	A peer may provide a Registration request within an active (Connected) session The Facilities shall: - Deregister the active (Connected) session
7	Deregistration from an ITS-A that is not registered.	The Facilities shall - Reply with ProtocolErrorCode(0)

9.3 Protocol compatibility

ID	Title	Description
1	Incompatible protocol	An application (not supporting the X-FI) connects to the TCP socket and starts communicating. The Facilities shall: - Parse the incoming data stream, taking into account that the data may not be coming for a peer supporting the X-FI protocol. - Not crash as result of another application opening and using the TCP port. - Disconnect the connection if the parsing of the incoming data fails. - Disconnect the connection after an idle timeout.
2	Application using older (supported) protocol version	Updates to the X-FI interface specifications (this document and the TLC-FI ([Ref 4]) and RIS-FI specifications ([Ref 5])) shall take compatibility into account, allowing an ITS-A to communicate with a Facilities implementing a newer version of the X-FI.
3	Application using older (un-supported) protocol version	Updates to the X-FI interface specifications (this document and the TLC-FI ([Ref 4]) and RIS-FI specifications ([Ref 5])) shall explicitly state a version incompatibility. The Facilities shall - detect this situation and report this explicitly back to the ITS-A The ITS-A shall: - stop communicating with the Facilities
4	Application using newer protocol version	The Facilities uses an older version of the X-FI protocol than the ITS-A. ITS-A shall detect this situation and interface with this Facilities correctly based on the functionality provided by the X-FI. The Facilities assumes that ITS-A will deal with this issue.

9.4 Timing

ID	Title	Description
1	Time-tick inconsistency	The time-tick of the Facilities may be slightly faster or slower than the time-tick of ITS-A. Both ITS-A and the Facilities shall take into account that:

		- Messages from the other peer are asynchronous. - The slower peer may receive, every once in a while, multiple sets of messages within the same system tick. - The faster peer may receive, every once in a while, no messages during a system tick.
2	Time-tick overflow	The ticks is an ever increasing value which identifies the delta time between updates. The value of the tick overflows approximately every 49 days. Both peers shall handle an overflow of the tick value so that it is possible to explicitly determine the elapsed tick time between two consecutive ticks.

9.5 Messages

ID	Title	Description
1	Unknown methods	A peer may receive a not supported (undefined) method; i.e. a method which is not implemented in the peer or the other peer uses a newer version of the protocol with more functionality. When a response is expected, the peer shall send a reply message containing a JSON error object with error code -32601 (Method not found) as defined in section 11.
2	Unknown object types	A peer may receive a not supported (undefined) object type. The receiver shall: - Reject the message - Discard the object(s) updated in this message - When part of a request: Send an error code UnknownObjectType - When notification: Log error - Close connection
3	Unknown attributes	A peer may receive a not supported (undefined) attribute. The peer shall ignore the unknown attributes and continue processing the remaining attributes.
4	Invalid attribute value types	A peer may receive a known attribute of an incorrect type. Each attribute is of a specific type, String, Number, etc. The peer shall: - Reject the attribute - Discard the object(s) updated in this message - When part of a request: Send an error code InvalidAttributeType - When notification: Log error - Close connection
5	Invalid attribute values	A peer may receive an attribute with an incorrect value. E.g. unknown enumeration, larger than maximum value etc. The peer shall: - Reject the attribute - Discard the object(s) updated in this message - When part of a request: Send an error code InvalidAttributeValue - When notification: Log error - Close connection
6	Invalid Object reference	A peer may receive an unknown object reference. The peer shall: - Reject the attribute

		- Discard the object(s) updated in this message - When part of a request: Send an error code InvalidObjectReference - When notification: Log error - Close connection
7	Invalid JSON message	A peer sends an invalid JSON encoded message. An invalid encoded JSON message points to incorrect implementation of the peer. The receiver shall: - be able to detect such a situation - update diagnostics - stop processing messages from this peer - deregister from the Facilities (ITS-A) - disconnect session
8	Buffer overflow	A peer sends a large valid JSON encoded message. As result the message doesn't fit the number of bytes buffered by the receiving peer. A peer shall: - be able to detect such a situation - discard the complete message - stop processing messages from this peer - deregister from the Facilities (ITS-A) - disconnect session

IRS Requirement tracing

10.1 TLC-FI

This section provides a statement of the compliance of this IDD with the *Beter Benutten Vervolg, project iVRI, Deliverable G2, IRS TLC Facilities Interface v1.2, jan 2016* (see [Ref
10 2])
The following statements are made for compliance with a requirement:

- C = Compliant
- P = Partially compliant
- N = Not compliant

A list of sections in this document in which the requirement is supported is listed and a comment describing the compliance statement.

Note that the list provides all requirements of the IRS, while a number of requirements is supported by the accompanying TLC-FI IDD, *Beter Benutten Vervolg, project iVRI – fase 2, Deliverable 1a IDD TLC Facilities Interface v1.1, dec 2016* (see [Ref 4]). In such cases, the sections column (also) refers to this document.

Requirement	Compliance	Sections	Comments
IRS-TLCFI-TIME-001	C	5.3	
IRS-TLCFI-PROT-001	C	4.2	
IRS-TLCFI-PROT-002	C	4.2	
IRS-TLCFI-PROT-003	C	4.2, 4.3	
IRS-TLCFI-COM-001	C	4.6	
IRS-TLCFI-COM-002	P	4.6	Updates on state changes, no periodic updates
IRS-TLCFI-COM-003	C	5.4.3	
IRS-TLCFI-COM-004	N		No periodic updates supported
IRS-TLCFI-COM-005	P	See [Ref 4]	Filtering based on type and subset of object ids
IRS-TLCFI-COM-006	N	-	No pre-defined filters supported
IRS-TLCFI-REG-001	P	6.3, 7.1, 8.1	No priority levels
IRS-TLCFI-REG-002	C	6.3	
IRS-TLCFI-REG-003	N	-	No priority levels
IRS-TLCFI-REG-004	C	6.4, 7.2, 8.2	
IRS-TLCFI-REG-005	C	9.2	
IRS-TLCFI-REG-006	C	5.4, 6.6, 7.3	
IRS-TLCFI-REG-007	C	5.4.3, 5.5, 9.2	
IRS-TLCFI-ICA-REG-001	C	See [Ref 4]	
IRS-TLCFI-ICA-AD-001	C	See [Ref 4]	
IRS-TLCFI-ICA-AD-002	C	See [Ref 4]	

IRS-TLCFI-ICA-AD-003	C	See [Ref 4]	
IRS-TLCFI-ICA-AD-004	C	See [Ref 4]	
IRS-TLCFI-ICA-AD-005	C	See [Ref 4]	
IRS-TLCFI-ICA-AD-006	N		An ITS-CLA controls one intersection. Multiple sessions are needed.
IRS-TLCFI-ICA-AD-007	C	See [Ref 4]	
IRS-TLCFI-TIF-OD-001	P	See [Ref 4]	No pre-defined filters
IRS-TLCFI-TIF-OD-002	C	6, See [Ref 4]	
IRS-TLCFI-TIF-OD-003	C	See [Ref 4]	
IRS-TLCFI-TIF-OD-004	C	6.1, 6.2, See [Ref 4]	
IRS-TLCFI-TIF-OD-005	P	See [Ref 4]	No addable / deletable objects
IRS-TLCFI-TIF-OD-006	C	5.1, 6.2, See [Ref 4]	
IRS-TLCFI-TIF-OM-001	N	-	No addable / deletable objects
IRS-TLCFI-TIF-OM-002	C	6.2, See [Ref 4]	
IRS-TLCFI-TIF-OM-003	**C**	6.2, See [Ref 4]	
IRS-TLCFI-TIF-OM-004	N		No addable / deletable objects
IRS-TLCFI-TIF-OT-001	C	See [Ref 4]	
IRS-TLCFI-TIF-OT-002	P	See [Ref 4]	Object doesn't contain: - Fault state - Special function variables - Active ITS-CLA (security concern)
IRS-TLCFI-TIF-OT-003	P	See [Ref 4]	The ITS-CLA is not informed of a higher priority request
IRS-TLCFI-TIF-OT-004	P	See [Ref 4]	Object doesn't contain: - Internal signal group state (including format) - Reason for deviation from external state - Fault state (deadlock, lamps) - Special function variables and status Meta: - Type (vehicle, bicycle, pedestrian, tram) - Related detectors
IRS-TLCFI-TIF-OT-005	C	See [Ref 4]	
IRS-TLCFI-TIF-OT-006	C	See [Ref 4]	
IRS-TLCFI-TIF-OT-007	P	See [Ref 4]	Object doesn't contain: Meta: Type
IRS-TLCFI-TIF-OT-008	C	See [Ref 4]	

IRS-TLCFI-TIF-OT-009	C	See [Ref 4]	
IRS-TLCFI-TIF-OT-010	C	See [Ref 4]	
IRS-TLCFI-TIF-OT-011	C	See [Ref 4]	
IRS-TLCFI-TIF-OT-012	P	6.3, See [Ref 4]	Objects don't provide: - Intersection topology data - ITS - Application status (security concern) - TLC Capability classes
IRS-TLCFI-QA-PERF-001	C	NA	
IRS-TLCFI-QA-PERF-002	C	9.1	No limit imposed in technology, objects or methods
IRS-TLCFI-QA-PERF-003	C	NA	No limit imposed in technology, objects or methods
IRS-TLCFI-QA-PERF-004	C	NA	No limit imposed in technology, objects or methods
IRS-TLCFI-QA-PERF-005	C	NA	No limit imposed in technology, objects or methods
IRS-TLCFI-QA-PERF-006	C	NA	No limit imposed in technology, objects or methods
IRS-TLCFI-QA-PERF-007	C	NA	No limit imposed in technology, objects or methods
IRS-TLCFI-QA-AVAIL-001	C	See [Ref 4]	
IRS-TLCFI-QA-AVAIL-002	N	-	No quality information is provided by an ITS-CLA
IRS-TLCFI-QA-AVAIL-003	C	5.2, See [Ref 4]	
IRS-TLCFI-QA-AVAIL-004	N	-	No reliance on UTC for the object exchange
IRS-TLCFI-QA-EVO-001	C	9.3	

Appendix: JSON-RPC 2.0 Specification

Below is a copy of the JSON-RPC 2.0 specification of http://www.jsonrpc.org/specification.

Origin Date:
11 2010-03-26 (based on the 2009-05-24 version)
Updated:
2013-01-04
Author:
JSON-RPC Working Group <json-rpc@googlegroups.com>

1 Overview

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

It is designed to be simple!

2 Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Since JSON-RPC utilizes JSON, it has the same type system (see http://www.json.org or RFC 4627). JSON can represent four primitive types (Strings, Numbers, Booleans, and Null) and two structured types (Objects and Arrays). The term "Primitive" in this specification references any of those four primitive JSON types. The term "Structured" references either of the structured JSON types. Whenever this document refers to any JSON type, the first letter is always capitalized: Object, Array, String, Number, Boolean, Null. True and False are also capitalized.

All member names exchanged between the Client and the Server that are considered for matching of any kind should be considered to be case-sensitive. The terms function, method, and procedure can be assumed to be interchangeable.

The Client is defined as the origin of Request objects and the handler of Response objects. The Server is defined as the origin of Response objects and the handler of Request objects.

One implementation of this specification could easily fill both of those roles, even at the same time, to other different clients or the same client. This specification does not address that layer of complexity.

3 Compatibility

JSON-RPC 2.0 Request objects and Response objects may not work with existing JSON-RPC 1.0 clients or servers. However, it is easy to distinguish between the two versions as 2.0 always has a member named "jsonrpc" with a String value of "2.0" whereas 1.0 does not. Most 2.0 implementations should consider trying to handle 1.0 objects, even if not the peer-to-peer and class hinting aspects of 1.0.

4 Request object
A rpc call is represented by sending a Request object to a Server. The Request object has the following members:

jsonrpc
: A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".

method
: A String containing the name of the method to be invoked. Method names that begin with the word rpc followed by a period character (U+002E or ASCII 46) are reserved for rpc-internal methods and extensions and MUST NOT be used for anything else.

params
: A Structured value that holds the parameter values to be used during the invocation of the method. This member MAY be omitted.

id
: An identifier established by the Client that MUST contain a String, Number, or NULL value if included. If it is not included it is assumed to be a notification. The value SHOULD normally not be Null [1] and Numbers SHOULD NOT contain fractional parts [2]

The Server MUST reply with the same value in the Response object if included. This member is used to correlate the context between the two objects.

[1] The use of Null as a value for the id member in a Request object is discouraged, because this specification uses a value of Null for Responses with an unknown id. Also, because JSON-RPC 1.0 uses an id value of Null for Notifications this could cause confusion in handling.

[2] Fractional parts may be problematic, since many decimal fractions cannot be represented exactly as binary fractions.

4.1 Notification
A Notification is a Request object without an "id" member. A Request object that is a Notification signifies the Client's lack of interest in the corresponding Response object, and as such no Response object needs to be returned to the client. The Server MUST NOT reply to a Notification, including those that are within a batch request.

Notifications are not confirmable by definition, since they do not have a Response object to be returned. As such, the Client would not be aware of any errors (like e.g. "Invalid params","Internal error").

4.2 Parameter structures
If present, parameters for the rpc call MUST be provided as a Structured value. Either by-position through an Array or by-name through an Object.

- by-position: params MUST be an Array, containing the values in the Server expected order.
- by-name: params MUST be an Object, with member names that match the Server expected parameter names. The absence of expected names MAY result in an error being generated. The names MUST match exactly, including case, to the method's expected parameters.

5 Response object

When a rpc call is made, the Server MUST reply with a Response, except for in the case of Notifications. The Response is expressed as a single JSON Object, with the following members:

jsonrpc
A String specifying the version of the JSON-RPC protocol. MUST be exactly "2.0".
result
This member is REQUIRED on success.
This member MUST NOT exist if there was an error invoking the method.
The value of this member is determined by the method invoked on the Server.
error
This member is REQUIRED on error.
This member MUST NOT exist if there was no error triggered during invocation.
The value for this member MUST be an Object as defined in section 5.1.
id
This member is REQUIRED.
It MUST be the same as the value of the id member in the Request Object.
If there was an error in detecting the id in the Request object (e.g. Parse error/Invalid Request), it MUST be Null.
Either the result member or error member MUST be included, but both members MUST NOT be included.

5.1 Error object
When a rpc call encounters an error, the Response Object MUST contain the error member with a value that is a Object with the following members:
code
A Number that indicates the error type that occurred.
This MUST be an integer.
message
A String providing a short description of the error.
The message SHOULD be limited to a concise single sentence.
data
A Primitive or Structured value that contains additional information about the error.
This may be omitted.
The value of this member is defined by the Server (e.g. detailed error information, nested errors etc.).
The error codes from and including -32768 to -32000 are reserved for pre-defined errors. Any code within this range, but not defined explicitly below is reserved for future use. The error codes are nearly the same as those suggested for XML-RPC at the following url: http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php

code	**message**	**meaning**
-32700	Parse error	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.
-32600	Invalid Request	The JSON sent is not a valid Request object.
-32601	Method not found	The method does not exist / is not available.
-32602	Invalid params	Invalid method parameter(s).
-32603	Internal error	Internal JSON-RPC error.
-32000 to -32099	Server error	Reserved for implementation-defined server-errors.

The remainder of the space is available for application defined errors.

6 Batch

To send several Request objects at the same time, the Client MAY send an Array filled with Request objects.

The Server should respond with an Array containing the corresponding Response objects, after all of the batch Request objects have been processed. A Response object SHOULD exist for each Request object, except that there SHOULD NOT be any Response objects for notifications. The Server MAY process a batch rpc call as a set of concurrent tasks, processing them in any order and with any width of parallelism.

The Response objects being returned from a batch call MAY be returned in any order within the Array. The Client SHOULD match contexts between the set of Request objects and the resulting set of Response objects based on the id member within each Object.

If the batch rpc call itself fails to be recognized as an valid JSON or as an Array with at least one value, the response from the Server MUST be a single Response object. If there are no Response objects contained within the Response array as it is to be sent to the client, the server MUST NOT return an empty Array and should return nothing at all.

7 Examples

Syntax:

```
--> data sent to Server
<-- data sent to Client
```

rpc call with positional parameters:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}

--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42], "id": 2}
<-- {"jsonrpc": "2.0", "result": -19, "id": 2}
```

rpc call with named parameters:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23,
"minuend": 42}, "id": 3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42,
"subtrahend": 23}, "id": 4}
<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
```

a Notification:

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
--> {"jsonrpc": "2.0", "method": "foobar"}
```

rpc call of non-existent method:

```
--> {"jsonrpc": "2.0", "method": "foobar", "id": "1"}
<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not
found"}, "id": "1"}
```

rpc call with invalid JSON:

```
--> {"jsonrpc": "2.0", "method": "foobar, "params": "bar", "baz]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse
error"}, "id": null}
```

rpc call with invalid Request object:

```
--> {"jsonrpc": "2.0", "method": 1, "params": "bar"}
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null}
```

rpc call Batch, invalid JSON:

```
--> [
  {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
  {"jsonrpc": "2.0", "method"
]
<-- {"jsonrpc": "2.0", "error": {"code": -32700, "message": "Parse
error"}, "id": null}
```

rpc call with an empty Array:

```
--> []
<-- {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null}
```

rpc call with an invalid Batch (but not empty):

```
--> [1]
<-- [
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null}
]
```

rpc call with invalid Batch:

```
--> [1,2,3]
<-- [
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null},
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null},
  {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null}
]
```

rpc call Batch:

```
--> [
        {"jsonrpc": "2.0", "method": "sum", "params": [1,2,4], "id": "1"},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]},
        {"jsonrpc": "2.0", "method": "subtract", "params": [42,23], "id":
"2"},
        {"foo": "boo"},
        {"jsonrpc": "2.0", "method": "foo.get", "params": {"name":
"myself"}, "id": "5"},
        {"jsonrpc": "2.0", "method": "get_data", "id": "9"}
    ]
<-- [
```

```
        {"jsonrpc": "2.0", "result": 7, "id": "1"},
        {"jsonrpc": "2.0", "result": 19, "id": "2"},
        {"jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid
Request"}, "id": null},
        {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method
not found"}, "id": "5"},
        {"jsonrpc": "2.0", "result": ["hello", 5], "id": "9"}
    ]
```

rpc call Batch (all notifications):

```
--> [
        {"jsonrpc": "2.0", "method": "notify_sum", "params": [1,2,4]},
        {"jsonrpc": "2.0", "method": "notify_hello", "params": [7]}
    ]
<-- //Nothing is returned for all notification batches
```

8 Extensions

Method names that begin with rpc. are reserved for system extensions, and MUST NOT be used for anything else. Each system extension is defined in a related specification. All system extensions are OPTIONAL.
