

DIN EN 15531-2

ICS 35.240.60

Ersatz für
DIN EN 15531-2:2015-12

**Öffentlicher Verkehr –
Dienstschnittstelle für Echtzeitinformationen bezogen auf Operationen
im öffentlichen Verkehr –
Teil 2: Kommunikationsinfrastruktur;
Englische Fassung EN 15531-2:2022**

Public transport –
Service interface for real-time information relating to public transport operations –
Part 2: Communications infrastructure;
English version EN 15531-2:2022

Transport public –
Interface de service pour les informations en temps réel relatives aux opérations de transport
public –
Partie 2: Communications;
Version anglaise EN 15531-2:2022

Gesamtumfang 162 Seiten

DIN-Normenausschuss Automobiltechnik (NAAutomobil)

Nationales Vorwort

Dieses Dokument (EN 15531-2:2022) wurde vom Technischen Komitee CEN/TC 278 „Intelligente Verkehrssysteme“, dessen Sekretariat von NEN (Niederlande) erarbeitet.

Das zuständige deutsche Normungsgremium ist der Gemeinschaftsarbeitsausschuss NA 052-00-71 GA „Gemeinschaftsarbeitsausschuss NAAutomobil/DKE: Intelligente Verkehrssysteme“ des DIN-Normenausschusses Automobiltechnik (NAAutomobil).

Dieses Dokument enthält unter Berücksichtigung des DIN-Präsidialbeschlusses 1/2004 nur die Englische Fassung von EN 15531-2:2022.

Aktuelle Informationen zu diesem Dokument können über die Internetseiten von DIN (www.din.de) durch eine Suche nach der Dokumentennummer aufgerufen werden.

Dienste für den öffentlichen Verkehr verlassen sich zunehmend auf Informationssysteme, welche zuverlässige, einfach handhabbare und überall zugängliche korrekte Information für Reisende liefern. Diese Art von Systemen decken folgende spezifischen Nutzungsprofile ab: Laden von Kalendern und Fahrplänen; Kontrolle von Fahrzeugbeständen; Ticket- und Rechnungsausstellung; aktuelle Informationen zu laufenden Diensten usw.

Dieser Teil der Normenreihe legt einen Rahmen fest, der Kommunikationen und Datenaustauschprotokolle für Organisationen spezifiziert, die den Austausch von Echtzeitinformationen bezogen auf den Betrieb im öffentlichen Verkehr wünschen.

DIN EN 15531 besteht unter dem allgemeinen Titel *Öffentlicher Verkehr — Dienstschnittstelle für Echtzeitinformationen bezogen auf Operationen im öffentlichen Verkehr* aus folgenden Teilen:

- Teil 1: Kontext und Grundstruktur
- Teil 2: Kommunikationsstruktur
- Teil 3: Funktionelle Dienstschnittstelle

DIN CEN/TS 15531 besteht unter dem allgemeinen Titel *Öffentlicher Verkehr — Dienstschnittstelle für Echtzeitinformationen bezogen auf Operationen im öffentlichen Verkehr* aus folgenden Teilen:

- Teil 4: Funktionale Dienst-Schnittstellen: Anlagenüberwachung
- Teil 5: Funktionelle Dienstschnittstellen: Situativer Austausch

Änderungen

Gegenüber DIN EN 15531-2:2015-12 wurden folgende Änderungen vorgenommen:

- a) Anpassungen basierend auf Implementierungen in SIRI (Version 2.1) durchgeführt;
- b) redaktionelle Anpassungen.

Frühere Ausgaben

DIN CEN/TS 15531-2 (DIN SPEC 70655): 2011-06
DIN EN 15531-2: 2015-12

Nationaler Anhang NA (informativ)

Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe in EN 15531-1.

ISO und IEC stellen terminologische Datenbanken für die Verwendung in der Normung unter den folgenden Adressen bereit:

- IEC Electropedia: verfügbar unter <http://www.electropedia.org/>
- ISO Online Browsing Platform: verfügbar unter <http://www.iso.org/obp>

— Leerseite —

English Version

Public transport - Service interface for real-time
information relating to public transport operations - Part
2: Communications infrastructure

Transport public - Interface de service pour les
informations en temps réel relatives aux opérations de
transport public - Partie 2 : Communications

Öffentlicher Verkehr - Dienstschnittstelle für
Echtzeitinformationen bezogen auf Operationen im
öffentlichen Verkehr - Teil 2:
Kommunikationsinfrastruktur

This European Standard was approved by CEN on 16 October 2022.

CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration. Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CEN member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

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



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

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Contents

Page

European Foreword.....	6
Introduction.....	8
1 Scope.....	10
2 Normative references.....	11
3 Terms and definitions	11
4 Symbols and abbreviations	11
5 Common communication aspects	11
5.1 Data Exchange Patterns of Interaction	11
5.1.1 Introduction.....	11
5.1.2 Request/Response Pattern	11
5.1.3 Publish/Subscribe Pattern	12
5.1.4 Publish/Subscribe with Broker Pattern.....	13
5.1.5 Request/Response – Compound Requests	14
5.1.6 Publish/Subscribe – Compound Subscriptions.....	15
5.2 Delivery Patterns	15
5.2.1 Introduction.....	15
5.2.2 Direct Delivery	15
5.2.3 Fetched Delivery	16
5.2.4 Data Horizon for Fetched Delivery.....	17
5.2.5 Get Current Message	18
5.2.6 Multipart Despatch of a Delivery	18
5.2.7 Multipart Despatch of a Fetched Delivery – MoreData	19
5.3 Mediation Behaviour	20
5.3.1 Introduction.....	20
5.3.2 Mediation Behaviour – Maintaining Subscription Last Updated State	20
5.3.3 Mediation Behaviour – Subscription Filters	23
5.4 Recovery Considerations for Publish Subscribe	25
5.4.1 Introduction.....	25
5.4.2 Check Status – Polling.....	25
5.4.3 Heartbeat – Pinging.....	25
5.4.4 Degrees of Failure	26
5.4.5 Detecting a Failure of the Producer	26
5.4.6 Detecting a Failure of the Consumer.....	28
5.5 Recovery Considerations for Direct Delivery.....	29
5.6 Request Parameters and Interactions	29
5.7 Error Conditions for Requests	32
5.8 Versioning.....	34
5.8.1 Introduction.....	34
5.8.2 The Overall SIRS Framework Version Level	34
5.8.3 The SIRS Functional Service Type Version Level.....	34
5.9 Access Controls: Security and Authentication.....	34
5.9.1 Introduction.....	34
5.9.2 System Mechanisms External to SIRS Messages.....	35
5.10 Service Discovery	36
5.10.1 Introduction.....	36
5.10.2 Discovery of Servers that Support SIRS Services	36
5.10.3 Discovery of the Capabilities of a SIRS Server.....	36
5.10.4 Discovery of the Coverage of a Given SIRS Functional Service	36

5.11	Capability Matrix	37
5.11.1	Introduction	37
5.11.2	SIRI General Capabilities	38
6	Request/Response	39
6.1	Making a Direct Request	39
6.1.1	Introduction	39
6.1.2	ServiceRequest Message — Element	39
6.1.3	The ServiceRequestContext — Element	41
6.1.4	Common Properties of ServiceRequest Messages — Element	43
6.1.5	ServiceRequest — Example	44
6.1.6	Access Controls on a Request	44
6.2	Receiving a Data Delivery	45
6.2.1	Introduction	45
6.2.2	ServiceDelivery	46
7	Subscriptions	50
7.1	Setting up Subscriptions	50
7.1.1	Introduction	50
7.1.2	SubscriptionRequest	51
7.1.3	SubscriptionResponse	54
7.2	Subscription Validity	57
7.3	Terminating Subscriptions	57
7.3.1	Introduction	57
7.3.2	The TerminateSubscriptionRequest	57
7.3.3	The TerminateSubscriptionResponse	58
7.3.4	The SubscriptionTerminatedNotification (+SIRI 2.0)	60
8	Delivering data	62
8.1	Direct Delivery	62
8.1.1	Introduction	62
8.1.2	Acknowledging Receipt of Data (DataReceivedAcknowledgement)	62
8.2	Fetches Delivery	63
8.2.1	Introduction	63
8.2.2	Signalling Data Availability (DataReadyNotification / DataReadyResponse)	63
8.2.3	Polling Data (DataSupplyRequest/ServiceDelivery)	65
8.3	Delegated Delivery +SIRI 2.0	67
9	Recovery from system failure	67
9.1	Introduction	67
9.2	Recovery after Client Failure	67
9.3	Recovery after Server Failure	68
9.4	Reset after Interruption of Communication	68
9.5	Alive Handling	69
9.5.1	Introduction	69
9.5.2	CheckStatusRequest	69
9.5.3	CheckStatusResponse	70
9.5.4	HeartbeatNotification	71
9.6	Additional Failure modes for delegated delivery (+SIRI v2.0)	72
10	Transport of SIRI messages	73
10.1	Separation of Addressing from Transport Protocol	73
10.2	Logical Endpoint Addresses	73
10.2.1	Endpoint Addresses	73
10.2.2	Endpoint Address — Examples	74
10.3	Parallelism and Endpoint Addresses	75
10.4	Encoding of XML messages	76
10.4.1	Principles	76
10.4.2	Encoding of Errors in XML	76
10.4.3	Character Set	76
10.4.4	Schema Packages	76

10.4.5	Siri.XSD – Use of XML Choice	77
10.4.6	SiriSG.XSD – Use of XML Substitution groups	78
10.5	Use of SIRI with SOAP / WSDL	79
10.5.1	Introduction	79
10.5.2	Web Services	80
10.5.3	Use of SOAP	82
10.5.4	SIRI WSDL	82
10.5.5	SIRI WSDL structure	83
10.5.6	SIRI RPC WSDL	86
10.5.7	SIRI Document WSDL (+SIRI v2.0)	90
10.5.8	SIRI WSDL 2.0 (+SIRI v2.0)	91
10.5.9	SIRI WSDL Status	91
11	Capability Discovery Requests	91
11.1	General	91
11.2	Capability Request	91
11.3	Service Capability Discovery	92
11.3.1	Service Capability Discovery Request — Element	92
11.3.2	Service Capability Discovery Response — Element	93
11.3.3	Functional Service Capability Discovery Response — Element	94
11.3.4	Service Capability Response — Example	96
11.4	Functional Service Capability Permission Matrix	98
11.4.1	Introduction	98
11.4.2	OperatorPermissions — Element	99
11.4.3	LinePermissions — Element	99
11.4.4	ConnectionLinkPermissions — Element	99
11.4.5	StopMonitorPermissions — Element	100
11.4.6	VehicleMonitorPermissions — Element	100
11.4.7	InfoChannelPermissions — Element	101
12	SIRI for Simple Web Services – SIRI Lite (+SIRI v2.0)	101
12.1	Introduction	101
12.1.1	Existing Implementations	102
12.1.2	Using SIRI-LITE services in combination	102
12.1.3	Alternative Response Encoding	103
12.1.4	Lossless transforms	104
12.1.5	Simple transforms	104
12.2	Encoding of URL Requests	104
12.2.1	Complete Request Encoding in HTTP URL's	104
12.2.2	General format of SIRI Lite request URL	104
12.2.3	Endpoints and Service Identification	105
12.2.4	Encoding of Service Parameters on http request	105
12.2.5	Naming of Request Parameters with Hierarchy	106
12.2.6	Naming of Parameters with Plural Cardinality	106
12.2.7	Handling of invalid request combinations	106
12.2.8	Specifying the encoding of the Response	106
12.3	Examples	106
12.3.1	General	106
12.3.2	SIRI-SM Simple Stop Monitoring request to fetch stop departures – SIRI LITE Examples	106
12.3.3	SIRI-VM Simple Vehicle Monitoring request to fetch vehicle positions – SIRI Lite examples	110
12.3.4	SIRI-VM Complex Vehicle Monitoring to obtain journeys – SIRI Lite examples	113
12.3.5	SIRI-SM Stop Monitoring failed request with Exception – SIRI LITE examples	120
12.4	Mapping of SIRI XML to Alternative encodings	121
12.4.1	Use of syntactic features of alternative rendering formats	121
12.4.2	Mapping of SIRI data types to alternative encodings	122
12.5	Recommendations for the use of SIRI Simple Web Services	122
12.5.1	General	122
12.5.2	Services useful for device Passenger Information Services	122
12.5.3	Response filtering	122

12.5.4	Incorporation of reference data in responses.....	123
12.5.5	Multiple functional service deliveries in the same response	123
12.5.6	Support a choice of response encodings	123
12.5.7	Provide reporting identifiers	123
13	Common SIRI elements & Data Types	124
13.1	General.....	124
13.2	Introduction	125
13.3	Base Data Types.....	125
13.3.1	W3C Simple Types	125
13.3.2	SIRI Simple Types.....	126
13.3.3	NationalLanguageStringStructure — Element.....	127
13.4	Shared Elements & Structures.....	127
13.4.1	FramedVehicleJourneyRef — Element.....	127
13.4.2	Location — Element.....	128
13.4.3	Error — Element.....	128
13.4.4	JourneyRelation — Element (+SIRI 2.1)	129
13.4.5	Branding — Element (+SIRI 2.1)	133
13.4.6	Extension — Element.....	133
13.4.7	KeyList — Element (+SIRI 2.1)	134
13.4.8	TypesOfValue — Element (+SIRI 2.1)	134
13.4.9	Train Formation/Composition Model — Element (+SIRI 2.1)	135
13.5	Shared groups of elements	139
13.5.1	ServiceInfoGroup — Group	139
13.5.2	JourneyInfoGroup — Group	140
13.5.3	VehicleJourneyInfoGroup — Group	140
13.5.4	JourneyPatternInfoGroup — Group	142
13.5.5	DisruptionGroup — Group	143
13.5.6	JourneyProgressGroup — Group	146
13.6	OperationalBlockGroup — Group.....	150
13.7	OperationalInfoGroup — Group.....	150
13.8	TypeOfValueGroup — Group (+SIRI 2.1).....	150
13.9	JourneyRelationInfoGroup — Group (+SIRI 2.1).....	151
13.10	JourneyPartViewGroup — Group (+SIRI 2.1)	151
13.11	VehicleTypeGroup — Group (+SIRI 2.1).....	152
13.12	TrainFormationReferenceGroup — Group (+SIRI 2.1).....	153
13.13	QuayAssignmentGroup — Group (+SIRI 2.1).....	153
13.13.1	General.....	153
13.13.2	TypeOfNestedQuayEnumeration — Allowed Values	154
13.14	BoardingPositionAssignmentGroup — Group (+SIRI 2.1).....	155
13.15	FlexibleStopLocationGroup — Group (+SIRI 2.1)	156

European foreword

This document (EN 15531-2:2022) has been prepared by Technical Committee CEN/TC 278 “Intelligent transport systems”, the secretariat of which is held by NEN.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by May 2023, and conflicting national standards shall be withdrawn at the latest by May 2023.

This document supersedes EN 15531-2:2015.

SIRI (CEN/TS 15531-2:2006) has been a CEN Technical Specification since 2007 and a European normative standard since 2013 and has been widely used in Europe and elsewhere and proven its usefulness. This document proposes a revised version of SIRI as a European Standard, and is currently submitted to the Formal Vote. The proposed revisions are minor enhancements arising from experience of the deployment of SIRI in many live systems. This document also clarifies the relationship of SIRI to NeTEx, the CEN Technical Standard for the XML exchange of Public Transport Reference data based on the Transmodel CEN European Standard.

This document presents Part 2 of the European Standard known as “SIRI”. SIRI provides a framework for specifying communications and data exchange protocols for organisations wishing to exchange Real-time Information (RTI) relating to public transport operations.

The SIRI European Standard is presented in three parts:

The SIRI European Standard is presented in three parts:

- context and framework, including background, scope and role, normative references, terms and definitions, symbols and abbreviations, business context and use cases (Part 1),
- the mechanisms to be adopted for data exchange communications links (Part 2),
- data structures for a series of individual application interface modules PT, ET, ST, SM, VM, CT, CM, GM (Part 3).

Two additional parts define additional functional services as CEN Technical Specifications:

- additional data structures for additional application interface module FM (Part 4),
- additional data structures for additional application interface module SX (Part 5).

The XML schema can be downloaded from <https://github.com/SIRI-CEN/SIRI>, guidance on its use, example XML files, and case studies of national and local deployments is located at <http://siri-cen.eu/>.

It is recognised that SIRC is not complete as it stands, and from time to time will need to continue to be enhanced to add additional capabilities. It is therefore intended that a SIRC Management Group should continue to exist, at European level, based on the composition of SG7.

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

Any feedback and questions on this document should be directed to the users' national standards body. A complete listing of these bodies can be found on the CEN website.

According to the CEN-CENELEC Internal Regulations, the national standards organisations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Türkiye and the United Kingdom.

Introduction

Public transport services rely increasingly on information systems to ensure reliable, efficient operation and widely accessible, accurate passenger information. These systems are used for a range of specific purposes: setting schedules and timetables; managing vehicle fleets; issuing tickets and receipts; providing real-time information on service running, and so on.

This document specifies a Service Interface for Real-time Information (SIRI) about Public Transport. It is intended to be used to exchange information between servers containing real-time public transport vehicle or journey time data. These include the control centres of transport operators and information systems that utilise real-time vehicle information, for example, to deliver services such as travel information.

Well-defined, open interfaces have a crucial role in improving the economic and technical viability of Public Transport Information Systems of all kinds. Using standardised interfaces, systems can be implemented as discrete pluggable modules that can be chosen from a wide variety of suppliers in a competitive market, rather than as monolithic proprietary systems from a single supplier. Interfaces also allow the systematic automated testing of each functional module, vital for managing the complexity of increasing large and dynamic systems. Furthermore, individual functional modules can be replaced or evolved, without unexpected breakages of obscurely dependent function.

This European Standard will improve a number of features of public transport information and service management:

- Interoperability – the European Standard will facilitate interoperability between information processing systems of the transport operators by: (i) introducing common architectures for message exchange; (ii) introducing a modular set of compatible information services for real-time vehicle information; (iii) using common data models and schemas for the messages exchanged for each service; and (iv) introducing a consistent approach to data management.
- Improved operations management – the European Standard will assist in better vehicle management by (i) allowing the precise tracking of both local and roaming vehicles; (ii) providing data that can be used to improve performance, such as the measurement of schedule adherence; and (iii) allowing the distribution of schedule updates and other messages in real-time.
- Delivery of real-time information to end-users – the European Standard will assist the economic provision of improved data by: (i) enabling the gathering and exchange of real-time data between AVMS systems; (ii) providing standardised, well defined interfaces that can be used to deliver data to a wide variety of distribution channels. Version 2.0 of SIRI includes a new Simple Web Service designed to support the widespread, massively scalable use of mobile devices and web browsers and other applications to display public transport data directly to users.

Technical advantages include the following:

- Reusing a common communication layer for all the various technical services enables cost-effective implementations and makes the European Standard readily extensible in future.

History

Version 1.0 of SIRI was developed in 2004-2005 and submitted to vote, eventually passing through the CEN process to become an approved CEN Technical Specification in 2007. As well as the normative Version 1.0 XSD schema, successive informal working versions of the schema (v 1.1 – 1.4) were released to allow for fixes and to implement some very minor enhancements agreed by the working group. A WSDL version was also developed.

Version 2.0 of SIRI was developed in 2012 to coincide with making the SIRI standard a full CEN norm.

SIRI includes a Simple Web Services “SIRI-LITE” as an additional transport method and a WSDL document literal version and a WSDL2 version;

Version 2.1 of SIRI was developed in 2020/21 to address lessons from the now widespread implementation of SIRI.

The changes in SIRI version 2.1 include:

- remove the direct relationship with TPEG and other standards to enable support as the other standards change;
- support for new modes in line with TRANSMODEL and NeTEx;
- support for the Reason / Effect / Advice structure for disruptions in SIRI SX;
- increased granularity for occupancy data and Vehicle structures;
- improved subscription renewal options and filtering options;
- additional options and flexibility for STOP POINTS and relationships between journeys;
- migration of XSD to Github to improve access and change control processes.

Compatibility with previous versions

All changes in version 2.1 are intended to be fully backwards compatible, that is to say, existing documents that validate against earlier versions of the schema will also validate against the 2.1 schema without alteration (other than to schema version numbers), and version 2.1 documents that do not use new features will validate against earlier versions. Version 2.1 documents that use new features will not be backwards compatible.

1 Scope

SIRI uses a consistent set of general communication protocols to exchange information between client and server. The same pattern of message exchange may be used to implement different specific functional interfaces as sets of concrete message content types.

Two well-known specific patterns of client server interaction are used for data exchange in SIRI: *Request/Response* and *Publish/Subscribe*.

- *Request/Response* allows for the ad hoc exchange of data on demand from the client.
- *Publish/Subscribe* allows for the repeated asynchronous push of notifications and data to distribute events and Situations detected by a Real-time Service.

The use of the *Publish/Subscribe* pattern of interaction follows that described in the Publish-Subscribe Notification for Web Services (WS-PubSub) specification, and as far as possible, SIRI uses the same separation of concerns and common terminology for publish/subscribe concepts and interfaces as used in WS-PubSub. WS-PubSub breaks down the server part of the *Publish/Subscribe* pattern into a number of separate named roles and interfaces (for example, Subscriber, Publisher, Notification Producer, and Notification Consumer): in an actual SIRI implementation, certain of these distinct interfaces may be combined and provided by a single entity. Although SIRI is not currently implemented as a full WS-PubSub web service, the use of a WS-PubSub architecture makes this straightforward to do in future.

Publish/Subscribe will not normally be used to support large numbers of end user devices.

For the delivery of data in responses (to both requests and subscriptions), SIRI supports two common patterns of message exchange, as realised in existent national systems:

- A one step 'Direct Delivery', as per the classic client-server paradigm, and normal WS-PubSub publish subscribe usage; and;
- A two-step 'Fetched Delivery' which elaborates the delivery of messages into a sequence of successive messages pairs to first notify the client, and then to send the data when the client is ready. Fetched Delivery is a stateful pattern in its own right.
- Each delivery pattern allows different trade-offs for implementation efficiency to be made as appropriate for different target environments.
- A SIRI implementation may support either or both delivery methods; in order to make the most efficient use of the available computational and communication resources. The delivery method may either be preconfigured and static for a given implementation, or each request or subscription may indicate the delivery method required by the client dynamically as part of the request policy, and the server may refuse a request if it does not support that method, giving an appropriate error code.
- The Interaction patterns and the Delivery patterns are independent aspects of the SIRI protocol and may be used in any combination in different implementations.
- For a given SIRI Functional Service type (Connection Monitoring, Stop Monitoring etc.), the message payload content is the same regardless of whether information is exchanged with a *Request/Response* or *Publish/Subscribe* pattern, or whether it is returned by Direct or Fetched Delivery.

- The SIRI *Publish/Subscribe* Protocol prescribes particular *mediation* behaviour for reducing the number of notifications and the amount of network traffic arising from subscriptions.
- The mediation groups the various subscriptions from a subscriber into one or more Subscriber Channels, and is able to manage notifications and updates for the aggregate.
- Only partial updates to the data set since the last delivery for the subscription need to be sent.
- The SIRI Communication protocols are designed to fail gracefully. Considerations for resilience and recovery are covered below.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

EN 15531-1, *Public transport - Service interface for real-time information relating to public transport operations - Part 1: Context and framework*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in EN 15531-1 apply.

4 Symbols and abbreviations

For the purposes of this document, the symbols and abbreviations given in EN 15531-1 apply.

5 Common communication aspects

5.1 Data Exchange Patterns of Interaction

5.1.1 Introduction

There are two main patterns of interaction for Data Exchange in SIRI: *Request/Response* and *Publish/Subscribe*. The patterns are complementary, that is an implementation may support both, and implementers may choose the most efficient pattern according to the nature of their application.

NOTE Publish/Subscribe can emulate a Request/Response interaction by use of a short subscription. A partial SIRI implementation that supports only Request/Response is useful for connecting many types of Public Transport Information System applications to AVMS and other Producer System data.

5.1.2 Request/Response Pattern

The *Request/Response* interaction allows for the immediate fulfilment of one-off data supply requests made by a Requestor to a Service. Pairs of *Request/Response* patterns are also used for the interactions that make up other patterns, such as *Publish/Subscribe*.

In the *Request/Response* interaction used to get data, the Client sends a request message to a Server that offers the required SIRI Functional Service, and immediately receives a *Delivery* message in response (Figure 1). A Data Delivery may be made as a one-step *Direct Delivery*, or as a two-step *Fetches Delivery* (see later).

The Requestor shall give a unique reference to each request, which will be returned in the matching response.

The Requestor expresses its specific interests through Topic and Delivery Policy parameters on the specific SIRI Functional Service Requests. If the request cannot be satisfied an error condition is returned diagnosing the reason.

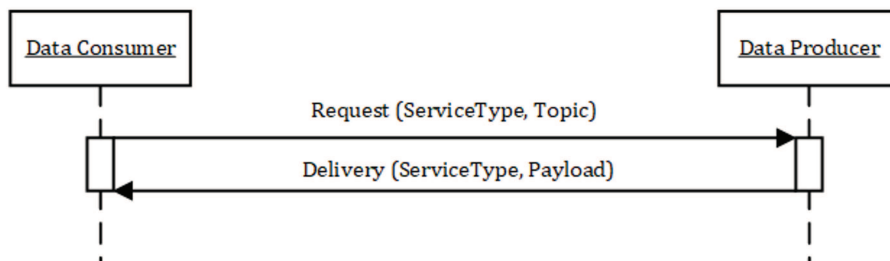


Figure 1 — Request / Response Interaction

Request/response allows for an efficient transmission of data on-demand from the Consumer and is extremely easy to implement using commodity internet software components.

5.1.3 Publish/Subscribe Pattern

The *Publish/Subscribe* interaction (see Figure 2) allows for the asynchronous detection of real-time events by a producer service, whose role is to generate and send notifications to one or more interested consumers.

In the *Publish/Subscribe* interaction, the Subscriber client sends a request message to the Notification Producer of a SIRI Functional Service to create a Subscription, which may or may not be granted. The Subscriber expresses its specific interests through Topic and Subscription Policy parameters, and receives an acknowledgement that this has been created, or an error condition.

Once a Subscription exists, the service, acting as the Notification Producer, uses it to determine when to send a notification to a consumer after a Situation, i.e. event is detected. The incoming event notification to be published is matched against the interests expressed by the Topic and other filter parameters of the Subscription and if satisfied, a notification message is sent to the Consumer. The actual Notification Message Delivery may be made either as a one-step *Direct Delivery* to a Notification Consumer, or as a two-step SIRI *Fetches Delivery*, with separate message pairs first to notify and then to deliver the payload.

In SIRI, the Subscriber and Consumer roles are normally implemented by the same client service, although they are logically separate. Every Consumer must know its Subscriber so that they can interact to handle recovery from service failures.

Subscriptions for different types of SIRI Functional Service are managed separately.

A Subscriber may add different Subscriptions at different times.

A Subscription Request includes an Initial Termination Time indicating the desired duration i.e. lease of the individual Subscription. The subscription will only be granted if this can be met, otherwise an error will be returned.

Subscriptions have a life span as specified by the Subscriber and will be terminated by the Notification Producer service when they reach their expiry time.

Subscribers may terminate their own existing Subscriptions before their predefined expiry time through a Subscription Manager. The Subscription Manager is subordinate to the Notification Producer, and in SIRI implementations, is normally provided by the same entity, although logically distinct. Each Subscription Manager knows its associated Notification Producer, and vice versa. Although the Notification Producer is the factory for creating new subscriptions, it does not manage them once created; rather this is done by the Subscription Manager. This design (i.e. the Notification Producer finds the Subscription Manager for the Subscriber, rather than the Subscription Manager finds the Notification Producer for the Subscriber) is required to conform to the WS-PubSub architecture. The WS-PubSub architecture allows for additional Subscription management functions to be added through the Subscription Manager for example renewal, pause/resume, or the dynamic tuning of subscription policies, but SIRI does not specify any of these at present. SIRI does however support a Terminate Subscription and a Terminate All Subscriptions function.

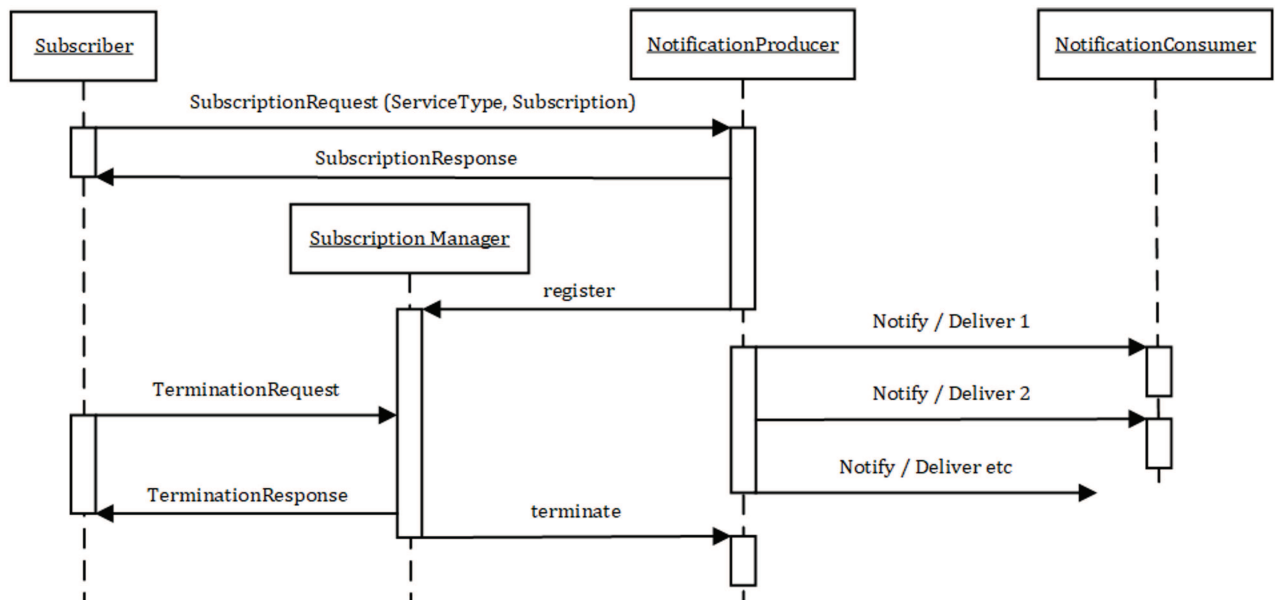


Figure 2 — Simple Publish/Subscribe Interaction

Subscriptions are a stateful resource: they need a unique identifier that can be used by Subscriber, Producer and Consumer to refer to the same subscription on different occasions. They will each hold their own representation of the subscription. In SIRI this identifier is issued by the Subscriber.

Publish/Subscribe allows for an efficient regular event driven exchange of updates to data. It requires a more elaborate implementation, with the holding of state by both participants and the dedication of computing resources to run the notification production.

5.1.4 Publish/Subscribe with Broker Pattern

The WS-PubSub architecture also allows for the logical separation of the concerns of Publishing and Notification Production, and in its fully articulated form, has a separate Publisher role that is a subordinate constituent of the Notification Producer service (see Figure 3). The Publisher produces

notifications of any significant situations, i.e. events. For a real-time service, the Publisher monitors the real-time data and if a change has occurred, it produces a notification. The Notification Producer then matches the Notification with the interests and policies expressed by Subscribers and despatches the notification delivery to the Notification Consumer indicated by the Subscription.

It is possible to have more than one Notification Producer sitting between the Publisher and the Consumer, either to carry out successive types of filtering and processing of the notifications, or for scalability. WS-PubSub distinguishes between *direct* notification – where the notification message from the Publisher is delivered unchanged, and *brokered* notification – where the Notification Producer filters and also possibly transforms the message. Both brokered (e.g. for SIRI Stop Monitoring) and unbrokered (e.g. for SIRI General Message) mediation occurs in different SIRI Functional Services.

The separation of concerns between Publisher and Notification producer is transparent to the Subscriber and Consumer, and so in SIRI is merely an implementation choice – which does not currently explicitly mandate any requirements for the interface between the Notification Producer and Publisher. Every Publisher knows its associated Notification Producer(s), and vice versa.

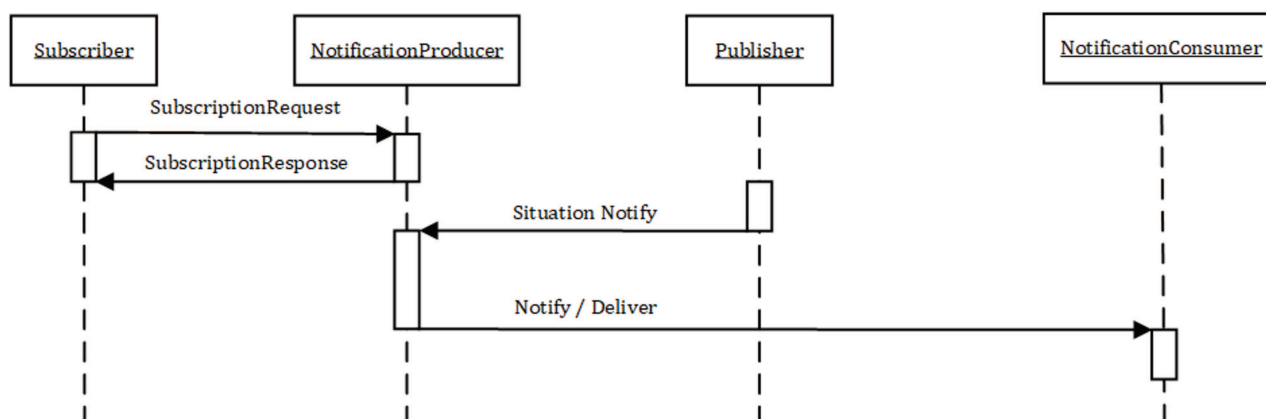


Figure 3 — Brokered Publish/Subscribe Interaction

Further Subscription and Subscriber filtering tasks, in particular the enforcement of SIRI Access controls, are implemented by the Notification Producer, not the Publisher.

5.1.5 Request/Response – Compound Requests

Multiple requests for a single SIRI Functional Service may be included in a single Data *Request/Response* interaction: each request may cover different topics and policies (Figure 4).

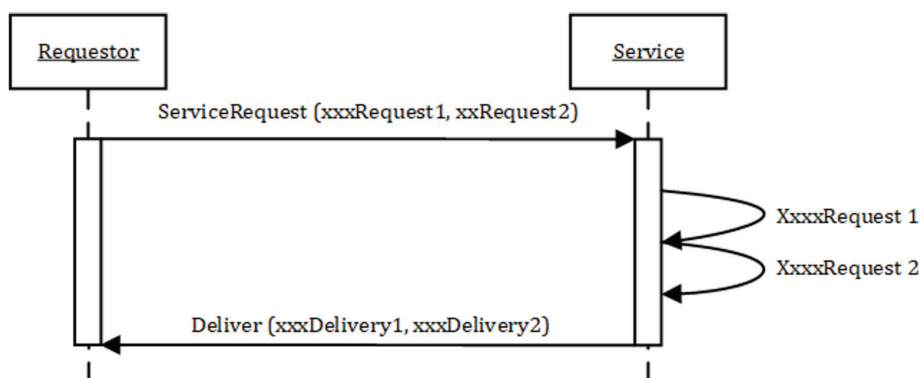


Figure 4 — Request/Response: Compound Requests

5.1.6 Publish/Subscribe – Compound Subscriptions

Multiple subscriptions to a single SIRS Functional Service may be included by a Subscriber in a single Subscription request: each subscription may cover different topics and policies (Figure 5). The handling of notifications and deliveries for compound subscriptions is discussed in the section on Mediation later below.

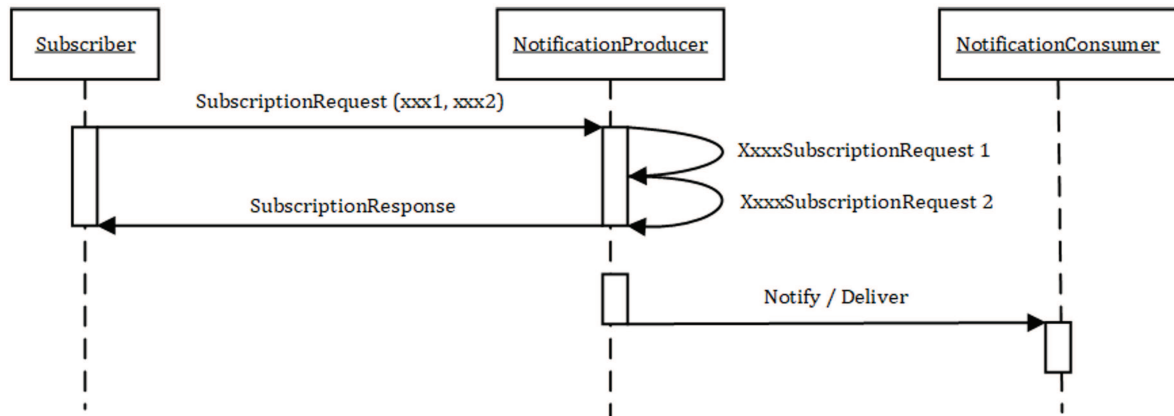


Figure 5 — Publish/Subscribe: Compound Subscriptions

5.2 Delivery Patterns

5.2.1 Introduction

Services return notifications and Situation content to the Consumer using Delivery messages. In real-time applications, it is important to be able to optimise systems to ensure rapid delivery, and SIRS supports two different message pattern variations for making a delivery, that in principle can be used interchangeably: these are; (i) *Direct Delivery*, and; (ii) *Fetches Delivery*.

The choice of delivery patterns may be pre-configured, or if the implementation supports both methods, be specified as a parameter on the request. For systems that support dynamic choice, if the SIRS implementation does not support the requested delivery method for a specific service type, an error message will be returned.

5.2.2 Direct Delivery

In *Direct Delivery*, the payload is sent as the content of a single message to the Consumer Client (Figure 6). For a *Request/Response* this will be the requestor. For a subscription this will be the Notification Consumer as indicated on the Subscription (i.e. the notification and the delivery are the same message.).

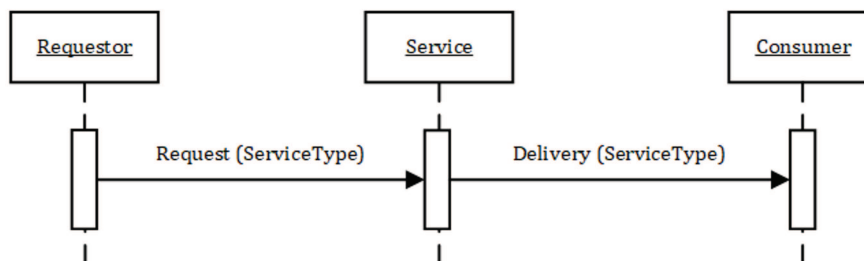


Figure 6 — One Step Direct Delivery

In *Direct Delivery*, the burden of holding and queuing messages is distributed to the client, with some advantages for scaling, as the central server needs neither retain data, nor allocate computation resource to service the additional data supply steps. The interaction is simpler, with fewer messages being exchanged, and a simpler mediation. However, the method does not allow the Consumer to optimise its own activities by separating its processing to detect the existence of an update from its processing to use the payload data. The full payload is always sent, even if it is not currently of interest to the client. *Direct Delivery* is appropriate for deployment with fast, reliable communications, and with adequate processing capability on the Consumer. It is especially efficient when most updates are relevant to the client and are used immediately.

5.2.3 Fetched Delivery

In *Fetched Delivery*, the delivery is done in two successive steps, separating the notification of an update from the delivery of the data payload (Figure 7). The steps are as follows:

- 1) The Producer sends a Data Ready Notification message to the Consumer.
- 2) The Consumer Acknowledges receipt with a Data Ready Response.
- 3) The Consumer sends a Data Supply Request to the Notification Producer.
- 4) The Notification Producer responds with a Data Supply Delivery.

The second read is allowed to be destructive, that is, the ability to recreate exactly the same delivery to that point is not guaranteed: the differential update may be deleted once it has been given to the Consumer.

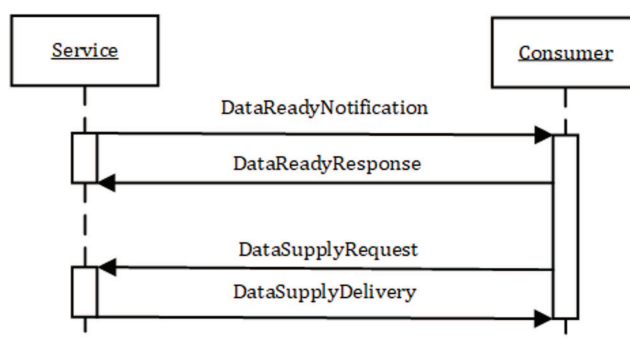


Figure 7 — Fetched Delivery

Fetched Delivery is a stateful pattern of interaction in its own right – requiring the ability of both parties to refer to the data update by a reference that persists for its currency. In this case the reference is

issued by the Producer. Whether the identifier needs to be exposed to the Consumer depends on the mediation model (see later): if all updates are aggregated through a single subscriber channel, then there is only one data set to fetch, and an explicit reference is not needed.

Fetches Delivery allows the Consumer to defer the sending of the full payload until it is ready to process it: if in the meantime the Situation has changed further, and a new notification message has arisen, only the latest update need be exchanged. This can give a more efficient use of bandwidth for applications that are bandwidth constrained. In addition, notifications are small messages that can typically be examined using less computational resource than a full delivery message containing a payload, so if the majority of updates are discarded (i.e. never fetched); the processing load on the Consumer may be less. Similarly, the storage requirement on the Consumer to queue small notification messages for *Fetches Delivery* is less than the storage requirement to queue larger payload messages for *Direct Delivery* (but larger on the Notification Producer to retain it until fetched). As a trade-off, there is additional computational and communication overhead required to conduct the extra interactions of fetched data supply messages, and also an additional latency to carry it out (in particular a fourfold communication overhead).

In practice, *Fetches Delivery* is used predominantly for *Publish/Subscribe*. Figure 8 shows the full sequence of interaction from subscription to delivery.

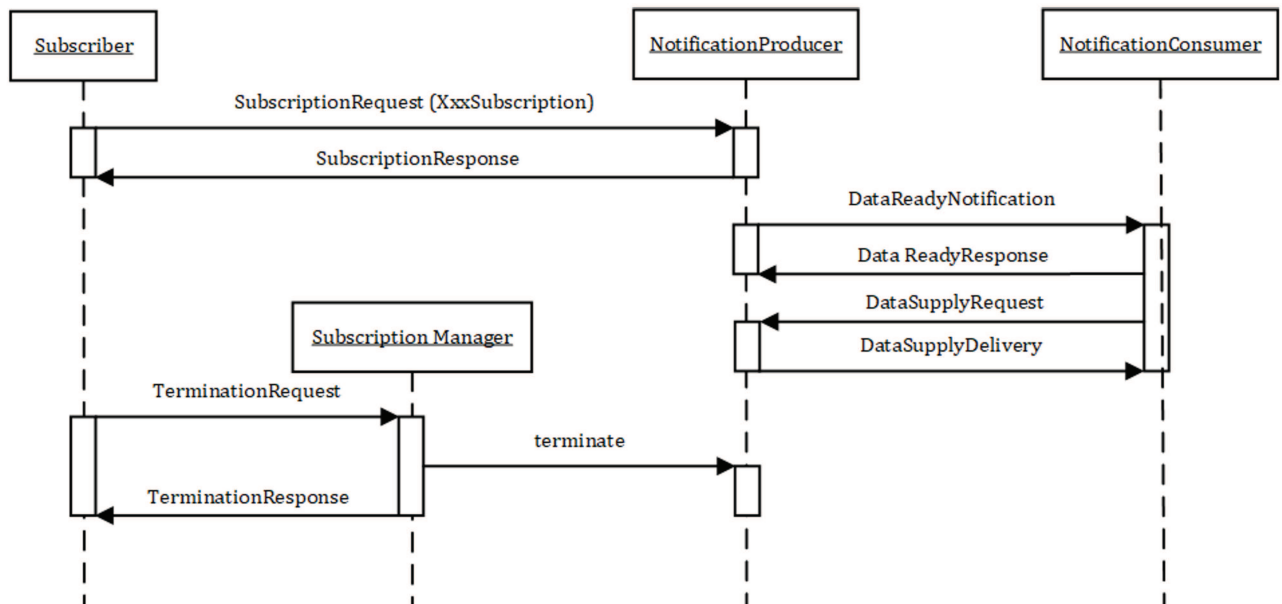


Figure 8 — Fetches Delivery for Publish/Subscribe

For completeness, we note that *Fetches Delivery* can also be used for the delivery of responses in a *Request/Response* interaction (Figure 9).

5.2.4 Data Horizon for Fetches Delivery

For *Fetches Delivery*, implementations may vary as to how long the data notified as ready will still be available to be fetched. At a minimum the most recent update must be available until it is stale, i.e. has reached the end of its currency time, or is superseded by another update. Some implementations may choose to keep previous updates available within a longer data horizon, for example the current day, and support historic access to a log of previous updates.

For further considerations as to the contents of fetched delivery, see discussion of Mediation Behaviour below.

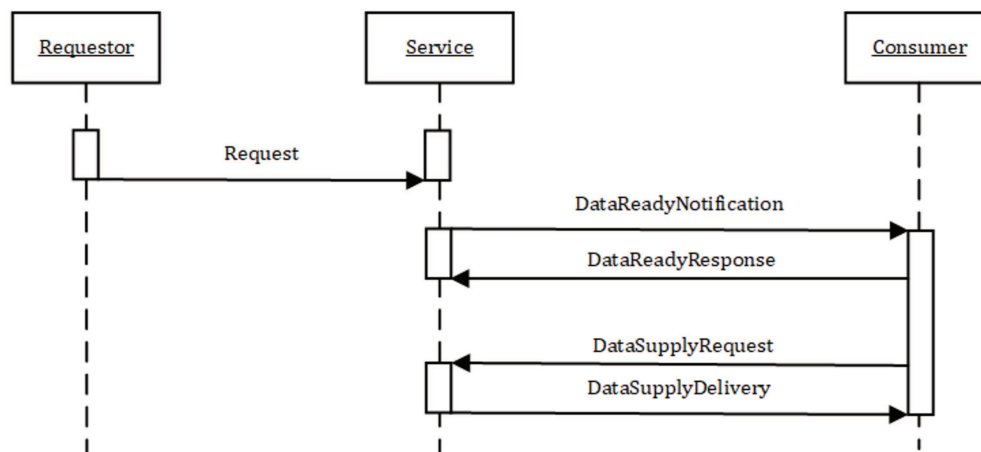


Figure 9 — Fetched Delivery for Request/Response

5.2.5 Get Current Message

The SIRI Data Supply Request may be used by itself without a previous notification to get the current data. This corresponds to the WS-PubSub capability of 'Get Current Message', which allows a Consumer to get the current messages published for a given subscription. In normal WS-PubSub usage this is a non-destructive read: the current data can be reread many times, provided the data is still relevant.

The SIRI Data Supply Request has a parameter which can be used to optimise usage: '*return latest*' i.e. whether to return only the most recent update for the subscription. (Return latest is in fact the default; the opposite, '*return all*' has to be explicitly specified on a request.) 'Return latest' cannot be repeated to obtain the same data, since once delivered, a new latest time will be held for the subscription.

In SIRI one of the mediations carried out by the Notification Producer (see later below) is to aggregate the individual subscriptions of a particular subscriber into a Subscription Channel. In effect, for each subscriber for each service, the Notification Producer keeps a Subscription channel or *Filter*: all subscriptions from the Subscriber are normally assigned to the Subscription Filter. This means that a SIRI Get Current Message request may need to indicate whether to return; (i) the latest data for a specific Subscription (the normal WS-PubSub interpretation); (ii) the latest data for a particular Subscriber Filter (i.e. group of subscriptions belonging to a single subscriber); or (iii) the latest data for a particular Subscriber (i.e. group of all the Subscriber Filters belonging to a particular Subscriber).

Support of '*return latest*' is a required SIRI Feature. Support for Get Current Message with '*return all*' is a required feature of WS-PubSub: a SIRI implementation will need to support it in order to be able to return a full data set to new subscribers who have just joined.

5.2.6 Multipart Despatch of a Delivery

If the amount of data to be delivered is large, some implementations allow the Producer to break the data supply step up into multiple messages, which the Consumer can then assemble into a single update (Figure 10). This has further implications for any recovery processes and is discussed in more detail later.

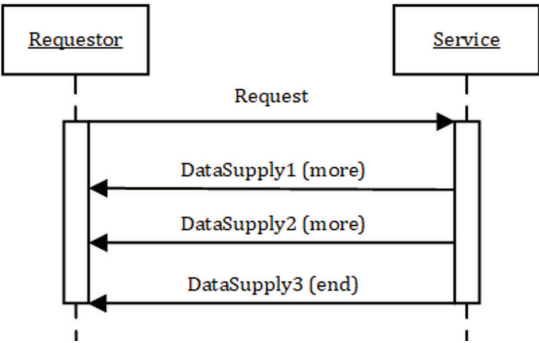


Figure 10 — Multipart Delivery

5.2.7 Multipart Despatch of a Fetched Delivery – MoreData

The data for a single subscription shall be completely included within a single Delivery. Data for a *Fetches Delivery* may only be split into separate messages if it is for different subscriptions for the same subscriber.

For multipart despatch, on the Service Delivery message the **MoreData** element indicates whether the content of a **DataSupplyResponse** contains all the updated data, or whether for implementation reasons, the transmission has been split into several sub-messages, requiring retrieval by the consumer with a series of chained data **DataSupplyRequests**; see Figure 11. Each Data Supply Delivery response message in the chain indicates that there is further data by means of a **MoreData** value of “true”; in the last data supply response message, the **MoreData** element is set to “false”.

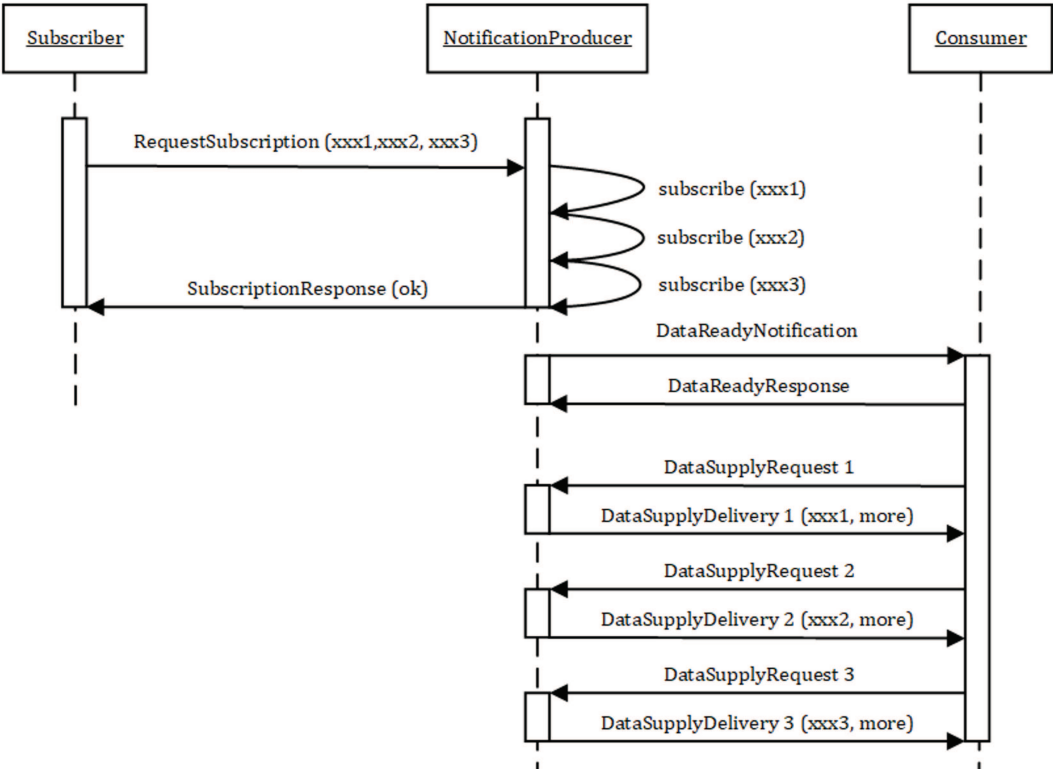


Figure 11 — Fetched Multipart Delivery

With the transmission of the final **ServiceDelivery** message, the Producer service considers the delivery of the data to be complete (i.e. it is permitted to reset the last update flags of the subscription as now being current with known notifications). A renewed **DataSupplyRequest** would then not be answered with the data of the recently polled subscription.

If the Optional SIRI Capability *ConfirmDelivery* is used, then after receiving the final Data Supply Delivery, the Consumer shall send a **DataAcknowledgement** message to the Producer to indicate that it has successfully received the data, which permits the Producer to close the delivery.

5.3 Mediation Behaviour

5.3.1 Introduction

In order to reduce the amount of network traffic and the volume of notification and delivery messages that consumers shall deal with, SIRI prescribes particular mediation behaviour on the part of the Notification Producer/Publisher. There are several different aspects to this particular mediation to which a SIRI implementation must conform; in particular; (i) Maintaining Subscription Last Updated State, and; (ii) Aggregating Subscriptions into a Subscriber Channel. Both of these are quite simple concepts, but when combined, especially with fetched delivery, lead to some complexity.

5.3.2 Mediation Behaviour – Maintaining Subscription Last Updated State

In the simplest incarnation of the WS-PubSub paradigm, notification messages are despatched to all interested subscribers immediately when they occur. When a Consumer joins a service it may be sent an initial delivery with the set of current data. Thereafter, it is sent only additional notifications and deliveries for new events. The full set of current data for a given topic expression may be retrieved by a Consumer at any subsequent time using the 'Get Current Message' function. In SIRI this basic mediation is further elaborated by (i) the requirement to support the *Fetched Delivery* pattern, and (ii) an ability to set a threshold value for sensitivity to change in data values for subscriptions in certain SIRI Functional Service types (notably Stop Monitoring, Vehicle Monitoring and Connection Monitoring).

With the SIRI *Fetched Delivery* pattern, payload data is not sent immediately along with the notification. This means that the *Notification Producer shall retain state for each subscription to hold the time that the most recent successful delivery was made to the individual Consumer*, so that when the Consumer comes to fetch the data with a data supply message, only the updates 'recent' to that individual Consumer are sent (i.e. those since the last fetch), and not the whole current data set.

The ability to establish *the point of last update for each individual subscription is also required to support the filtering of notification messages by a change threshold* regardless of whether *Direct* or *Fetched Delivery* is used. If a change threshold is set, then a notification for a data event representing a change to an earlier event will only be sent if the change in some quantitative value (for example, the predicted arrival time) exceeds the specified threshold of difference to that in the last delivery.

The use of Last Update mediation for a *Direct Delivery* is shown in Figure 12 as a simple message sequence example. (See Figure 13 for a *Fetched Delivery* sequence.) On subscription, the Producer computes the current data based on notifications from the Publisher (for example 1.1, 1.2) according to the subscription topic and sends it to the Consumer; the time (T1) of supply is recorded against the individual subscription. Subsequent notifications from the Publisher (2.1, 2.2, 2.3) are filtered until the sensitivity threshold is exceeded, at which point all the updates since the last update time T1 (2.1, 2.2, 2.3) are aggregated and sent and the new time of supply recorded (T2).

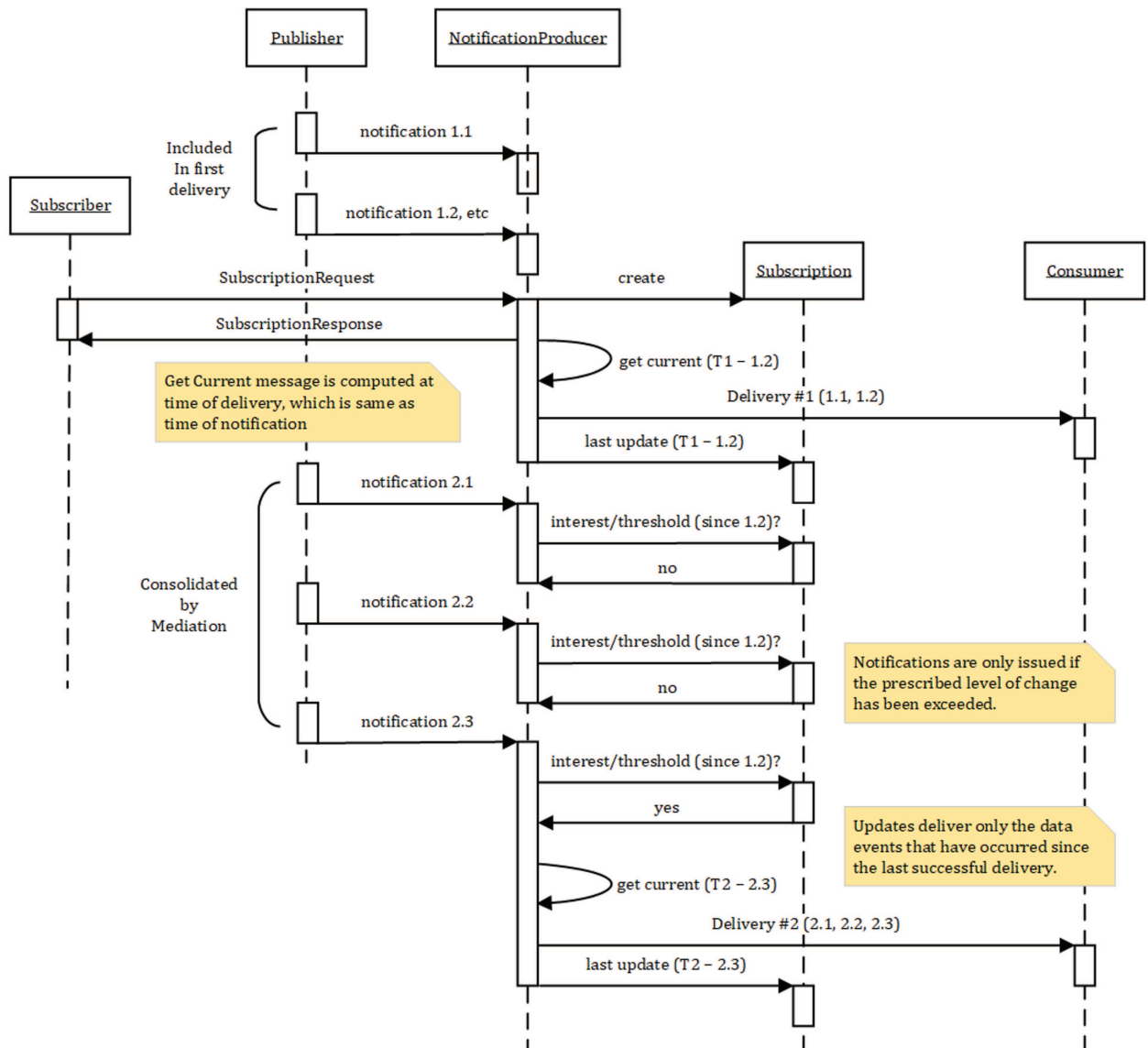


Figure 12 — Mediation: Update Tracking and sensitivity threshold for Direct Delivery

In a *Fetches Delivery* interaction, there may be a delay between the despatch of a Notification that a new update exists to the Consumer and the despatch of the Data Delivery message – which occurs only in response to an explicit Data Supply request from the Consumer. In the meantime, further situations may occur in the interim, giving rise to further notifications from the Publisher to the Notification Producer (but not to the Consumer – the Notification Producer also retains state that a notification has already been issued). The data delivery, when finally made, shall represent the most current position: the data supply message shall include all known subsequent data events that have taken place since the time of last data supply.

This is shown in Figure 13 for a *Fetches Delivery* sequence. On subscription, the Producer checks if there is any current data (1.1, 1.2) according to the subscription topic and if so, sends a data ready notification (#1) to the Consumer. When the Consumer requests the data supply, the Producer recomputes the current data (1.1, 1.2) and sends it to the Consumer. The time (T1) of supply is recorded against the individual subscription. Subsequent notifications from the Publisher (2.1, 2.2) are filtered until the sensitivity threshold is exceeded, at which point (T2) a new notification (#2) is sent to the Consumer. When the Consumer requests the data supply at a later time (T3), the Producer

recomputes the data since the last update which, as well as the notifications that triggered the Data Ready Notification (2.1, 2.2), in the example also includes an additional item (2.3) which has arrived in the meantime. The data is despatched to the Consumer and the new time (T3) of supply is recorded against the individual subscription.

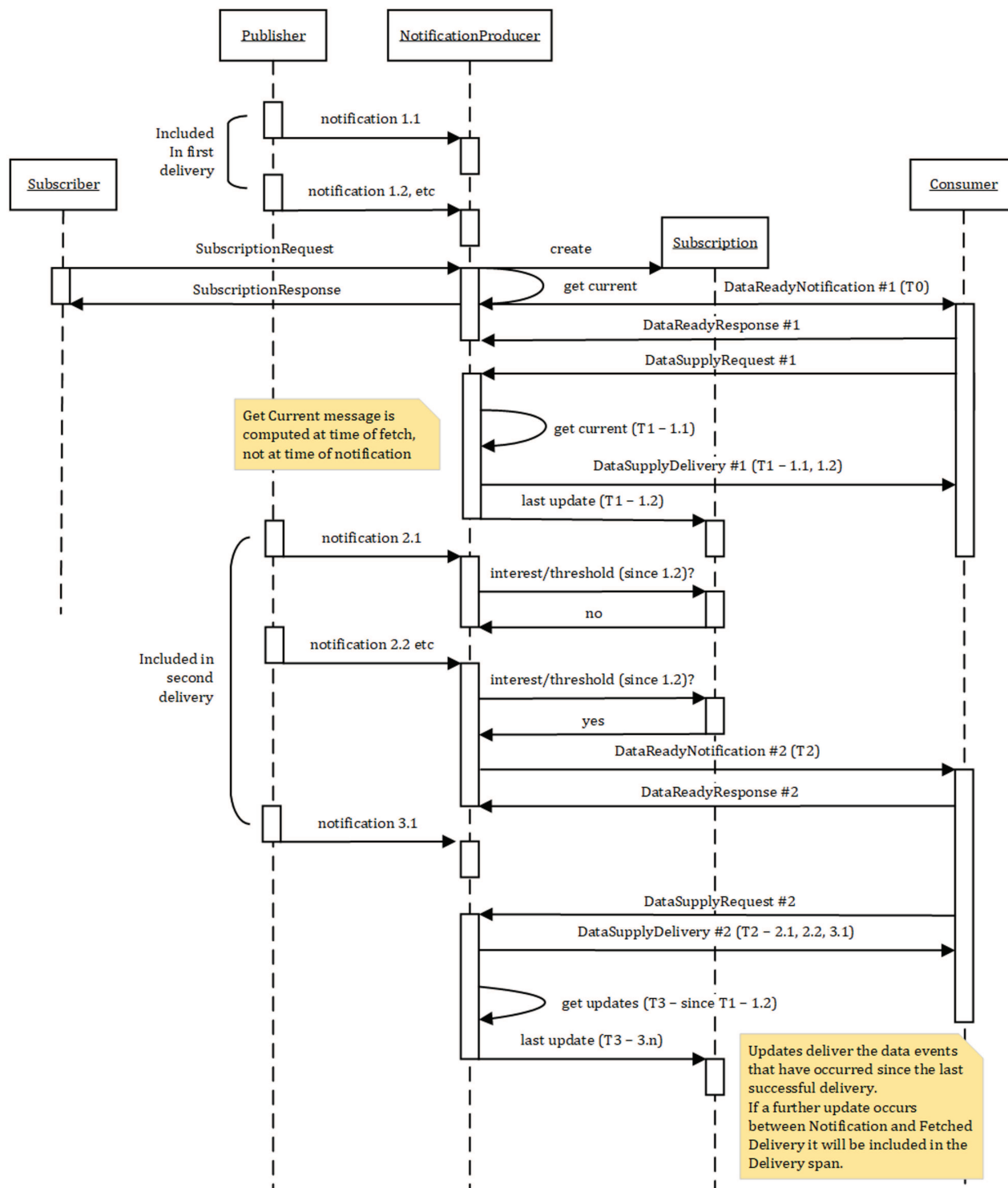


Figure 13 — Mediation: Handling Fetched Delivery Latencies

5.3.3 Mediation Behaviour – Subscription Filters

Another optimisation that SIRI can make in order to reduce the network traffic is the aggregation of subscriptions for a given Subscriber into a single filter group for notification and delivery.

When a Subscription is created for a Subscriber to a given SIRI functional service type, it is automatically assigned to a Subscription Filter. Subsequently, a *single notification is sent for all subscriptions of a Filter*. For example, if a SIRI Vehicle Monitoring Service Subscriber has subscriptions to two separate LINES, and SITUATIONs arise for both of them (from data notifications from the Publisher) within the data refresh or processing cycle of the Notification Producer, only a single Data Ready Notification will be sent to the service Consumer, and the single Data Supply message will return the payload for both data ready notifications as content within a single Service Delivery. Thus for each Subscription Filter, the system retains ‘*Already Notified*’ state, which gets reset every time last update is updated.

A subscriber may add different Subscriptions at different times to a Subscription Filter.

Some implementations support only a single Subscription Filter per Subscriber; others may support multiple filters. If multiple filters are supported and a Subscriber wishes to use a separate Subscription Filter, a Filter Identifier should be indicated on the subscription request. If no filter is specified on a request, the first filter created for the Subscriber will be used by default. The use of multiple filters allows a Subscriber interested in many different subscriptions from the same Notification Producer to keep transaction boundaries (and the computational overhead needed for the filtering, transmission and processing of delivery packages) down to a manageable size. Otherwise there is the possibility that a small urgent item of information will be swamped within a larger packet of data for additional content that is triggered for delivery at the same time.

Subscription Filtering is shown in Figure 14 for a *Direct Delivery* sequence (the *Fetches Delivery* sequence would be logically similar).

- The initial subscription request contains two separate individual service subscription requests (001, 002).
- On subscription, the Producer checks if there is any current data (1.1) according to both subscriptions’ topic expressions, and if any data is found, the initial delivery (#1) will contain data for both subscriptions.
- If an additional Subscription (003) is created for the Subscriber, the Producer will add it to the Subscriber’s Subscription Filter.
- The Producer only make a notification/delivery (#2) if there is any data for the new subscription: in Figure 14 none is shown as having occurred. However if there was any data, i.e. if any data is sent at all, it will include updates for any subscription of the Consumer, even if the other updates have not reached the threshold. Again, the time (T2) of supply will be recorded against each individual subscription.
- Subsequent notifications from the Publisher (2.1) are filtered until the sensitivity threshold is exceeded, at which point (T3) a new data notification and delivery (#3) is sent to the Consumer. The data supply includes updates since the last supply time T2 affecting any subscription in the Subscription Filter.

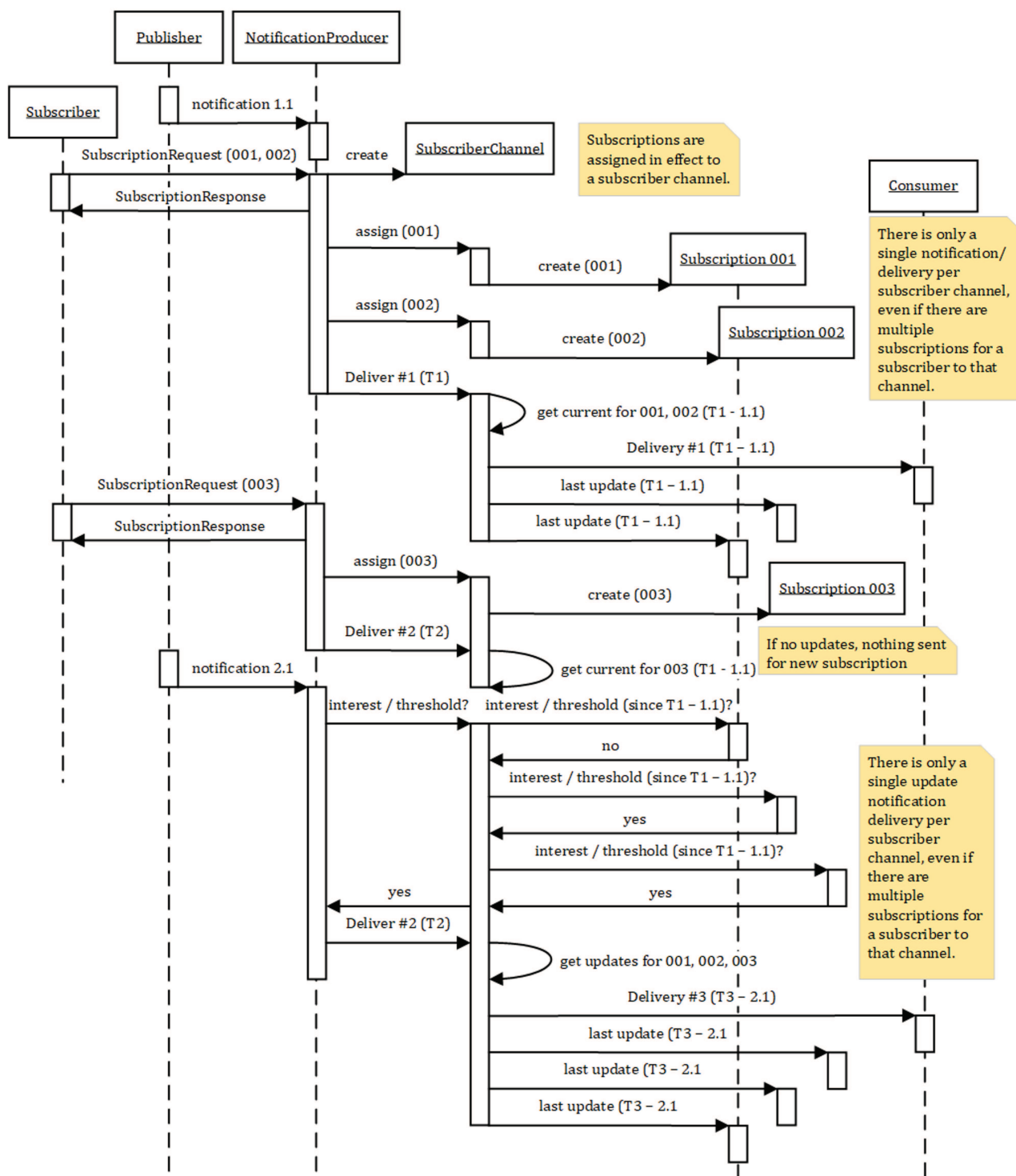


Figure 14 — Mediation: Subscription Filter

5.4 Recovery Considerations for Publish Subscribe

5.4.1 Introduction

Publish/Subscribe is a stateful pattern of interaction, and consideration shall be given to failure of either the Notification Producer, or Consumer systems, or to the communications connection between them.

Once created, subscriptions are held by the Notification Producer. In the event of a system failure by the Notification Producer service, SIRI does not require the Notification Producer to recover the subscriptions. Instead, it is the responsibility of the Subscriber to recreate new subscriptions to replace the existing ones lost in the failure. Each Consumer knows the Subscriber which submitted its subscriptions (usually these will be the same implementation entity).

It is then the Consumer's responsibility to monitor whether the Notification Producer Service and the connection to it are still active, and to inform the Subscriber if the Subscription needs to be renewed. SIRI supports two ways of monitoring the Notification Producer (i) Status Polling (a required SIRI capability), and (ii) Heartbeat (an optional SIRI capability). This allows implementers to choose the most efficient approach given the traffic and cycle characteristics of their deployment.

From v2.0 SIRI also allows a SubscriptionTerminatedNotification to be sent by a Producer to inform Subscribers that their subscriptions have been terminated unilaterally.

5.4.2 Check Status – Polling

The Consumer may send periodic Check Status requests to the Notification Producer to check that Notification Producer is still active. A Check Status message is a required feature of all SIRI implementations.

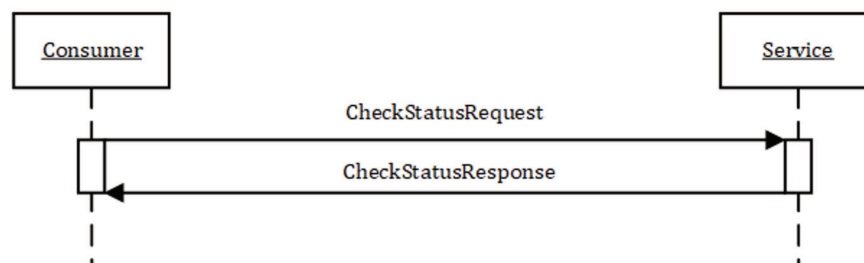


Figure 15 — Check Status — UML Sequence

5.4.3 Heartbeat – Pinging

Using a Heartbeat message, the Notification Producer sends a regular notification message to the Consumer at a predefined interval to show that it is still active, even if no update messages have been sent. This removes the need for the Consumer to poll, as the Consumer can detect a potential failure by the failure of a Heartbeat to arrive within the prescribed interval. The heartbeat interval may either be preconfigured for the whole system or be specified as a parameter in the subscription request specified for individual subscription on the Subscription Policy.

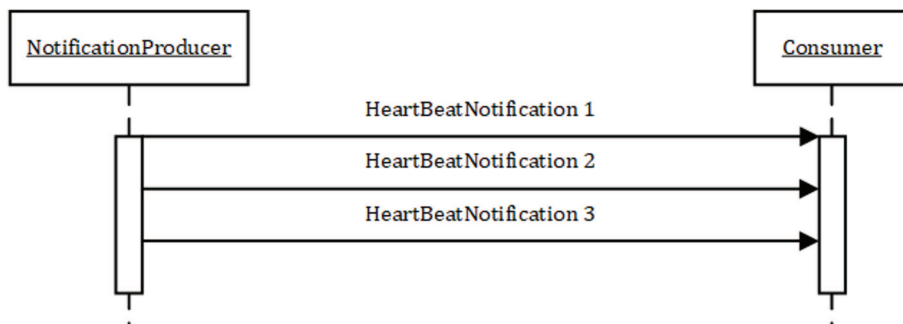


Figure 16 — Heartbeat Message — UML Sequence

5.4.4 Degrees of Failure

There are in effect different degrees of failure of the Producer, or in access to the producer: (i) *Total Failure* in which case the subscriptions are lost and (ii) *Partial failure*, in which case the subscriptions are still in existence, but the data flow is interrupted for a period. Total failure shall always be marked by a new Service Start time.

To recover from partial failure the Producer shall send all current data for a subscription since the last update for that subscription.

5.4.5 Detecting a Failure of the Producer

5.4.5.1 Detecting a Failure Using Check Status

A Consumer can at any time explicitly poll to detect a Service breakdown by the Notification Producer, by sending a **CheckStatusRequest** to the Service. A **CheckStatusResponse** should be returned. A Common Check Status message is used for all services (Figure 15).

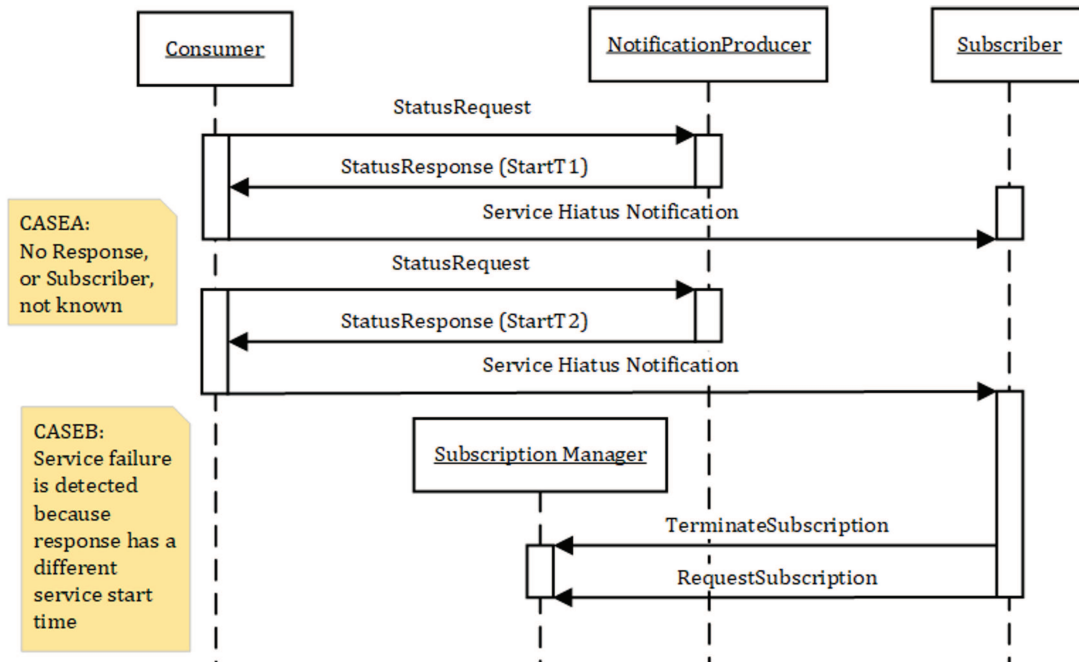


Figure 17 — Check Status with Recovery & Re-subscribe — UML Sequence

If there has been only a brief interruption of service, this may be detected by comparing the start time for the service returned on the check status message with the start time found on previous messages, (either a *SubscriptionResponse*, or a *CheckStatusResponse*). A hiatus suggests that there may be missed data, and/or subscriptions, and so the Consumer asks the Subscriber to recreate its subscriptions.

5.4.5.2 Detecting a Failure Using Heartbeat

If a Heartbeat capability is available, a Consumer may detect a Service failure by monitoring for the absence of *HeartbeatNotification* messages within the expected interval, rather than by polling explicitly for service availability and integrity with a *CheckStatusRequest*. If a failure is detected, the Consumer asks the Subscriber to recreate the subscription. Again there are two levels of failure: (i) *Total Absence of Service* with loss of subscriptions, and (ii) *Brief Interruption of Service*, with subscriptions retained but loss of real-time data.

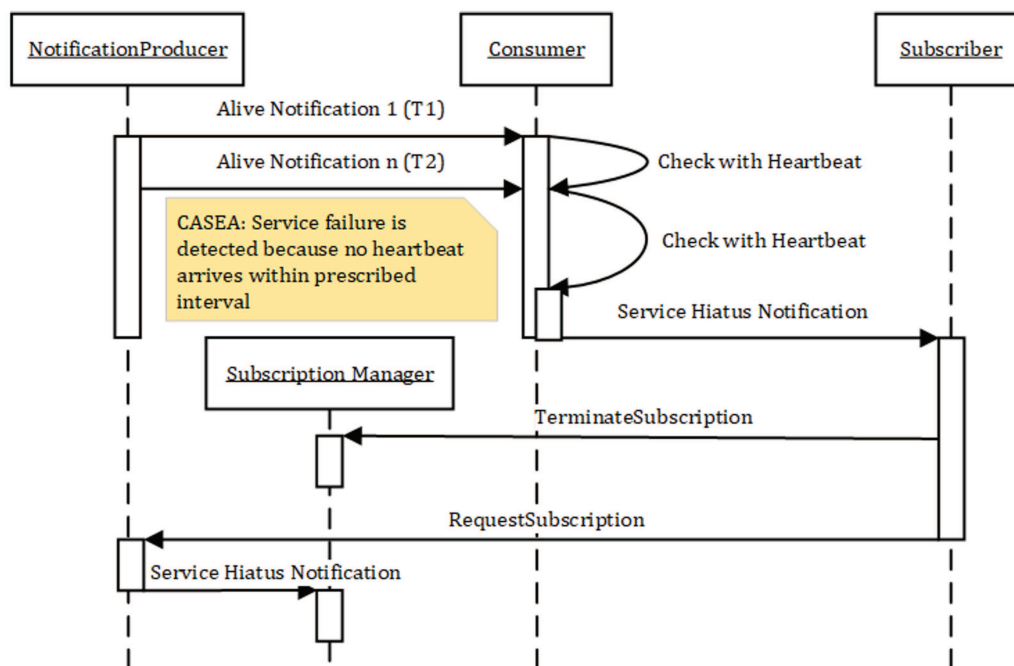


Figure 18 — Heartbeat Monitoring – Loss of Service — UML Sequence

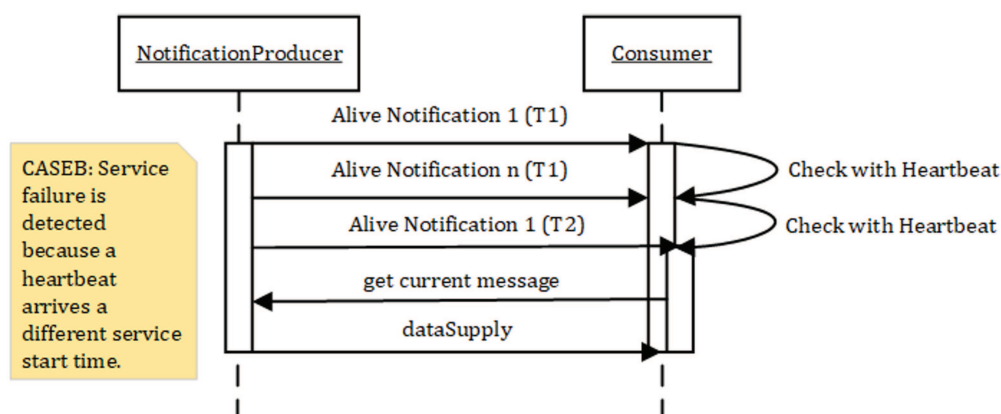


Figure 19 — Heartbeat Monitoring – Interruption of Service — UML Sequence

5.4.6 Detecting a Failure of the Consumer

It is the Consumer entity's responsibility to restart itself after a failure of its system. It must decide if it may have missed data and take any appropriate recovery action – either to get the current message, or to completely recreate its subscriptions.

In the event of System failure by the Notification Consumer, the Notification Producer will continue to send Notification messages to the Notification Consumer until the subscription has expired. In order to optimise efficiency, a well-behaved Consumer will terminate its subscription if it is no longer needed, or if a prolonged outage is planned.

An implementation may have a policy that states the Notification Producer will drop a subscription if it does not receive an acknowledgement for Deliveries within a certain period of tolerance.

5.5 Recovery Considerations for Direct Delivery

For Direct Delivery, two levels of resilience are possible: (i) *Simple Despatch* and (ii) *Acknowledged Despatch*.

For *Simple Despatch*, data is sent without an acknowledgement of receipt. For a *Request/Response* data supply interaction, the Requestor knows that it is waiting for a response and can determine whether it has been satisfied. However for asynchronous, i.e. *Publish/Subscribe* interaction, the Notification Producer can send a data message to the Consumer at any arbitrary point in time; so if for some reason the message fails to arrive, the transmission failure will not be detected by either party and so no recovery action will be attempted. In many cases this is sufficient, as another update shortly after will update the messages as effectively as a recovery action.

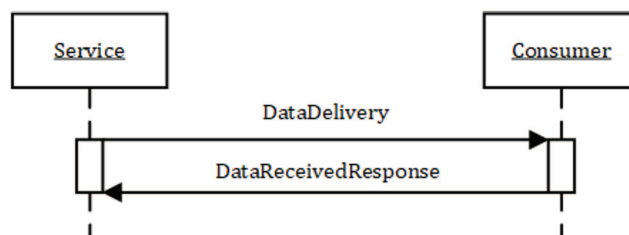


Figure 20 — Robust Direct Delivery — UML Sequence

With *Acknowledged Despatch*, reliable transmission is signalled through an additional 'Acknowledge Data Received' response from the Consumer to the Producer, confirming successful receipt of data. If the Notification Producer fails to receive a response from the Consumer, it would then keep resending at predefined intervals, exactly as for the initial notification step of *Full Fetched Delivery*.

5.6 Request Parameters and Interactions

Both *Request/Response* and *Publish/Subscribe* interactions involve the sending of requests to a Service running on a server. In both cases, the requests shall specify information about the content that is to be returned, including any variable aspects of real-time behaviour, as well as additional policies for carrying out the request, such as a throttle on the volume of data to return.

SIRI uses a standard set of parameters for the various terms of the requests, organised according to WS Pub-Sub concepts such as *Endpoint Reference*, *Topic*, *Request Policy*, *Subscription Policy* and *Payload*.

For a given type of SIRI Functional Service interface (for example Connection Timetable, Vehicle Monitoring, or General Message), the analogous groups of parameters are used on both *Request/Response* and *Publish/Subscribe* requests, for example to specify the content to be returned, (e.g. the data reference system, the topics of interest, the temporal window, and the level of detail), and the processing policy.

Additional parameters are specified on a subscription to control those aspects of processing which are specific to the asynchronous servicing of subscriptions, i.e. that apply to the Publisher and Notification Producer processes; for example, sensitivity threshold.

Some parameter types shall be included on individual requests; others may be configured for all message exchanges.

The SIRI parameter set includes some terms which are common to all requests, regardless of service interface (for example; end point references, delivery policies), and others which vary according to the

individual Service type. Table 1 shows the general groups of parameters, and whether they are common or whether they vary by message type.

Table 1 — SIRI Request and Subscription Parameters

Group	Subgroup	SIRI REQUEST	SIRI SUBSCRIPTION	COMMON	Notes
Identity	Identity	<i>RequestTimestamp</i>		Y	Timestamp
End-point_References	Address	<i>Address</i>	<i>Address</i>	Y	URL to respond
			<i>ConsumerAddress</i>	Y	
	Participant	<i>RequestorRef</i>	<i>SubscriberRef</i>	Y	Participant Reference, also known as Control Centre Code
		<i>MessageIdentifier</i>	<i>SubscriptionIdentifier</i>	Y	Issued by Subscriber
Content		<i>SiriServiceVersion</i>		Y	
	Topic Filters	<i>Topic</i>		N	Depends on message type
		<i>TemporalWindow</i>		N	Depends on message type
		<i>DetailLevelToReturn</i>		N	Depends on message type
		<i>Language</i>		Y	May be configured.
	Policy	<i>VolumeFilter</i>		Y	Number of items in Response. Depends on message type
	Delivery	<i>DeliveryMethod</i>		Y	Direct or Fetched. May be preconfigured.
		<i>AllowMultipartDelivery</i>		Y	May be preconfigured
Subscription	Lease	–	<i>InitialTerminationTime</i>	Y	Also known as. Lease Wanted.
	Notification Filters	–	<i>SensitivityFilter</i>	N	Also known as. Hysteresis. Depends on message type.
	Heartbeat Policy	–	<i>HeartbeatPolicy</i>	Y	May be preconfigured.

The general parameters which are common to all service requests are concerned with managing the messages: that is, (i) how messages may be referenced, and (ii) the delivery policy: how responses should be delivered.

The parameters which vary according to each SIRI Functional Service are concerned with (i) the content to be included – the *Topic* terms, (ii) how the request should be interpreted – the *Request Policy* terms, and (iii) any parameterised aspects controlling the Publishing and Notification processes – the *Subscription Policy* terms (e.g. sensitivity threshold, preview window etc.). Table 2 provides a comparative list of the SIRI Functional Service Specific parameters.

Table 2 — Topics and Policies for SIRI Functional Service Types

	Production Timetable	Estimated Timetable	Stop Timetable	Stop Monitoring	Vehicle Monitoring	Connection Timetable	Connection Monitoring
Topic (reference) content	<i>Validity-Period</i>	<i>Preview-Interval</i>		<i>Preview-Interval</i>		–	<i>Preview-Interval</i>
	–	–	<i>Departure Window</i>	<i>StartTime</i>	–	<i>Arrival-Window</i>	<i>Connecting-TimeFilter</i>
	<i>TimetableVersionRef</i>		<i>MonitoringRef</i>		<i>Vehicle-Monitoring-Ref</i>	<i>ConnectionLinkRef</i>	
	<i>OperatorRef</i>		–	<i>OperatorRef</i>	–	–	–
	<i>LineRef</i>						
	<i>DirectionRef</i>						
	<i>VehicleMode</i>		–	–	–	–	–
	<i>ProductCategoryRef</i>		–	–	–	–	–
	<i>StopPointRef</i>		–	–	–	–	–
	–	–	–	<i>DestinationRef</i>		–	–
	–	–	–	<i>StopVisit-Types</i>	<i>VehicleRef</i>	–	<i>Dated-Vehicle-JourneyRef</i>
Request Policy				<i>Maximum-StopVisits</i>	<i>Maximum-Vehicles</i>		
				<i>Minimum-StopVisits</i>			
				<i>Maximum-TextLength</i>			
	<i>Language</i>						
	<i>IncludeTranslations</i>						
	<i>IncludeInterchanges</i>		–	–	–	–	–
	<i>IncludeJourneyRelations</i>		–	–	–	–	–
	–	–	–	<i>DetailLevel</i>		–	–
	–	–	–	<i>MaxPreviousCalls</i>		–	–
	–	–	–	<i>MaxOnwardsCalls</i>		–	–
Subscription Policy	<i>IncrementalUpdates</i>	–	–	<i>IncrementalUpdates</i>		–	–
	–	–	–	<i>ChangeBeforeUpdates</i>		–	<i>Change-Before-Updates</i>
	–	–	–	<i>UpdateInterval</i>		–	–

	General Message	Situation Exchange	Facility Monitoring
Topic (reference) content	<i>PreviewInterval</i>	<i>PreviewInterval</i>	<i>PreviewInterval</i>
	–	<i>StartTime</i>	<i>StartTime</i>
	<i>InfoChannelRef</i>	<i>SituationRef</i>	<i>FacilityRef</i>
	–	<i>Various properties</i>	<i>Various properties</i>

	General Message	Situation Exchange	Facility Monitoring
Request Policy	<i>Language</i>		
	<i>IncludeTranslations</i>		
	–	<i>MaximumNumberOfSituationElements</i>	<i>MaximumNumberOfFacilityConditions</i>
Subscription Policy	–	<i>IncrementalUpdates</i>	<i>IncrementalUpdates</i>

5.7 Error Conditions for Requests

Requests may fail for a variety of reasons, in practice the reasons fall into two groups: Systemic and Application.

- *Systemic* error conditions prevent the request from being interpreted further: for example, the request times out, or the request itself cannot be validated against the prescribed XML schema version, or the system does not support the requested version level; in which case the request will be rejected outright. The request string is echoed back to assist diagnosis. Typically the errors occur in the communications or transport layer. They are generic, i.e. not specific to the SIRI application.
- *Application* level error conditions involve the interpretation of the request parameters and the detection of an error condition according to the semantics of the application. For example, a Stop Monitoring request might ask for information about a stop that is not covered by the system. Most terms of a request have a specific application error condition associated with them.

Table 3 describes the SIRI error conditions, with condition, description, code and severity (sev).

Table 3 — System and Application Error Conditions

Group	Condition	Description	Code	Sev
Success	OK (true)	Request successful	200	5
Systemic Error	<i>RequestTimeout</i>	Server not responding	408	1
	<i>InvalidRequest</i>	The server does not "understand" the request. The client should not repeat the request.	400	1
	<i>Unauthorized</i>	User name and password are required for the request, or credentials not satisfied	401	1
	<i>Forbidden</i>	The server "understands" the request but cannot carry it out.	403	2
	<i>NotFound</i>	The requested URL was not found.	404	1
Distribution	<i>UnknownParticipant</i>	Recipient for a message to be distributed is unknown. +SIRI v2.0	601	2
	<i>UnknownEndpoint</i>	Endpoint to which a message is to be distributed is unknown. +SIRI v2.0	602	2
	<i>EndpointDeniedAccess</i>	Distribution message could not be delivered because not authorised. +SIRI v2.0	603	2
	<i>EndpointNotAvailable</i>	Recipient of a message to be distributed is not available. +SIRI v2.0	604	2
Access	<i>UnapprovedKey</i>	User authentication key is not approved. SIRI v2.0	610	1
Application Error	<i>VersionNotSupported</i>	Version of SIRI interface is not supported.	701	2
	<i>CapabilityNotSupported</i>	Service does not support the requested capability.	704	2

Group	Condition	Description	Code	Sev
	<i>ServiceNotAvailable</i>	Functional service is not available to use (but it is still capable of giving this response).	710	2
	<i>AccessNotAllowed</i>	Requestor is not authorised to the service or data requested because a capability is not enabled.	720	2
	<i>InvalidDataReferences</i>	Request contains references to identifiers that are not known. +SIRI v2.0	730	2
	<i>BeyondDataHorizon</i>	Request is for data outside of real-time data horizon.	732	2
	<i>NoInfoForTopic</i>	Valid request was made but service does not hold any data for the requested topic expression.	740	4
	<i>ParametersIgnored</i>	Request contained parameters that were not supported by the producer. A response has been provided but some parameters have been ignored. +SIRI v2.0	742	3
	<i>UnknownExtensions</i>	Request contained extensions that were not supported by the producer. A response has been provided but some or all extensions have been ignored. +SIRI v2.0	743	3
	<i>UnknownSubscriber</i>	Subscriber not found.	721	2
	<i>UnknownSubscription</i>	Subscription not found.	722	2
	<i>AllowedResource-UsageExceeded</i>	Valid request was made, but request would exceed the permitted resource usage of the client.	742	2
	<i>OtherError</i>	Other Error Type	700	

Each Application Error Condition arising from a failed request comprises an error code and a textual description. Each term of a SIRI Functional Service request typically has a different error condition associated with it. The SIRI schema defines explicit error code for application errors wherever possible: these are reified as concrete tags, for example, ***NoInfoForTopicError***, ***UnknownSubscriber***, etc. The most specific error condition should always be returned – the catchall ***OtherError*** should only be used in exceptional circumstances.

Table 4 relates the request terms to their possible error conditions.

Table 4 — Application Error Conditions Related to Request Parameters

Group	Subgroup	Request Term	Possible Error Condition	Notes
System	Version	<i>SIRIVersion</i>	<i>VersionNotSupported</i>	Must respond with same level.
Endpoint References	Endpoint	<i>ParticipantRef</i>	<i>UnknownSubscriber</i>	i.e. Control Centre not known.
	Identity	<i>SubscriptionIdentifier</i>	<i>UnknownSubscription</i>	
	Credentials	<i>Credentials</i>	<i>AccessNotAllowed</i>	Not known, or not authorised.
Common Content	Version	<i>SIRIServiceVersion</i>	<i>VersionNotSupported</i>	
	Topic Filters	<i>Topic</i>	<i>NoInfoForTopic.</i> <i>CapabilityNotSupported.</i>	Depends on message type.
		<i>TemporalWindow</i>	<i>BeyondDataHorizon</i>	Depends on message type.
		<i>DetailLevelToReturn</i>	<i>CapabilityNotSupported</i>	Depends on message type.
		<i>Language</i>	<i>CapabilityNotSupported</i>	
	Policy	<i>VolumeFilter</i>	<i>AllowedResource-UsageExceeded</i>	
	Delivery	<i>DeliveryMethod</i>	<i>CapabilityNotSupported</i>	
Subscription	Lease	<i>InitialTermination-Time</i>	<i>BeyondDataHorizon</i>	Depends on message type.

Group	Subgroup	Request Term	Possible Error Condition	Notes
	Notification Filters	<i>SensitivityFilter</i>	<i>AllowedResource-UsageExceeded</i>	Depends on message type.
	Heartbeat Policy	<i>HeartbeatPolicy</i>	<i>CapabilityNotSupported</i>	

5.8 Versioning

5.8.1 Introduction

Several different release versions of the SIRI schema may be current at the same time and may be run on the same client or server computer.

Each Request or Subscription Request should indicate a version level. If the SIRI Functional Service supports the version level requested, then all responses, both synchronous and asynchronous, should be at the same version level as that of the request. If the version level is not supported, an error condition will be returned. This makes it possible to deploy new upgrades at different times on different systems and still maintain continuous operations.

There are in effect two separate version levels within SIRI: (i) the *SIRI Version*, and (ii) the *SIRI Functional Service Version*. Both use the standard version 'n.mx' numbering scheme as per ISO 24531:2013 where 'n' is the major release number, 'm' is a point version number, and 'x' indicates a draft status. For example; '2.0', '3.2a'

5.8.2 The Overall SIRI Framework Version Level

The overall *SIRI Version* indicates the level of the general messages and communications framework.

5.8.3 The SIRI Functional Service Type Version Level

The *SIRI Functional Service Version* indicates the level of a specific SIRI Functional Service, such as the Stop Monitoring or General Message service. Each SIRI Functional Service type has its own version level. The SIRI Functional Service knows whether its version level is compatible with the Framework version level, so in practice the SIRI Functional Service Type Version level can be used as a single definitive version identifier on requests for a particular SIRI Functional Service type.

5.9 Access Controls: Security and Authentication

5.9.1 Introduction

Implementers may wish to control access by subscribers and consumers to all or some services and to all or some data with a service. SIRI considers two separate layers of access control; (i) System Level, and; (ii) Application Level.

5.9.2 System Mechanisms External to SIRI Messages

5.9.2.1 General

SIRI may be used in combination with a normal range of System Level Security Measures, such as IP authentication, encryption, or VPN tunnelling. These are general purpose mechanisms, implemented at the transport level, that are independent of SIRI. Implementers will choose mechanisms appropriate to the security requirements of their own deployments.

WS-PubSub lays down a number of recommendations for the securing of systems see [WS-Security, www.oasis-open.org] and traffic. In particular only authorised participants should be allowed access to access or subscribe to the system in the first place.

The SIRI Simple Web Services (+SIRI v2.0) are intended to support direct delivery of SIRI data to consumer devices and additionally may include an authentication key.

5.9.2.2 Authentication Key (+SIRI v2.0)

SIRI requests may include authentication data elements. These are primarily for use by SIRI simple web services which may wish to control or track access to end services by particular applications. An Account Identifier may be used to attribute requests to a specific user account for authentication or reporting purposes. An Authentication key may be used to authenticate the request to ensure the user is a registered client for that user ID.

NOTE The account is not typically for an individual user, but for an application. It could be used to track and authenticate an individual user, but then appropriate consent and privacy mechanisms should also be included.

5.9.2.3 Application Level Authentication

SIRI also supports the concept of Application Level Authentication to allow a given Service Provider to specify an approved requestor or subscriber's allowed access to specific types of content and to specific levels of resource usage by the application. Access control is an optional feature of SIRI, and may be supported for some or all services. Application level access controls allow the same SIRI system to provide services to many different customers from the same endpoint addresses, restricting the information out given to individual authorised users to use specific content and services.

There are two components to the SIRI Access Controls: (i) Configuration through an Access Permission Matrix, and (ii) Run time Request Authentication.

5.9.2.4 The Access Permission Matrix

The Access Permission matrix is a configuration-time schema used to specify the access that a Service will provide to specific users. It comprises a tuple of a Participant Reference, a Service Type and Capability, and a Service specific Topic or Policy. For example, an entry might specify that participant 'ABC' may make request/response requests to the SIRI Stop Monitoring service to see departures for a specific stop with identifier '123' (See 6.1.6 for an example encoding). Permissions for different SIRI functional services are described under the individual service sections in Part 3.

5.9.2.5 Request Authentication

The Notification Producer uses the Participant Reference to validate requests against the Access Permission matrix. Invalid requests are rejected with an appropriate error condition. As far as possible, subscription requests are authenticated once at the time of subscription, rather than dynamically on each data notification.

5.10 Service Discovery

5.10.1 Introduction

Once there are many systems covering a large number of stops, it is of great benefit to have discovery services that allow systems to automatically provision the reference data to be used. Service discovery can be broken down into three steps: (i) Universal Server Discovery; (ii) SIRI Capability Discovery; and (iii) SIRI Functional Service Coverage Discovery.

5.10.2 Discovery of Servers that Support SIRI Services

The first stage in universal discovery is to find the servers and sites supporting the SIRI protocols. Since current use of AVMS almost invariably involves interaction between known parties who operate subject to appropriate agreements, universal discovery is not currently a SIRI requirement. If needed in future, general purposes web service discovery protocols such as WSDL are likely to be sufficient.

The Discovery services conform to the WS-PubSub requirement to have mechanisms to disclose metadata about the publication services.

5.10.3 Discovery of the Capabilities of a SIRI Server

Once a SIRI server is known, a client needs to know which SIRI Functional Services and features it supports, and at which version level. SIRI defines a simple Capability Discovery message, which allows a client to obtain this information from each SIRI Producer Service. The Service returns a fixed list of the capabilities supported by the service, and the configuration details. See Capability Matrix in 5.11, and individual service descriptions in Part 3.

Implementing a **CapabilityDiscoveryResponse** is not mandatory; however if the implementation does not provide an actual **CapabilityResponse**, it should at least be possible to provide an exact description of its capabilities, version level and configuration, as a static XML document identical to the payload of a **CapabilityDiscoveryResponse** message.

The **CapabilityDiscoveryResponse** can be used to implement adaptive services, which are able to tune their interaction to the capabilities of the service; the Capability model also has an important role for documenting the exact specification of a particular implementation.

The **CapabilityDiscoveryResponse** also returns the **ServiceRequestContext** which provides a set of configuration parameters that formally define the properties of a given SIRI implementation.

5.10.4 Discovery of the Coverage of a Given SIRI Functional Service

Once a client knows what services a SIRI server can provide, it may wish to know what data it holds, and what access rights it has to the data and services.

Application data discovery services are specific to the application content of the SIRI Functional Services. For example, to find the stops that a particular SIRI Stop Monitoring Service covers or the Product Category codes used. Such discovery services can be extremely useful for reducing the costs of provisioning and re-provisioning systems. A coverage service typically involves a simple request that is run occasionally by the client to find out all the values of reference data that may be used in requests. For example the places or points, or sets of reference data such as the LINES and DESTINATIONS supported. In SIRI coverage discovery services are optional and are discussed under the particular service type. Table 5 shows the SIRI coverage discovery services.

Table 5 — SIRI Discovery Service Matrix

Feature	Required?	Capability Name	Notes	
Discovery	0	<i>StopDiscovery</i>	Returns Stops covered by service	SERVICE FRAME
	0	<i>LineDiscovery</i>	Returns Lines & direction covered by service	SERVICE FRAME
	0	<i>ProductCategoryDiscovery</i>	Returns Product categories used in SIRI	RESOURCE FRAME
	0	<i>ServiceFeatureDiscovery</i>	Returns Service features used in SIRI	RESOURCE FRAME
	0	<i>VehicleFeatureDiscovery</i>	Returns Vehicle features used in SIRI	RESOURCE FRAME
	0	<i>InfoChannelDiscovery</i>	Returns available SIRI GMS info channels	SIRI ONLY
	0	<i>ConnectionLinksDiscovery</i>	Returns Connection Links covered by service	RESOURCE FRAME

SIRI includes as informative content examples of a discovery services.

Alternatively, it is possible to use NeTEx to exchange any type of reference data, including stops, lines, etc. NeTEx data objects may be exchanged using the SIRI protocol, using a NeTEx **DataObjectRequest** & **DataObjectDelivery**(see the NeTEx specification). A SERVICE FRAME is used to group SCHEDULED STOP POINTs, LINES. A RESOURCE FRAME is used to group a set of Product categories or other codes.

5.11 Capability Matrix

5.11.1 Introduction

SIRI comprises a number of different functional services and capabilities; some of the capabilities can be used as alternatives. Each SIRI Capability is given a name, indicating a related set of specific input parameters, output parameters or processing behaviour which will be present if the capability is supported, and that will be absent if it is not. The capabilities serve several purposes:

- To document the functions of SIRI in a modular fashion.
- To indicate which features are essential and which are optional.
- To allow a variety of levels of implementation to match different operational requirements.
- To accommodate different legacy variations.
- To allow customers to specify exactly what functions should be present in a system being procured.
- To manage the compatibility of different implementations.

Table 6 summarises the named SIRI Functional Capabilities and whether they are required or optional.

The SIRI Capability Discovery service (see 11.2), Table 42 — CapabilityDiscoveryRequest is itself an optional capability of a SIRI implementation which may be used to discover the Capabilities of a given implementation.

5.11.2 SIRI General Capabilities

Table 6 — SIRI General Capabilities

Feature	Required?	Capability Name	Sub-capability
Management	R	<i>Versioning</i>	<i>RequestChecking</i>
	R	<i>Capability</i>	<i>CapabilityChecking</i>
	O		<i>CapabilityDiscovery</i>
Interaction	At least one	<i>InteractionPattern</i>	<i>DirectRequest</i>
			<i>Publish/Subscribe</i>
	R	<i>Mediation</i>	<i>GetCurrent</i>
	R		<i>GetLastUpdate</i>
	R		<i>ChangeSensitivity</i>
	O		<i>Historic</i>
	R	<i>SubscriptionFilter</i>	<i>SingleFilter</i>
	O		<i>MultipleFilters</i>
	O	<i>DynamicContext</i>	--
	O	<i>AccessControl</i>	<i>ByCapability</i>
	O		<i>ByTopicValue</i>
Delivery	At least one	<i>DeliveryMethod</i>	<i>DirectDelivery</i>
			<i>FetchDelivery</i>
	O	<i>ConfirmDelivery</i>	--
	O	<i>VisitCountsOrder</i>	--
Service Status	R	<i>CheckStatus</i>	--
	O	<i>Heartbeat</i>	--
Discovery	O	<i>Capabilities</i>	--
Message Transport	R	<i>MessageTransport</i>	<i>HttpPost</i>
	O		<i>SoapEnvelope</i>
	At least one	<i>Addresses</i>	<i>Implicit</i>
			<i>Explicit</i>
	O	<i>Compression</i>	<i>None / Gzip / other</i>

6 Request/Response

6.1 Making a Direct Request

6.1.1 Introduction

Request/Response is the simplest pattern of SIRI interaction. For data exchange, the requestor sends a Service Request to a Specific SIRI Functional Service as located by the Service's Endpoint Reference (see 10.2 later), and is returned an immediate data delivery, i.e. a response message that contains application payload data.

Each specific SIRI Functional Service Request is wrapped within a general **ServiceRequest** element, and the corresponding delivery is similarly wrapped within a **ServiceDelivery** element. There is a different SIRI Functional Service Request message type for each different SIRI Functional Service, and also a distinct SIRI Functional Service Delivery message with which to return the content for the individual service (see Table 7).

Table 7 — SIRI Request Delivery Types

	SIRI Functional Service	Request	Delivery
	Container	<i>ServiceRequest</i>	<i>ServiceDelivery</i>
Timetable	Production	<i>ProductionTimetableRequest</i>	<i>ProductionTimetableDelivery</i>
	Real-time	<i>EstimatedTimetableRequest</i>	<i>EstimatedTimetableDelivery</i>
Progress	Stop Timetable	<i>StopTimetableRequest</i>	<i>StopTimetableDelivery</i>
	Stop Monitoring	<i>StopMonitoringRequest</i>	<i>StopMonitoringDelivery</i>
		<i>StopMonitoringMultipleRequest</i>	<i>StopMonitoringDelivery</i>
	Vehicle Monitoring	<i>VehicleMonitoringRequest</i>	<i>VehicleMonitoringDelivery</i>
Interchange	Connection Timetable	<i>ConnectionTimetableRequest</i>	<i>ConnectionTimetableDelivery</i>
	Connection Monitoring	<i>ConnectionMonitoringRequest</i>	<i>ConnectionMonitoringFeederDelivery</i>
			<i>ConnectionMonitoringDistributorDelivery</i>
Info	General Message	<i>GeneralMessageRequest</i>	<i>GeneralMessageDelivery</i>
Amenities	Facility Monitoring	<i>FacilityMonitoringRequest</i>	<i>FacilityMonitoringDelivery</i>
Incidents	Situation	<i>SituationExchangeRequest</i>	<i>SituationExchangeDelivery</i>

Multiple Functional Service Requests may be included within a **ServiceRequest**, but all requests must be for the same service type.

6.1.2 ServiceRequest Message — Element

The **ServiceRequest** is sent to the [GetData] endpoint of a SIRI Functional Service. Table 8 shows the common parameters that may be specified on a **ServiceRequest**.

Each request may contain Endpoint information, including Endpoint Reference Properties. For Direct delivery the endpoint address is the [Consumer] endpoint to which the data is sent. For fetched Delivery it is the [Notify] endpoint to which the data ready notification is to be sent.

The Endpoint properties may include:

- The *Participant Reference*, which will be unique to the requestor in communication between the two parties
- A *Message Identifier*, with which to reference the specific request message in subsequent conversations, and which will be unique within the scope of the SIRI Functional Service Type and the Participant scope.

NOTE Both the **ServiceRequest** and the concrete SIRI Functional Service Requests contained within it can have their own specific message references.

If SIRI Access Controls are supported, the Participant Reference is used to determine if the Requestor is authorised to make the request for which it has asked. The Reference will be checked against the Access Matrix, and if the permitted access is exceeded, the **NotAuthorised** error condition will be returned.

Table 8 — ServiceRequest — Attributes

ServiceRequest			<i>+Structure</i>	Request from a Consumer to a Producer for immediate delivery of data. Answered with a ServiceDelivery . (For <i>Fetches Delivery</i> this will be after a further DataReadyRequest).
	ServiceRequestContext	0:1	<i>+Structure</i>	General request properties – typically configured rather than repeated on request.
<i>log</i>	RequestTimestamp	1:1	<i>xsd:dateTime</i>	Timestamp on request.
<i>Auth.</i>	AccountId	0:1	<i>+Structure</i>	Account Identifier. May be used to attribute requests to a specific user account for authentication or reporting purposes +SIRI v2.0
	AccountKey	0:1	<i>+Structure</i>	Authentication key for request. May be used to authenticate the request to ensure the user is a registered client. +SIRI v2.0
<i>End-point Properties</i>	Address	0:1	<i>Endpoint-Address</i>	Address to which response is to be sent: [Notify] endpoint. If omitted, this may also be determined from RequestorRef and pre-configured data, or the http request.
	RequestorRef	1:1	<i>→Participant-Code</i>	Identifier of Requestor. May be used to identify an individual participant system or individual device client. If used for a device client should be an anonymous token, divulged with user consent.
	MessageIdentifier	0:1	<i>Message-Qualifier</i>	Arbitrary identifier that may be given to message.
<i>Delegator End-point</i>	DelegatorAddress	0:1	<i>Endpoint-Address</i>	Address of originated system to which delegated response is to be returned. +SIRI 2.0. If request has been proxied by an intermediate aggregating system this provides tracking information relating to the original requestor. This allows the aggregation to be stateless.
	DelegatorRef	0:1	<i>→Participant-Code</i>	Identifier of delegating system that originated message. +SIRI 2.0
	<i>Concrete service subscription</i>			If more than one, must all be same type.
<i>Payload</i>	<i>a</i>	ProductionTimetableRequest	<i>-1:*</i>	See SIRI Part 3 – Production Timetable.
	<i>b</i>	EstimatedTimetableRequest		See SIRI Part 3 – Estimated Timetable.

ServiceRequest			+Structure	Request from a Consumer to a Producer for immediate delivery of data. Answered with a ServiceDelivery . (For <i>Fetches Delivery</i> this will be after a further DataReadyRequest).
	c	StopTimetableRequest	+Structure	See SIRI Part 3 – Stop Timetable.
	d	StopMonitoringRequest	+Structure	See SIRI Part 3 – Stop Monitoring.
	e	StopMonitoringMultiple-Request	+Structure	See SIRI Part 3 – Stop Monitoring.
	f	VehicleMonitoringRequest	+Structure	See SIRI Part 3 – Vehicle Monitoring.
	g	ConnectionTimetableRequest	+Structure	See SIRI Part 3 – Connection Timetable.
	h	ConnectionMonitoringRequest	+Structure	See SIRI Part 3 – Connection Monitoring.
	i	GeneralMessageRequest	+Structure	See SIRI Part 3 – General Message.
	j	FacilityMonitoringRequest	+Structure	See SIRI Part 4 – Facility Monitoring. SIRI v1.3.
	k	SituationExchangeRequest	+Structure	See SIRI Part 5 – Situation Exchange. SIRI v1.3.

6.1.3 The ServiceRequestContext — Element

6.1.3.1 General

The **ServiceRequestContext** contains any general configuration parameters that are common to all request types and that may be preconfigured, rather than being repeated on individual requests. Examples of such parameters are the delivery method and the default National Language. A primary role of the **ServiceRequestContext** is for documentation: it records a number of important properties of a given SIRI implementation. Normally the context is fixed for the implementation and cannot be changed on individual request, so is not explicitly passed. If the implementation supports a **DynamicContext**, then a context may be attached to individual requests to specify overrides to those properties which the implementation allows to be set dynamically. The SIRI Capability Discovery Request can be used to retrieve the default context.

Table 9 shows the common parameters that may be specified in a **ServiceRequestContext**.

Table 9 — ServiceRequestContext Parameters

ServiceRequestContext			+Structure	General request properties – typically configured rather than repeated on request.
Server Endpoint Address	CheckStatusAddress	0:1	EndpointAddress	Address to which CheckStatus requests are to be sent.
	SubscribeAddress	0:1	EndpointAddress	Address to which requests for new subscriptions are to be sent.
	ManageSubscriptionAddress	0:1	EndpointAddress	Address to which requests to manage existing subscriptions are to be sent. If absent, same as SubscribeAddress .
	GetDataAddress	0:1	EndpointAddress	Address to which requests to return data are to be sent.
Client End- point Address	StatusResponseAddress	0:1	EndpointAddress	Address to which CheckStatus responses and HeartbeatNotification messages are to be sent. If absent, same as SubscriberAddress .
	SubscriberAddress	0:1	EndpointAddress	Address to which subscription responses are to be sent.

ServiceRequestContext			+Structure	General request properties – typically configured rather than repeated on request.
	NotifyAddress	0:1	<i>EndpointAddress</i>	Address to which notifications requests are to be sent. If absent, same as <i>SubscriberAddress</i> .
	ConsumerAddress	0:1	<i>EndpointAddress</i>	Address to which data is to be sent. If absent, same as <i>NotifyAddress</i> .
<i>Name-space</i>	DataNameSpaces	0:1	+Structure	Scope for identifiers
<i>Name-Space</i>	StopPointNameSpace	0:1	<i>xsd:anyUri</i>	Namespace for stop references.
	LineNameSpace	0:1	<i>xsd:anyUri</i>	Namespace for LINE names and DIRECTIONS.
	ProductCategory-NameSpace	0:1	<i>xsd:anyUri</i>	Namespace for product categories
	ServiceFeatureNameSpace	0:1	<i>xsd:anyUri</i>	Namespace for Service Features
	VehicleFeatureNameSpace	0:1	<i>xsd:anyUri</i>	Namespace for Vehicle features
<i>Language</i>	Language	0:1	<i>xml:lang</i>	Default language.
<i>Location</i>	<i>a</i> WgsDecimalDegrees	0:1	<i>EmptyType</i>	Geospatial coordinates are given as WGS84 latitude and longitude, decimal degrees of arc.
	<i>b</i> GmlCoordinateFormat		<i>srsNameType</i>	Name of GML Coordinate format used for Geospatial points in responses.
<i>Units</i>	DistanceUnits	0:1	<i>xsd:normalized-String</i>	Units for <i>DistanceType</i> . Default is metres. +SIRI v2.0
	VelocityUnits	0:1	<i>xsd:normalized-String</i>	Units for <i>VelocityType</i> . Default is metres per second. +SIRI v2.0
<i>Temporal Span</i>	DataHorizon	0:1	<i>Positive-DurationType</i>	Maximum data horizon for requests
	RequestTimeout	0:1	<i>Positive-DurationType</i>	Default Timeout for requests
<i>Delivery Method</i>	DeliveryMethod	0:1	<i>DeliveryMethod-Enum</i>	Delivery interaction pattern to use to deliver data.
	MultipartDespatch	0:1	<i>xsd:boolean</i>	Whether multi-part delivery is allowed, i.e. the breaking up of updates into more than one delivery messages with a MoreData flag,
	ConfirmDelivery	0:1	<i>xsd:boolean</i>	Whether Consumers should issue an acknowledgement on successful receipt of a delivery. Default is 'false'.
<i>Resource Use</i>	MaximumNumberOf-Subscriptions	0:1	<i>xsd:positive-Integer</i>	Maximum number of subscriptions that can be sustained by a subscriber
<i>Prediction</i>	AllowedPredictors	0:1	<i>AllowedPredictor Enum</i>	Who may make a prediction. Documentation only. Default is 'anyone'.
	PredictionFunction	0:1	<i>xsd:string</i>	Allows a named to be given to the prediction function. Documentation only.
any	Extensions	0:1	any	Placeholder for user extensions.

6.1.3.2 *DeliveryMethod* — Allowed values

Allowed values for **DeliveryMethod** (DeliveryMethodEnumeration).

Table 10 — *DeliveryMethod* —Allowed Values (SIRI 2.0)

Value	Description
<i>fetch</i>	Deliveries are sent in two steps using notifications.
<i>direct</i>	Deliveries are sent direct without use of notifications.

6.1.3.3 *AllowedPredictors* — Allowed values

Allowed values for **AllowedPredictors** (AllowedPredictorEnumeration).

Table 11 — *AllowedPredictors* —Allowed Values (SIRI 2.0)

Value	Description
<i>avmsOnly</i>	Predictions may only be made by central AVMS prediction engine.
<i>anyone</i>	Downstream systems may interpolate or compute predictions.

6.1.4 Common Properties of ServiceRequest Messages — Element

All the individual SIRI Functional Service request message types, (for example **StopMonitoringRequest**, **VehicleMonitoringRequest**), etc., have a number of common elements – see Table 12.

Table 12 — SIRI Functional Service Common Request — Attributes

<i>xxxRequest</i>			<i>+Structure</i>	SIRI Functional service request for service xxx
<i>Attributes</i>	version	1:1	<i>VersionString</i>	Version Identifier of Functional Service, e.g. '1.0c'.
<i>Endpoint Properties</i>	RequestTimestamp	1:1	<i>xsd:dateTime</i>	Time of Request
	MessageIdentifier	0:1	<i>MessageQualifier</i>	Arbitrary unique reference to this message.
<i>Topic</i>	{Depends on Specific SIRI Functional Service – See Part 3 <i>xxxRequest</i> .}			
<i>Request Policy</i>	{Depends on Specific SIRI Functional Service– See Part 3 <i>xxxRequest</i> .}			
<i>any</i>	Extensions	0:1	<i>any</i>	Placeholder for user extensions.

6.1.5 ServiceRequest — Example

The following is a generalised example of a **ServiceRequest**. It includes a **ServiceRequestContext** context – normally the context would be implicit.

```
<Siri xmlns="http://www.siri.org.uk/siri" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.siri.org.uk/ http://www.siri.org.uk/schema\1.0\siri.xsd" version="1.0">
  <ServiceRequest>
    <!--=====GENERAL CONTEXT=====-->
    <ServiceRequestContext>
      <DataNameSpace>
        <RequestorParticipantRef>NADER</RequestorParticipantRef>
      </DataNameSpace>
      <Language>en</Language>
      <DataHorizon>P1Y2M3DT10H30M</DataHorizon>
      <RequestTimeout>P1Y2M3DT10H30M</RequestTimeout>
      <DeliveryMethod>direct</DeliveryMethod>
      <MultipartDespatch>true</MultipartDespatch>
      <ConfirmDelivery>false</ConfirmDelivery>
    </ServiceRequestContext>      <!--=====ENDPOINT REFERENCES=====-->
    <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
    <RequestorRef>NADER</RequestorRef>
    <xxxRequest version="1.0">>
.....
    </xxxRequest>
    <xxxRequest version="1.0">>
.....
    </xxxRequest>
  </ServiceRequest>
```

6.1.6 Access Controls on a Request

If the SIRI implementation supports the optional *AccessControl* capability, then requests are checked to see if the client is permitted to make the requested use of a SIRI Functional Service.

The access controls for a service may be specified by an Access Control Matrix comprising a set of permissions. The permission structures allowed for each service are included in the respective SIRI Functional Service schemas, and may be referenced by configuration documents. For example, the following example shows some permissions for using the SIRI Stop Monitoring service. It states that all participants may use the RequestResponse service, but only specific users may use Publish Subscribe. It also restricts public access to line A1; a separate permission specifically grants access to user NADER.

```
<StopMonitoringPermissions xmlns="http://www.siri.org.uk/siri" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.siri.org.uk/ siri_stopMonitoring_service.xsd">
  <!-- ==General permissions
    All participants may use RequestResponse.
    Only Specific users may use PublishSubscribe ===== -->
  <StopMonitoringPermission>
    <AllParticipants/>
    <GeneralCapabilities>
```

```

        <RequestResponse>true</RequestResponse>
        <PublishSubscribe>false</PublishSubscribe>
    </GeneralCapabilities>
    <OperatorPermissions>
        <AllowAll>true</AllowAll>
    </OperatorPermissions>
<!-- Public can't access line A1 -->
    <LinePermissions>
        <LinePermission>
            <Allow>false</Allow>
            <LineRef>A1</LineRef>
        </LinePermission>
    </LinePermissions>
    <StopMonitorPermissions>
        <StopMonitorPermission>
            <Allow>false</Allow>
            <MonitoringRef>Bar</MonitoringRef>
        </StopMonitorPermission>
    </StopMonitorPermissions>
</StopMonitoringPermission>
<!-- =====Permissions for NADER - May Also Pub Sub May see line A1 ===== -->
<StopMonitoringPermission>
    <ParticipantRef>NADER</ParticipantRef>
    <GeneralCapabilities>
        <PublishSubscribe>true</PublishSubscribe>
    </GeneralCapabilities>
<!-- Permissions for NADER - May see line A1 -->
    <LinePermissions>
        <LinePermission>
            <Allow>true</Allow>
            <LineRef>A1</LineRef>
            <DirectionRef>Foo</DirectionRef>
        </LinePermission>
    </LinePermissions>
</StopMonitoringPermission>
</StopMonitoringPermissions>

```

6.2 Receiving a Data Delivery

6.2.1 Introduction

Delivery responses are sent to the [*Consumer*] endpoint for the request. The delivery may be received as a single step *Direct Delivery* or as the last step of a *Fetches Delivery*.

Each Delivery comprises a general **ServiceDelivery** message, containing one or more SIRI Functional Service delivery responses, for example **ConnectionMonitoringFeederDelivery**, **StopMonitoringDelivery**.