EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

FINAL DRAFT
FprEN 15531-2

March 2015

ICS 35.240.60

Will supersede CEN/TS 15531-2:2007

English Version

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

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

Öffentlicher Verkehr - Serviceschnittstelle für Echtzeitinformationen bezogen auf Operationen im öffentlichen Verkehr - Teil 2: Kommunikationsstruktur

This draft European Standard is submitted to CEN members for formal vote. It has been drawn up by the Technical Committee CEN/TC 278.

If this draft becomes a European Standard, 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.

This draft European Standard was established by CEN 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, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Warning : This document is not a European Standard. It is distributed for review and comments. It is subject to change without notice and shall not be referred to as a European Standard.

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

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Contents

Page

Foreword

This document (FprEN 15531-2:2015) has been prepared by Technical Committee CEN/TC 278 “Intelligent transport systems”, the secretariat of which is held by NEN.

This document is currently submitted to the Formal Vote.

This document will supersede CEN/TS 15531-2:2007.

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

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 http://www.siri.org.uk/, along with available guidance on its use, example XML files, and case studies of national and local deployments.

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

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 European Standard 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, as well as between server and end-user devices like smartphones or web browsers. 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; (ii) 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 VAMS systems; (ii) providing standardised, well defined interfaces that can be used to deliver data to a wide variety of distribution channels.

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.

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, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

FprEN 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 FprEN 15531-1 apply.

4 Symbols and abbreviations

For the purposes of this document, the symbols and abbreviations given in FprEN 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 *Fetched 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 *Fetched 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 shall 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.

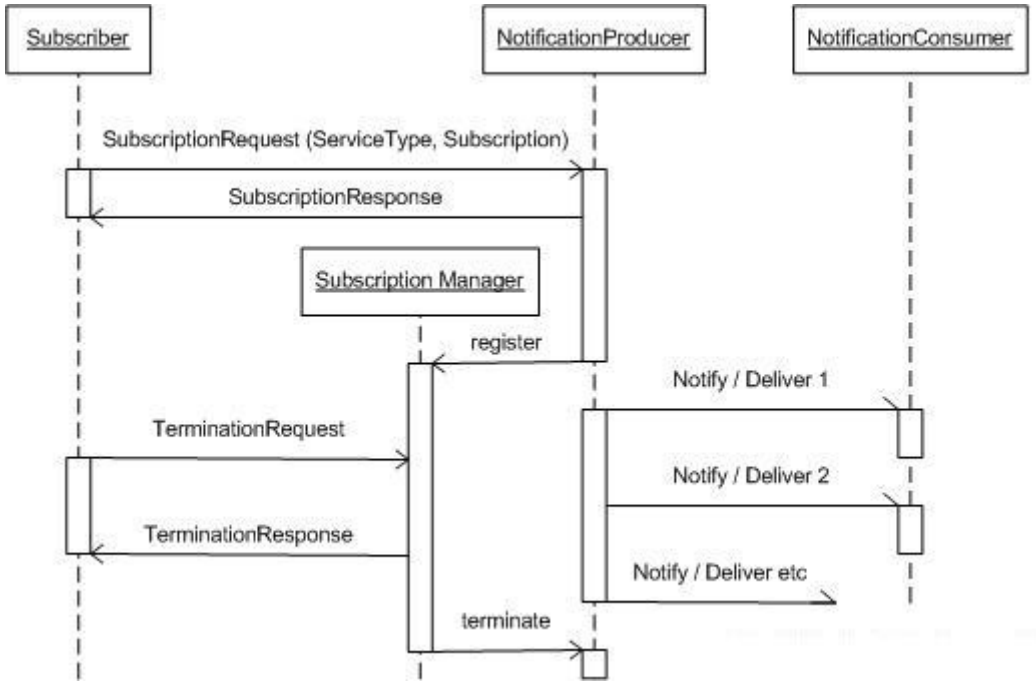

Figure 2 — Simple Publish/Subscribe Interaction

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.

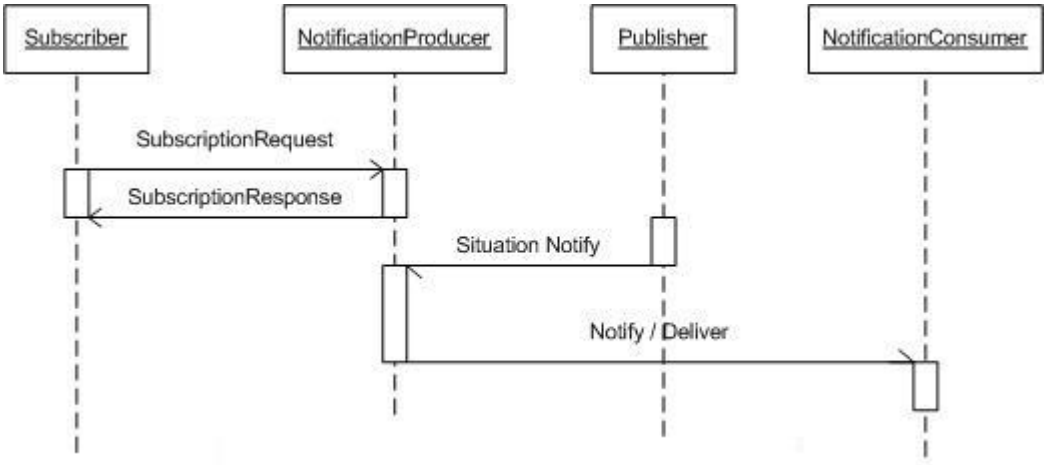

Figure 3 — Brokered Publish/Subscribe Interaction

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).

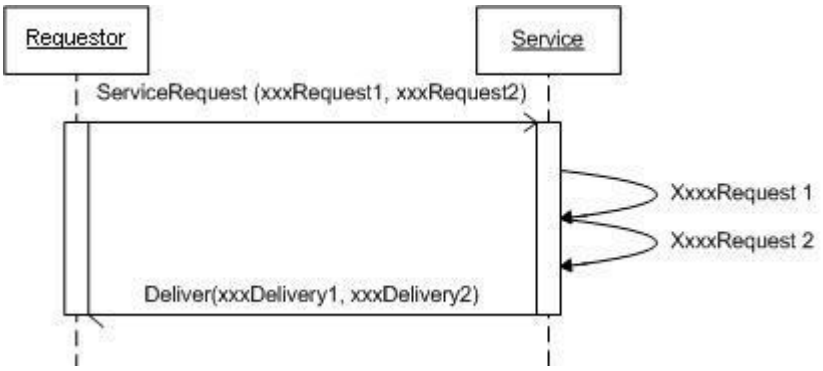

Figure 4 — Request/Response: Compound Requests

5.1.6 Publish/Subscribe – Compound Subscriptions

Multiple subscriptions to a single SIRI 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 clause on Mediation later below.

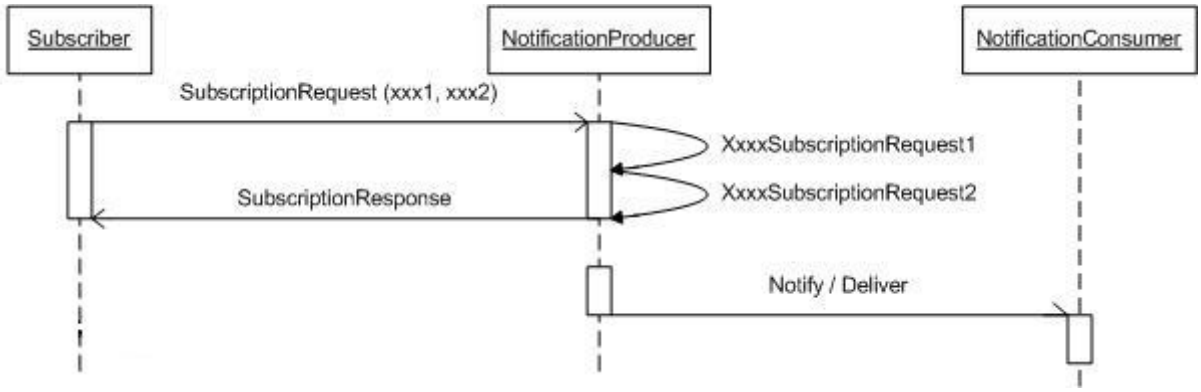

Figure 5 — Publish/Subscribe: Compound Subscriptions

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 SIRI supports two different message pattern variations for making a delivery, that in principle can be used interchangeably: these are; (i) *Direct Delivery*, and; (ii) *Fetched 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 SIRI 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.).

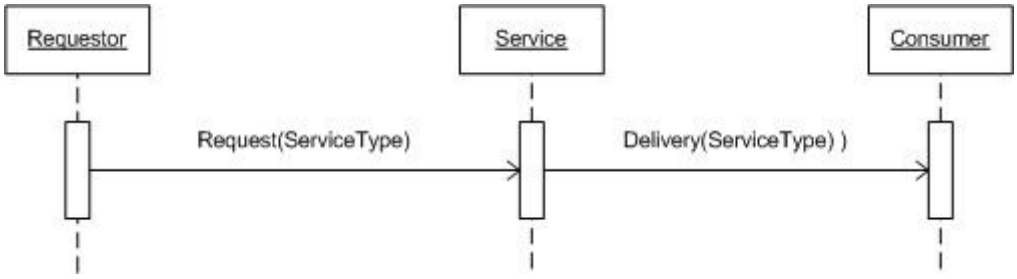

Figure 6 — One Step Direct Delivery

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 a 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.

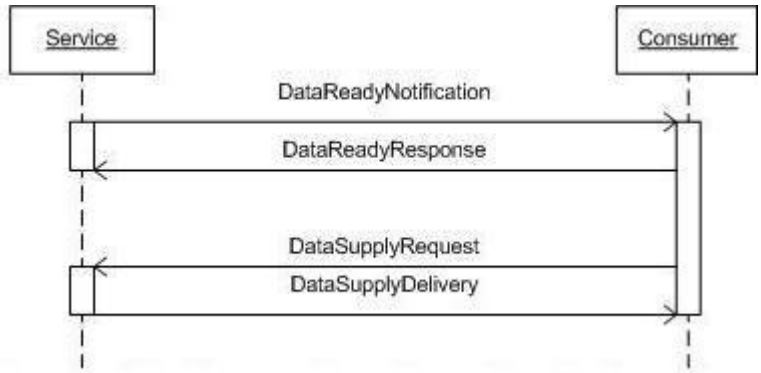

Figure 7 — Fetched Delivery

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

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

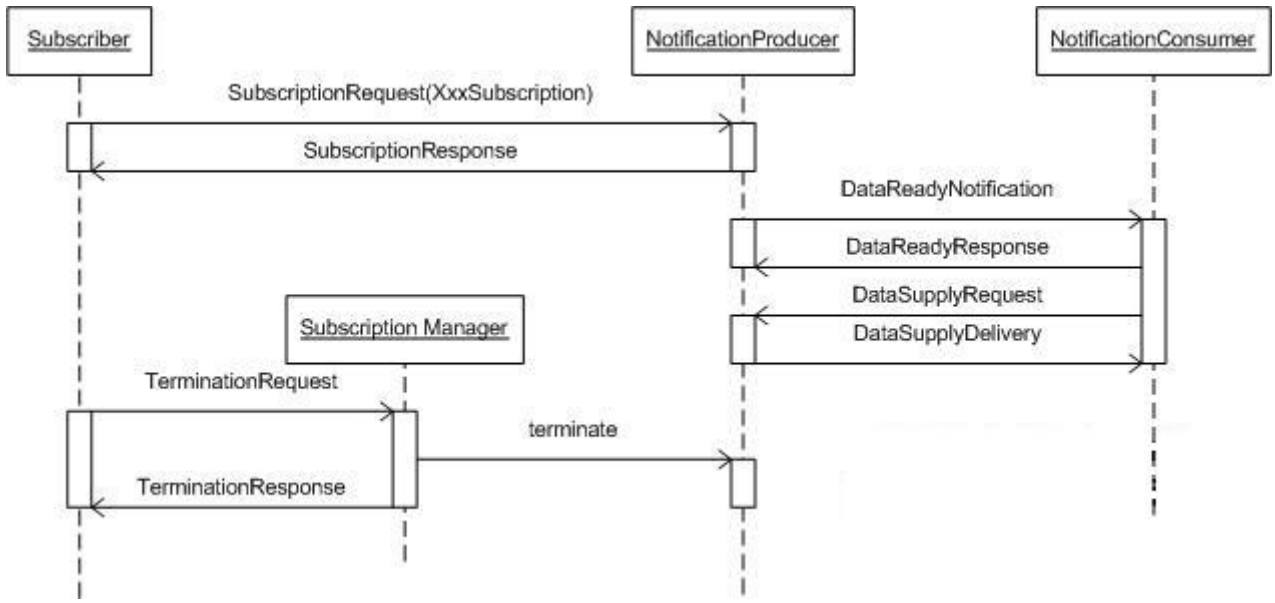

Figure 8 — Fetched Delivery for Publish/Subscribe

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

5.2.4 Data Horizon for Fetched Delivery

For *Fetched 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 shall 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.

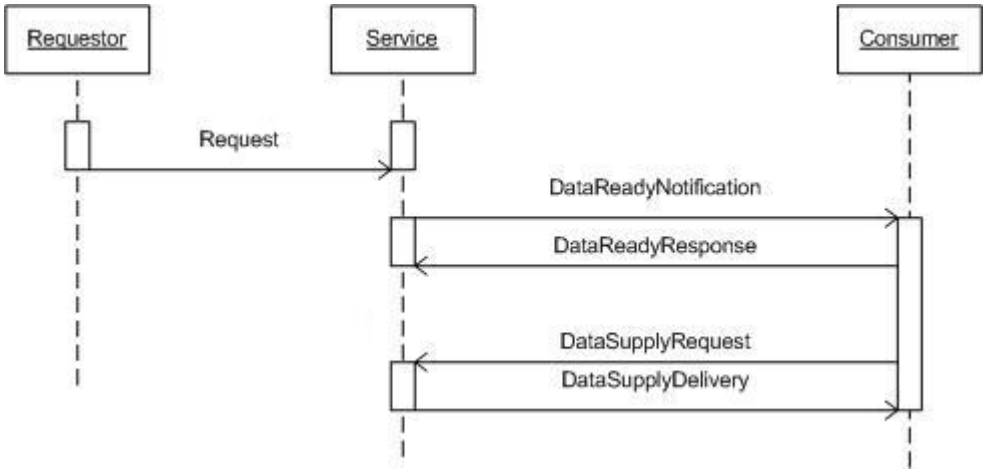

Figure 9 — Fetched Delivery for Request/Response

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.

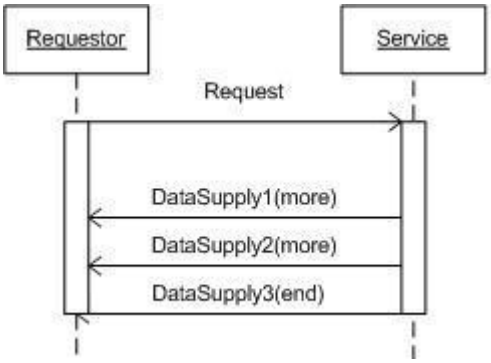

Figure 10 — Multipart Delivery

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 *Fetched 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"; i*n the last data supply response message, the ***MoreData*** element is set to "*false*".

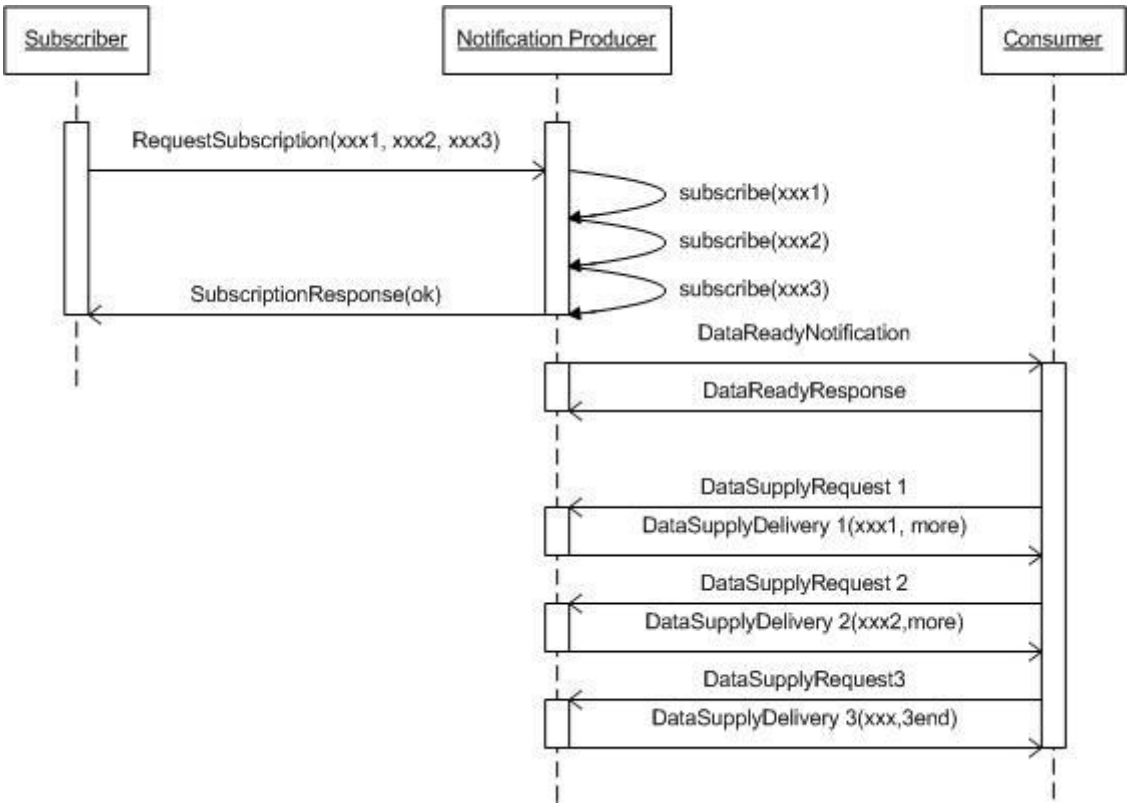

Figure 11 — Fetched Multipart Delivery

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 shall 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).

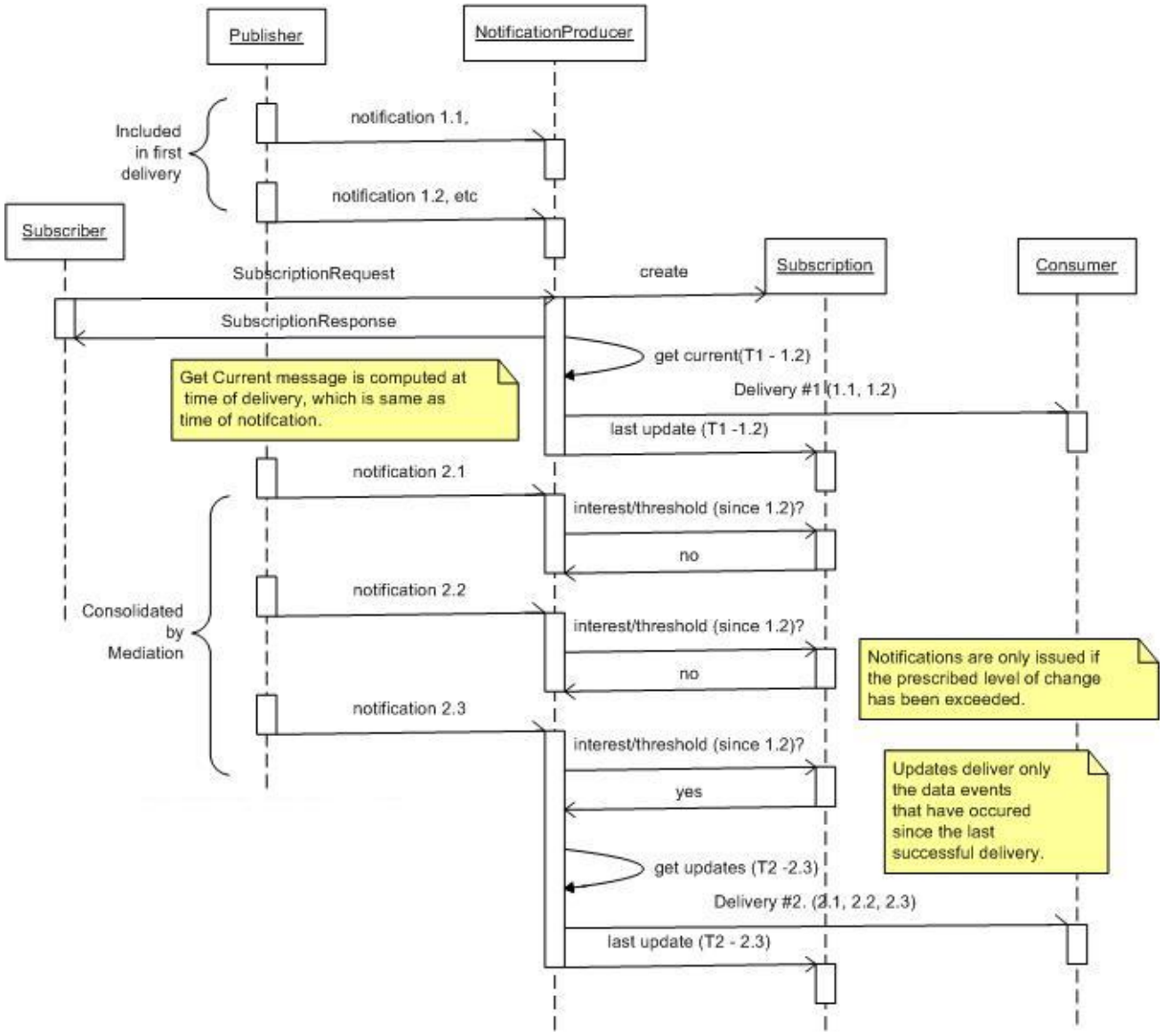

Figure 12 — Mediation: Update Tracking and sensitivity threshold for Direct Delivery

In a *Fetched 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 *Fetched 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.

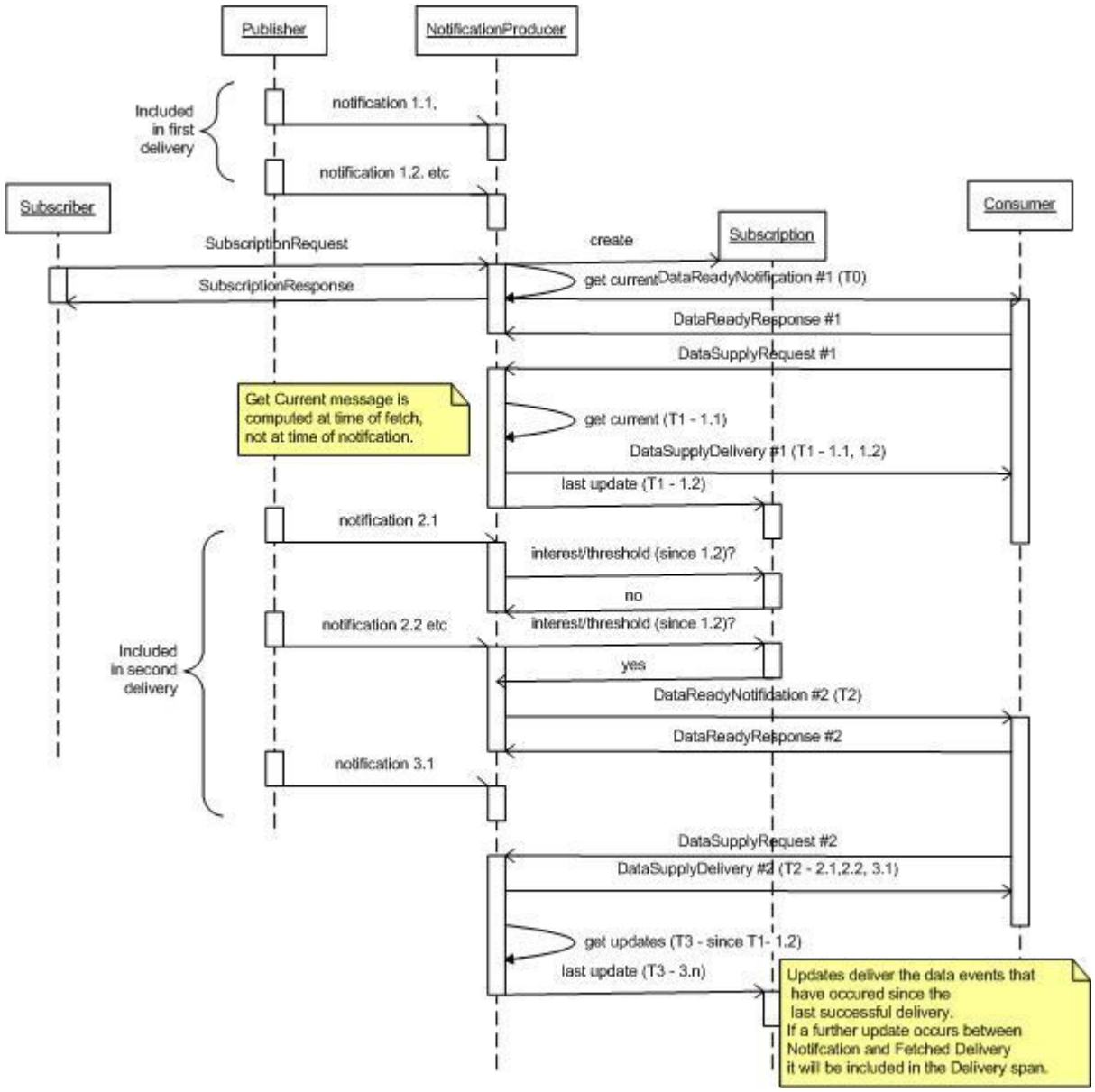

Figure 13 — Mediation: Handling Fetched Delivery Latencies

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 *Fetched 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.

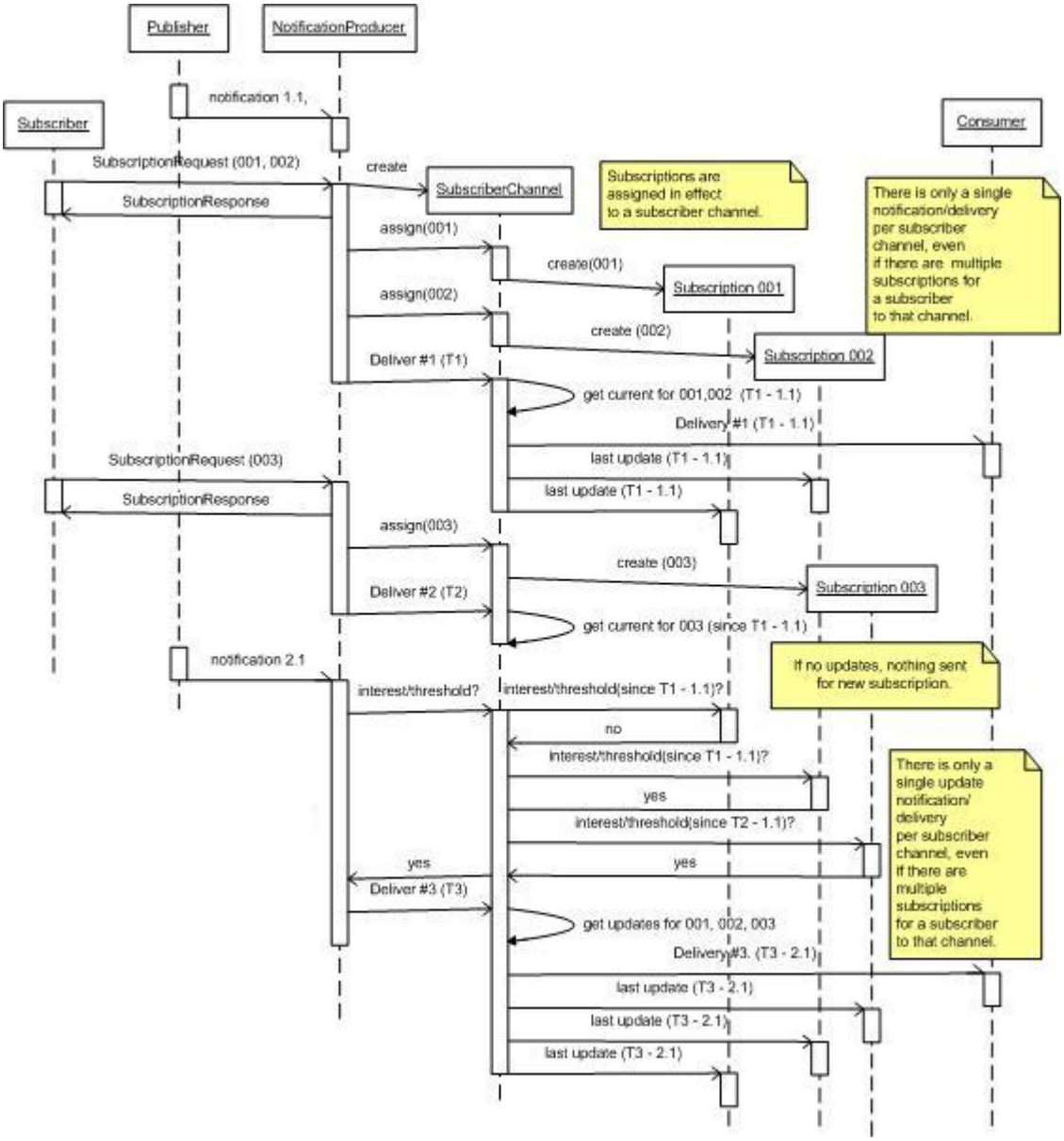

Figure 14 — Mediation: Subscription Filter

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 SubscriptionTerminatedNotifcation 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.

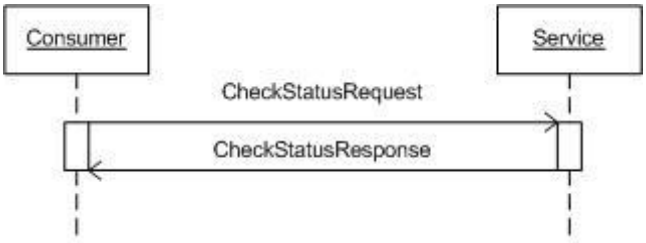

Figure 15 — Check Status — UML Sequence

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.

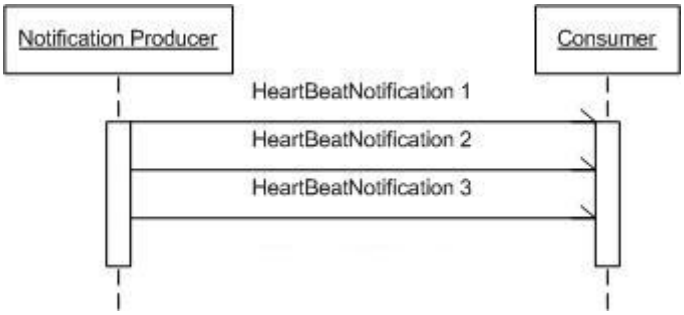

Figure 16 — Heartbeat Message — UML Sequence

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).

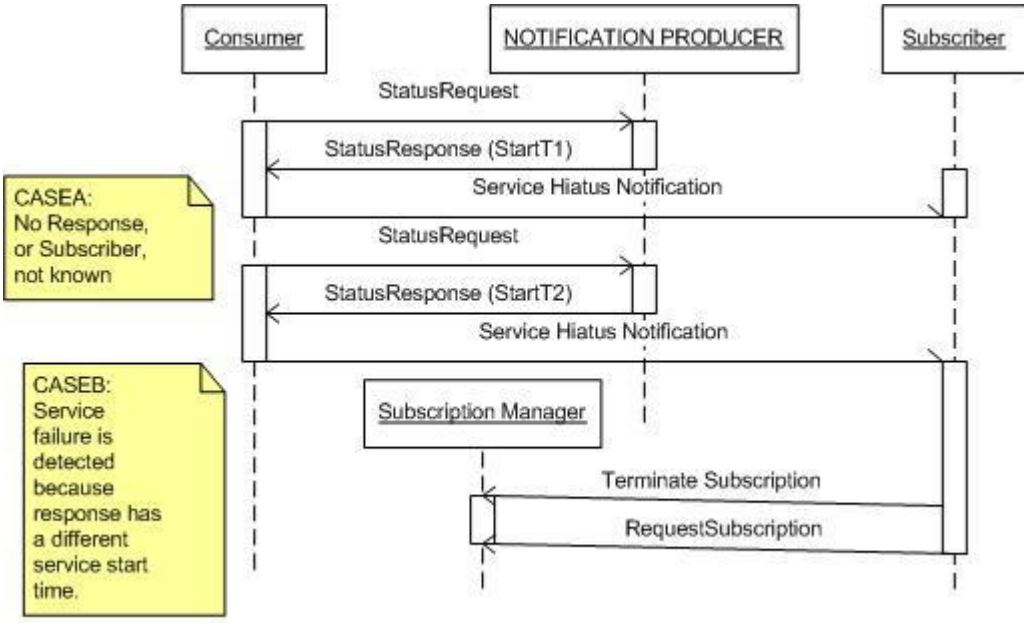

Figure 17 — Check Status with Recovery & Re-subscribe — UML Sequence

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.

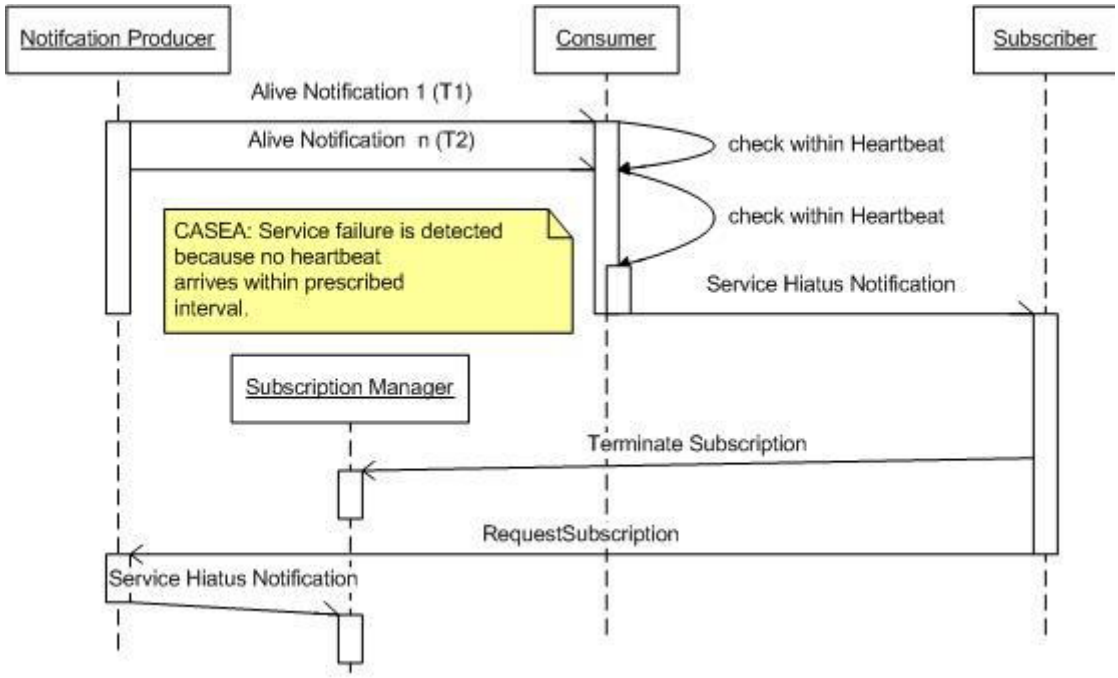

Figure 18 — Heartbeat Monitoring – Loss of Service — UML Sequence

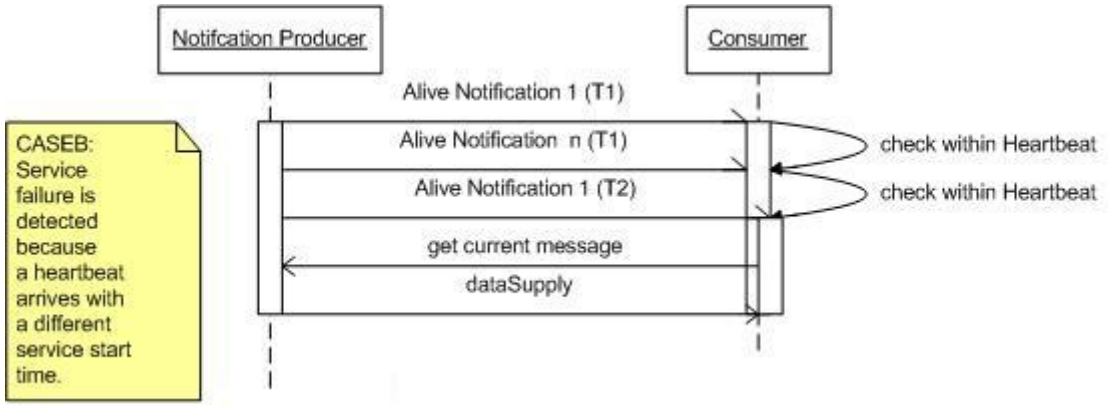

Figure 19 — Heartbeat Monitoring – Interruption of Service — UML Sequence

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 shall 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.

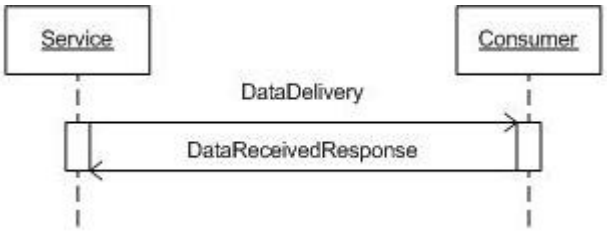

Figure 20 — Robust Direct Delivery — UML Sequence

With *Acknowledged Despatch,* reliable transmission is signalled though 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 preconfigured 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, organized 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	***RequestTimestamp***		Y	Timestamp
Endpoint – References	Address	***Address***	***Address***	Y	URL to respond
			ConsumerAddress	Y	
	Participant	***RequestorRef***	***SubscriberRef***	Y	Participant Reference, also known as Control Centre Code
		MessageIdentifier	***SubscriptionIdentifier***	Y	Issued by Subscriber
Content		***SiriServiceVersion***		Y	
	Topic Filters	***Topic***		N	Depends on message type
		TemporalWindow		N	Depends on message type
		DetailLevelToReturn		N	Depends on message type
		Language		Y	May be configured.
	Policy	***VolumeFilter***		Y	Number of items in Response. Depends on message type
	Delivery	***DeliveryMethod***		Y	Direct or Fetched. May be preconfigured.
		AllowMultipartDelivery		Y	May be preconfigured
Subscription	Lease	–	***InitialTerminationTime***	Y	Also known as. Lease Wanted.
	Notification Filters	–	***SensitivityFilter***	N	Also known as. Hysteresis. Depends on message type.
	Heartbeat Policy	–	***HeartbeatPolicy***	Y	May be preconfigured.

The general parameters which are common to all service request 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	*ValidityPeriod*	*PreviewInterval*		*PreviewInterval*		–	*PreviewInterval*
	–	–	*DepartureWindow*	*StartTime*	–	*ArrivalWindow*	*ConnectingTimeFilter*
	TimetableVersionRef		*MonitoringRef*		*VehicleMonitoringRef*	*ConnectionLinkRe*	
	OperatorRef			*OperatorRef*	–	–	–
	LineRef						
	DirectionRef						
	–	–	–	*DestinationRef*		–	–
	–	–	–	*StopVisitTypes*	*VehicleRef*	–	*DatedVehicleJourneyRef*
Request Policy				*MaximumStopVisits*	*MaximumVehicles*		
				MinimumStopVisits			
				MaximumTextLength			
	Language						
	IncludeTranslations						
	–	–	–	*DetailLevel*		–	–
	–	–	–	*MaxPreviousCalls*		–	–
	–	–	–	*MaxOnwardsCallss*		–	–
Subscription Policy	*IncrementalUpdates*	--	–	*IncrementalUpdates*		–	–
	–	–	–	*ChangeBeforeUpdates*		–	*ChangeBeforeUpdates*
	–	–	–	*UpdateInterval*		–	–

	General Message	Situation Exchange	Facility Monitoring
Topic (reference) content	*PreviewInterval*	*PreviewInterval*	*PreviewInterval*
	–	*StartTime*	*StartTime*
	InfoChannelRef	*SituationRef*	*FacilityRef*
	–	*Various Situation properties*	*Various properties*
Request Policy	*Language*		
	IncludeTranslations		

	–	***MaximumNumberOfSituationElements***	***MaximumNumberOfFacilityCOnditions***
Subscription Policy	–	***IncrementalUpdates***	***IncrementalUpdates***

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	***RequestTimeout***	Server not responding	408	1
	InvalidRequest	The server does not "understand" the request. The client should not repeat the request.	400	1
	Unauthorized	User name and password are required for the request, or credentials not satisfied	401	1
	Forbidden	The server "understands" the request, but cannot carry it out.	403	2
	NotFound	The requested URL was not found.	404	1
Distribution	***UnknownParticipant***	Recipient for a message to be distributed is unknown. +SIRI v2.0	601	2
	UnknownEndpoint	Endpoint to which a message is to be distributed is unknown. +SIRI v2.0	602	2
	EndpointDeniedAccess	Distribution message could not be delivered because not authorised. +SIRI v2.0	603	2
	EndpointNotAvailable	Recipient of a message to be distributed is not available. +SIRI v2.0	604	2
Access	***UnapprovedKey***	User authentication key is not approved. SIRI v2.0	610	1
Application Error	***VersionNotSupported***	Version of SIRI interface is not supported.	701	2
	CapabilityNotSupported	Service does not support the requested capability.	704	2
	ServiceNotAvailable	Functional service is not available to use (but it is still capable of giving this response).	710	2
	AccessNotAllowed	Requestor is not authorised to the service or data requested because a capability is not enabled.	720	2
	InvalidDataReferences	Request contains references to identifiers that are not known. +SIRI v2.0	730	2

Group	Condition	Description	Code	Sev
	BeyondDataHorizon	Request is for data outside of real-time data horizon.	732	2
	NoInfoForTopic	Valid request was made but service does not hold any data for the requested topic expression.	740	4
	ParametersIgnored	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
	UnknownExtensions	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
	UnknownSubscriber	Subscriber not found.	721	2
	UnknownSubscription	Subscription not found.	722	2
	AllowedResourceUsageExceeded	Valid request was made, but request would exceed the permitted resource usage of the client.	742	2
	OtherError	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	***SIRIVersion***	***VersionNotSupported***	Shall respond with same level.
Endpoint References	Endpoint	***ParticipantRef***	***UnknownSubscriber***	I.e. Control Centre not known.
	Identity	***SubscriptionIdentifier***	***UnknownSubscription***	
	Credentials	***Credentials***	***AccessNotAllowed***	Not known, or not authorised.
Common Content	Version	***SIRIServiceVersion***	***VersionNotSupported***	
	Topic Filters	***Topic***	***NoInfoForTopic.*** ***CapabilityNotSupported.***	Depends on message type.
		TemporalWindow	***BeyondDataHorizon***	Depends on message type.
		DetailLevelToReturn	***CapabilityNotSupported***	Depends on message type.
		Language	***CapabilityNotSupported***	
	Policy	***VolumeFilter***	***AllowedResourceUsageExceeded***	
	Delivery	***DeliveryMethod***	***CapabilityNotSupported***	
Subscription	Lease	***InitialTerminationTime***	***BeyondDataHorizon***	Depends on message type.
	Notification Filters	***SensitivityFilter***	***AllowedResourceUsageExceeded***	Depends on message type.
	Heartbeat Policy	***HeartbeatPolicy***	***CapabilityNotSupported***	

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 date 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 clauses 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	O	***StopDiscovery***	Returns Stops covered by service	SERVICE FRAME
	O	***LineDiscovery***	Returns Lines & direction covered by service	SERVICE FRAME
	O	***ProductCategoryDiscovery***	Returns Product categories used in SIRI	RESOURCE FRAME
	O	***ServiceFeatureDiscovery***	Returns Service features used in SIRI	RESOURCE FRAME
	O	***VehicleFeatureDiscovery***	Returns Vehicle features used in SIRI	RESOURCE FRAME
	O	***InfoChannelDiscovery***	Returns available SIRI GMS info channels	SIRI ONLY
	O	***ConnectionLinksDiscovery***	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 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	*Versioning*	*RequestChecking*
	R	*Capability*	*CapabilityChecking*
	O		*CapabilityDiscovery*
Interaction	At least one	*InteractionPattern*	*DirectRequest*
			Publish/Subscribe
	R	*Mediation*	*GetCurrent*
	R		*GetLastUpdate*
	R		*ChangeSensitivity*
	O		*Historic*
	R	*SubscriptionFilter*	*SingleFilter*
	O		*MultipleFilters*
	O	*DynamicContext*	--
	O	*AccessControl*	*ByCapability*
	O		*ByTopicValue*
Delivery	At least one	*DeliveryMethod*	*DirectDelivery*
			FetchedDelivery
	O	*ConfirmDelivery*	--

Feature	Required?	Capability Name	Sub-capability
	O	*VisitCountIsOrder*	--
	O	*MultipartDespatch*	--
Service Status	R	*CheckStatus*	--
	O	*Heartbeat*	--
Discovery	O	*Capabilities*	--
Message Transport	R	*MessageTransport*	*HttpPost*
	O		*SoapEnvelope*
	At least one	*Addresses*	*Implicit*
			Explicit
	O	*Compression*	*None* | *Gzip* | other

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	***ServiceRequest***	***ServiceDelivery***
Timetable	Production	***ProductionTimetableRequest***	***ProductionTimetableDelivery***
	Real-time	***EstimatedTimetableRequest***	***EstimatedTimetableDelivery***
Progress	Stop Timetable	***StopTimetableRequest***	***StopTimetableDelivery***
	Stop Monitoring	***StopMonitoringRequest***	***StopMonitoringDelivery***
		StopMonitoringMultipleRequest	***StopMonitoringDelivery***
	Vehicle Monitoring	***VehicleMonitoringRequest***	***VehicleMonitoringDelivery***
Interchange	Connection Timetable	***ConnectionTimetableRequest***	***ConnectionTimetableDelivery***
	Connection Monitoring	***ConnectionMonitoringRequest***	***ConnectionMonitoringFeederDelivery***
			ConnectionMonitoringDistributorDelivery
Info	General Message	***GeneralMessageRequest***	***GeneralMessageDelivery***
Amenities	Facility Monitoring	***FacilityMonitoringRequest***	***FacilityMonitoringDelivery***
Incidents	Situation	***SituationExchangeRequest***	***SituationExchangeDelivery***

Multiple Functional Service Requests may be included within a ***ServiceRequest***, but all requests shall 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			*+Structure*	Request from a Consumer to a Producer for immediate delivery of data. Answered with a ***ServiceDelivery***. (For *Fetched Delivery* this will be after a further ***DataReadyRequest***).
	ServiceRequestContext	0:1	*+Structure*	General request properties – typically configured rather than repeated on request.
log	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Timestamp on request.
Auth.	***AccountId***	0:1	*+Structure*	Account Identifier. May be used to attribute requests to a specific user account for authentication or reporting purposes +SIRI v2.0
	AccountKey	0:1	*+Structure*	Authentication key for request. May be used to authenticate the request to ensure the user is a registered client. +SIRI v2.0
Endpoint Properties	***Address***	0:1	*EndpointAddress*	Address to which response is to be sent: [***Notify***] endpoint. If omitted, this may also be determined from ***RequestorRef*** and preconfigured data, or the http request.
	RequestorRef	**1:1**	*→ParticipantCode*	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	*MessageQualifier*	Arbitrary identifier that may be given to message.

Delegator Endpoint		***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
	Concrete service subscription				If more than one, shall all be same type.
Payload	*a*	***ProductionTimetableRequest***	**-1:***	*+Structure*	See SIRI Part 3 – Production Timetable.
	b	***EstimatedTimetableRequest***		*+Structure*	See SIRI Part 3 – Estimated Timetable.
	c	***StopTimetableRequest***		*+Structure*	See SIRI Part 3 – Stop Timetable.
	d	***StopMonitoringRequest***		*+Structure*	See SIRI Part 3 – Stop Monitoring.
	e	***StopMonitoringMultipleRequest***		*+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 Endpoint 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.
	NotifyAddress	0:1	*EndpointAddress*	Address to which notifications requests are to be sent. If absent, same as *SubscriberAddress*.
	ConsumerAddress	0:1	*EndpointAddress*	Address to which data is to be sent. If absent, same as *NotifyAddress*.
Namespace	***DataNameSpaces***	0:1	*+Structure*	Scope for identifiers
NameSpace	***StopPointNameSpace***	0:1	*xsd:anyUrl*	Namespace for stop references.
	LineNameSpace	0:1	*xsd:anyUrl*	Namespace for LINE names and DIRECTIONs.
	ProductCategoryNameSpace	0:1	*xsd:anyUrl*	Namespace for product categories
	ServiceFeatureNameSpace	0:1	*xsd:anyUrl*	Namespace for Service Features
	VehicleFeatureNameSpace	0:1	*xsd:anyUrl*	Namespace for Vehicle features
Language	***Language***	0:1	*xml:lang*	Default language.
Location	*a* ***WgsDecimalDegrees***	0:1	*EmptyType*	Geospatial coordinates are given as WGS84 latitude and longitude, decimal degrees of arc.
	b ***GmlCoordinateFormat***		*srsNameType*	Name of GML Coordinate format used for Geospatial points in responses.
Units	***DistanceUnits***	0:1	*xsd:normalizedString*	Units for *DistanceType*. Default is metres. +SIRI v2.0
	VelocityUnits	0:1	*xsd:normalizedString*	Units for *VelocityType*. Default is metres per second. +SIRI v2.0
Temporal Span	***DataHorizon***	0:1	*PositiveDuration Type*	Maximum data horizon for requests
	RequestTimeout	0:1	*PositiveDuration Type*	Default Timeout for requests

Delivery Method	***DeliveryMethod***	0:1	*DeliveryMethodEnum*	Delivery interaction pattern to use to deliver data.
	MultipartDespatch	0:1	*xsd:boolean*	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	*xsd:boolean*	Whether Consumers should issue an acknowledgement on successful receipt of a delivery. Default is '*false'*.
Resource Use	***MaximumNumberOfSubscriptions***	0:1	*xsd:positiveInteger*	Maximum number of subscriptions that can be sustained by a subscriber
Prediction	***AllowedPredictors***	0:1	*AllowedPredictorEnum*	Who may make a prediction. Documentation only. Default is '*anyone'*.
	PredictionFunction	0:1	*xsd:string*	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
fetch	Deliveries are sent in two steps using notifications.
direct	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
avmsOnly	Predictions may only be made by central AVMS prediction engine.
anyone	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

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

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

Most Deliveries contain one or more instances of different types of 'Item' specific to the service, for example ***MonitoredStopVisit***, ***GeneralMessage*** etc. All items have a time of recording and an optional identifier; they may variously contain other model elements associated with the item type.

Table 13 — Delivery Content Elements

Delivery	Item (Recorded At)	Primary Association	Children
ProductionTimetableDelivery	***DatedTimetable***	***DatedVehicleJourney***	***DatedCall TargetedInterchange***
EstimatedTimetableDelivery	***EstimatedTimetableVersionFrame***	***EstimatedVehicleJourney***	***EstimatedCall***
		EstimatedServiceJourneyInterchange	
StopTimetableDelivery	***TimetabledStopVisit***	***TargetedVehicleJourney***	***TargetedCall***
	TimetabledStopVisitCancellation	–	–
StopMonitoringDelivery	***MonitoredStopVisit***	***MonitoredVehicleJourney***	***MonitoredCall***
	MonitoredStopVisitCancellation	–	–
	LineNotice	–	–
	LineNoticeCancellation	–	–
VehicleMonitoringDelivery	***VehicleActivity***	***MonitoredVehicleJourney***	***MonitoredCall***
	VehicleActivityCancellation	–	–
ConnectionTimetableDelivery	***TimetabledFeederArrival***	***ConnectingVehicleJourney (Feeder)***	–
	TimetabledFeederArrivalCancellation		–
ConnectionMonitoringFeederDelivery	***MonitoredFeederArrival***	***ConnectingVehicleJourney (Feeder)***	–
	MonitoredFeederArrivalCancellation		–
ConnectionMonitoringDistributorDelivery	***WaitProlongedDeparture***	***ConnectingVehicleJourney (Distributor)***	–
	StoppingPositionChangedDeparture		–
	DistributorDepartureCancellation		–
GeneralMessageDelivery	***GeneralMessage***	–	–
	GeneralMessageCancellation	–	–
FacilityMonitoringDelivery	***FacilityMonitoringDelivery***	*FacilityCondition*	
SituationExchange Delivery	***SituationExchange Delivery***	*PTSituation*	*+Structure*

6.2.2 ServiceDelivery

6.2.2.1 ServiceDelivery— Element

The ***ServiceDelivery*** contains any general parameters that are common to all delivery types.

Table 14 — ServiceDelivery— Attributes

ServiceDelivery				*+Structure*	Response from Producer to Consumer to deliver payload data. Either answers a direct ServiceRequest, or satisfies a subscription asynchronously. May be sent directly in one step, or be fetched in response to a Data Supply Request.
Attributes	***srsName***		0:1	*xsd:string*	Default GML coordinate format for any spatial points defined in response by Coordinates parameter.
Log	***ResponseTimestamp***		**1:1**	*xsd:dateTime*	Time individual response element was created.
Endpoint properties	***ProducerRef***		0:1	*→ParticipantCode*	Participant reference that identifies producer of data. May be available from context.
	Address		0:1	*EndpointAddress*	Address to which any acknowledgment should be sent. Only needed if ***ConfirmDelivery*** specified.
	ResponseMessageIdentifier		0:1	*MessageQualifier*	An arbitrary unique reference associated with the response which may be used to reference it.
	RequestMessageRef		0:1	*→MessageQualifier*	Reference to a unique message identifier associated with the request which gave rise to this response.
Delegator endpoint	***DelegatorAddress***		0:1	*EndpointAddress*	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 itself to be stateless.
	DelegatorRef		0:1	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Status	***Status***		0:1	*xsd:boolean*	Whether the complete request could be processed successfully or not. Default is *true*. If any of the individual requests within the delivery failed, should be set to *false*.
	ErrorCondition		0:1	*See below*	Description of any error or warning conditions that applies to the overall request. More Specific error conditions should be included in the error conditions attached to each functional service response that fails.
	a	***CapabilityNotSupportedError***	**1:1**	*+Error*	Capability not supported.
	b	***OtherError***		*+Error*	Error other than a well-defined category.
		Description	0:1	*→ErrorDescription*	Description of Error.
	MoreData		0:1	*xsd:boolean*	Whether there are more delivery messages making up this data supply group. Default is *false*. Optional SIRI Capability: *MultipartDespatch*.
Payload	*Concrete SIRI Service:*				One or more of a single type of the following:
	a	***ProductionTimetableDelivery***	0:*	*+Structure*	See SIRI Part 3 – Production Timetable.
	b	***EstimatedTimetableDelivery***		*+Structure*	See SIRI Part 3 – Estimated Timetable.
	c	***StopTimetableDelivery***		*+Structure*	See SIRI Part 3 – Stop Timetable.
	d	***StopMonitoringDelivery***		*+Structure*	See SIRI Part 3 – Stop Monitoring.
	e	***VehicleMonitoringDelivery***		*+Structure*	See SIRI Part 3 – Vehicle Monitoring.

	f	***ConnectionTimetableDelivery***		+*Structure*	See SIRI Part 3 – Connection Timetable.
	g	***ConnectionMonitoringFeederDelivery***		+*Structure*	See SIRI Part 3 – Connection Monitoring.
	h	***ConnectionMonitoringDistributorDelivery***		+*Structure*	See SIRI Part 3 – Connection Monitoring.
	i	***GeneralMessageDelivery***		+*Structure*	See SIRI Part 3 – General Message.
	j	***FacilityMonitoringDelivery***		+*Structure*	See SIRI Part 4 – Facility Monitoring. SIRI v1.3
	k	***SituationExchangeDelivery***		+*Structure*	See SIRI Part 5 – Situation Exchange. SIRI v1.3

6.2.2.2 Common Properties of SIRI Functional Service Delivery Messages

All the individual SIRI Functional Service delivery message types, (for example ***StopMonitoringDelivery***, ***VehicleMonitoringDelivery***), etc., have a number of common elements – see Table 15.

NOTE Cardinality with first member indicating "-1" stands for indicating a choice

Table 15 — SIRI Function Service xxxDelivery— Attributes

xxxDelivery				+*Structure*	Delivery for xxx Service
Log		***ResponseTimestamp***	**1:1**	*xsd:dateTime*	Time individual response element was created.
Endpoint properties		***RequestMessageRef***	0:1	→*MessageQualifier*	For direct requests, Identifier of request that this Delivery satisfies.
		SubscriberRef	0:1	→*ParticipantCode*	Required if Delivery is for a Subscription, Participant Reference of Subscriber.
		SubscriptionFilterRef	0:1	→*SubcriptionFilterCode*	Unique identifier of Subscription filter to which this subscription is assigned. If there is only a single filter, then can be omitted.
		SubscriptionRef	**1:1**	→*SubscriptionQualifier*	Required if Delivery is for a Subscription, Identifier of Subscription issued by Requestor. Unique within Subscriber (i.e. within ***ParticipantRef*** of Subscriber), and SIRI Functional Service type.
Delegation		***DelegatorAddress***	0:1	*Xsd:anyURI*	Address of original Consumer, i.e. requesting system to which delegating response is to be returned. +SIRI 2.0
		DelegatorRef	0:1	→*ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Status		***Status***	0:1	*xsd:boolean*	Whether the complete request could be processed successfully or not. Default is true. If any of the individual requests within the delivery failed, should be set to *false*.
		ErrorCondition	0:1	+*Structure*	Description of any error or warning conditions that apply to the specific functional request or response.
				choice	One of the following Error codes.
	A	***ServiceNotAvailableError***			Error: Functional service is not available to use (but it is still capable of giving this response).
	b	***CapabilityNotSupportedError***	**-1:1**	+ *Error*	Error: Capability not supported.
	c	***AccessNotAllowedError***		+*Error*	Error: Requestor is not authorised to the service or data requested.

	d	***InvalidDataReferencesError***		*+Error*	Error: Request contains references to identifiers that are not known. +SIRI v2.0.
	E	***BeyondDataHorizon***		*+Error*	Error: Data period or subscription period is outside of period covered by service. +SIRI v2.0.
	f	***NoInfoForTopicError***		*+Error*	Error: Valid request was made but service does not hold any data for the requested topic expression.
	G	***ParametersIgnoredError***		*+Error*	Error: Request contained parameters that were not supported by the producer. A response has been provided but some parameters have been ignored. +SIRI v2.0.
	H	***UnknownExtensionsError***		*+Error*	Error: 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.
	i	***AllowedResourceUsageExceededError***		*+Error*	Error: Valid request was made but request would exceed the permitted resource usage of the client.
	j	***OtherError***		*+Error*	Error other than a well-defined category.
		Description	0:1	*→ErrorDescription*	Description of Error.
	ValidUntil		0:1	*xsd:dateTime*	End of data horizon of the data producer.
	ShortestPossibleCycle		0:1	*PositiveDurationType*	Minimum interval at which updates can be sent.
	DefaultLanguage			*Xsd:language*	Default language for text elements.
Payload	{Content Specific to SIRI Functional Service type. See Part 3.}				
any	***Extensions***		0:1	*any*	Placeholder for user extensions.

6.2.2.3 ServiceDelivery — Example

For specific examples, see the individual concrete SIRI Functional Services.

The following is a generic example of a ***ServiceDelivery*** in the case of a direct delivery.

```
<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">
    <!- =======RESPONSE============================== ->
<ServiceDelivery>
       <!-=======HEADER================================ ->
       <RequestRef>2004-12-17T09:30:46-05:00</RequestRef>
       <RequestorRef>NADER</RequestorRef>
       <Status>true</Status>
       <MoreData>false</MoreData>
       <!-=======FUNCTIONAL SERVICE HEADER================= ->
       < XxxDelivery version="1.0">
          <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
          <Status>true</Status>
          <ValidUntil>2004-12-17T09:30:47-05:00</ValidUntil>
          <ShortestPossibleCycle>PT3M</ShortestPossibleCycle>
          <!-==== FUNCTIONAL SERVICE PAYLOAD ======== ->
          <Xxx content>
```

```
……
        </xxxDelivery>
    </ServiceDelivery>
```

The following is an example of a ***ServiceDelivery*** in the case of a subscription interface.

```
<ServiceDelivery>
        <!-=======HEADER========================================== ->
        <RequestorRef>NADER</RequestorRef>
        <RequestRef>2004-12-17T09:30:46-05:00</RequestRef>
        <Status>true</Status>
        <MoreData>false</MoreData>
        <!-======= FUNCTIONAL SERVICE HEADER ====================== ->
        < XxxDelivery version="1.0">
            <ResponseTimestamp>2004-12-17T09:30:47-05:00</ ResponseTimestamp>
            <SubscriberRef>NADER</SubscriberRef>
            <SubscriptionRef> MYSUB457</SubscriptionRef>
            <Status>true</Status>
            <ValidUntil>2004-12-17T09:30:47-05:00</ValidUntil>
            <ShortestPossibleCycle>PT3M</ShortestPossibleCycle>
            <!-==== FUNCTIONAL SERVICE PAYLOAD ====================== ->
            <Xxx content>
……
        </xxxDelivery>
</ServiceDelivery>
```

7 Subscriptions

7.1 Setting up Subscriptions

7.1.1 Introduction

A Subscription is created by sending a ***SubscriptionRequest*** to the Notification Producer of the desired SIRI Functional Service type, as located by the [*Subscription*] endpoint. The Notification Producer service responds with a ***SubscriptionResponse*** message that confirms the granting of the subscription, or provides an error condition that indicates why the subscription could not be created. The Notification Producer then creates the accepted subscriptions and starts to supply them with data.

The specific SIRI Functional Service Subscription Requests are wrapped within a general ***SubscriptionRequest*** element, and any corresponding delivery for the Functional Service is similarly wrapped within a ***ServiceDelivery*** element. There is a different SIRI Subscription Request message type for each different SIRI Functional Service type, and also a distinct SIRI Subscription Delivery message type for each response type (see Table 16).

Table 16 — SIRI Request and Delivery Types

SIRI Functional Service	***SubscriptionRequest***	***ServiceDelivery***	**Publishes**
Production Timetable	***ProductionTimetableSubscriptionRequest***	***ProductionTimetableDelivery***	Timetables
Estimated Timetable	***EstimatedTimetableSubscriptionRequest***	***EstimatedTimetableDelivery***	Timetable Changes
Stop Timetable	***StopTimetableSubscriptionRequest***	***StopTimetableDelivery***	Stop Timetable
Stop Monitoring	***StopMonitoringSubscriptionRequest***	***StopMonitoringDelivery***	Visits to stop
Vehicle Monitoring	***VehicleMonitoringSubscriptionRequest***	***VehicleMonitoringDelivery***	Vehicle Movements
Connection Timetable	***ConnectionTimetableSubscriptionRequest***	***ConnectionTimetableDelivery***	Connections
Connection Monitoring	***ConnectionMonitoringSubscriptionRequest***	***ConnectionMonitoringFeederDelivery*** ***ConnectionMonitoringDistributorDelivery***	Connection changes
General Message	***GeneralMessageSubscriptionRequest***	***GeneralMessageDelivery***	Travel News
Facility Monitoring	***FacilityMonitoring SubscriptionRequest***	***FacilityMonitoringDelivery***	Amenity Status
Situation Exchange	***SituationExchange SubscriptionRequest***	***SituationExchange Delivery***	Incidents

Multiple functional service subscriptions may be created by a single ***SubscriptionRequest***. The ***SubscriptionRequest*** is a container for one or more individual Functional Service subscription requests. This is indicated by ***xxxSubscriptionRequest*** subelements, where 'Xxx' indicates the individual SIRI Functional Service being requested. A given ***SubscriptionRequest*** can only contain individual service requests for a single SIRI Functional Service type.

If the ***SubscriptionRequest*** is compound, that is, for multiple subscriptions, the response will also be compound, that is, contain individual ***ResponseStatus*** subelements for each service subscription that is successfully created. If any individual Functional Service cannot be created, the ***SubscriptionResponse*** should contain a separate ***ResponseStatus*** with an error description for each failed service subscription request, and the overall response status should be set to false.

The ***SubscriptionIdentifier*** is an Endpoint Property that uniquely identifies each individual functional service subscription. The subscription identifier is made up of two parts: the Participant Reference, and a ***SubscriptionIdentifier*** which will be unique within the Subscriber's Participant Reference and SIRI Functional Service type. It will be included in subsequent deliveries to the Consumer, and is also used to manage the Subscription at a later date.

If a Consumer Address is not supplied on the individual subscription request, it will be obtained from configuration details that associate the Participant Reference with a specific Address. See 10.2 on Logical Endpoints.

If a Service Subscription Request is submitted with a Subscriber reference of a Subscription that already exists, *the existing subscription is deleted* and a new subscription created in its place. Logically this is the same as sending a ***TerminateSubscription*** message, followed by a new ***SubscriptionRequest*** message to create a new subscription.

7.1.2 SubscriptionRequest

7.1.2.1 SubscriptionRequest — Element

The Subscription request is sent to the [*Subscriber*] endpoint of a SIRI Functional Service.

When requesting each new Functional Service Subscription, a subscriber can specify both a [*Subscriber*] endpoint address and a [*Notify*] endpoint that identifies a separate Consumer address. Confirmation that the subscription has been created goes to the [*Subscriber*] endpoint. The [*Notify*] endpoint determines the internet address where data ready notifications should be sent.

Table 17 shows the parameters that may be specified on a ***SubscriptionRequest***, either as parameters, or through the request context.

Table 17 — SubscriptionRequest — Attributes

SubscriptionRequest			*+Structure*	Request from Subscriber to Producer for a subscription. Answered with a **SubscriptionResponse**.
Log	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Time of subscription request issued by Requestor.
Auth.	***AccountId***	0:1	*+Structure*	Account Identifier. May be used to attribute requests to a specific user account for authentication or reporting purposes +SIRI v2.0
	AccountKey	0:1	*+Structure*	Authentication key for request. May be used to authenticate the request to ensure the user is a registered client. +SIRI v2.0
Endpoint properties	***Address***	0:1	*EndpointAddress*	Address to which subscription response is to be sent: [Subscriber] endpoint. This may also be determined from ***RequestorRef*** and preconfigured data.
	RequestorRef	**1:1**	*→ParticipantCode*	Identifier of requestor –Identifier of Participant.
	MessageIdentifier	0:1	*MessageQualifier*	Optional Arbitrary unique reference to the Subscription Request message.
	DelegatorAddress	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
	ConsumerAddress	0:1	*EndpointAddress*	Address to which notification is to be sent: if different from Address: [Notify] endpoint. This may also be determined from RequestorRef and preconfigured data.
	SubscriptionFilterIdentifier	0:1	*xsd:NMTOKEN*	Identifier of Subscriber Channel with which this subscription is to be aggregated for purposes of notification and delivery. If absent, use the default channel. If present, use any existing channel with that identifier, if none found, create a new one. Optional SIRI feature.
Policy	***SubscriptionContext***	0:1	*+Structure*	General subscription parameters.

Payload	*Concrete service subscription:*			*choice*	SIRI Functional Service Subscriptions. For a given ***SubscriptionRequest***, shall all be of the same type.
	a	***ProductionTimetableSubscriptionRequest***	**–1:***	+*Structure*	See SIRI Part 3 - Production Timetable.
	b	***EstimatedTimetableSubscriptionRequest***		+*Structure*	See SIRI Part 3- Estimated Timetable.
	c	***StopTimetableSubscriptionRequest***		+*Structure*	See SIRI Part 3 - Stop Timetable.
	d	***StopMonitoringSubscriptionRequest***		+*Structure*	See SIRI Part 3 - Stop Monitoring.
	e	***VehicleMonitoringSubscriptionRequest***		+*Structure*	See SIRI Part 3 - Vehicle Monitoring.
	f	***ConnectionTimetableSubscriptionRequest***		+*Structure*	See SIRI Part 3 - Connection Timetable.
	g	***ConnectionMonitoringSubscriptionRequest***		+*Structure*	See SIRI Part 3 - Connection Monitoring.
	h	***GeneralMessageSubscriptionRequest***		+*Structure*	See SIRI Part 3 – General Message.
	i	***FacilityMonitoringSubscriptionRequest***		+*Structure*	See SIRI Part 4 - Facility Monitoring. SIRI v1.3
	j	***SituationExchangeSubscriptionRequest***		+*Structure*	See SIRI Part 5 – Situation Exchange. SIRI v1.3

7.1.2.2 SubscriptionRequestContext — Element

The ***SubscriptionRequestContext*** contains any general configuration parameters that are common to all subscription request types and that may be preconfigured, rather than being repeated on individual requests. A primary role of the ***SubscriptionRequestContext*** is for documentation: it records the important properties of a SIRI implementation. Normally the context is fixed for the configuration. If the implementation supports a DynamicContext, then a context may be attached to individual requests and to override those properties which the implementation allows to be set dynamically. The SIRI Capability Discovery Request can be used to retrieve the current ***SubscriptionRequestContext***

Table 18 shows the common parameters that may be specified in a ***SubscriptionRequestContext***.

Table 18 — SubscriptionContext — Attributes

SubscriptionRequestContext			+*Structure*	General values that apply to subscription. Usually set by configuration.
Payload	***HeartbeatInterval***	0:1	*PositiveDurationType*	Interval for heartbeat.

7.1.2.3 The Common Properties of SIRI Functional Service Subscription Requests

All the individual SIRI Functional Service subscription request message types, (for example ***StopMonitoringSubscriptionRequest***, ***VehicleMonitoringSubscriptionRequest***, etc.), have a number of common elements – see Table 19.

Table 19 — SIRI Functional Service Common Subscription — Attributes

xxxSubscription			*+Structure*	Request for a subscription to a SIRI Functional Service
Identity	***SubscriberRef***	0:1	*→ParticipantCode*	Participant Identifier of Subscriber. Normally this will be given by context, i.e. be the same as on the Subscription Request.
	SubscriptionIdentifier	**1:1**	*SubscriptionQualifier*	Identifier to be given to Subscription. Unique within SubscriberRef and service type.
Lease	***InitialTerminationTime***	**1:1**	*xsd:dateTIme*	Requested end time for subscription. See SIRI subscription procedures.
Request	***xxxRequest***	**1:1**	*+Structure*	Request for the Specific SIRI Functional Service.
Policy	{Depends on Specific SIRI Functional Service. See Part 3.}			
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

7.1.2.4 SubscriptionRequest — Example

The following is a general example of a ***SubscriptionRequest***. For specific examples, see the individual concrete SIRI Functional Services.

```
<SubscriptionRequest>
       <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
       <RequestorRef> NADER</RequestorRef>
       <XxxSubscriptionRequest>
          <SubscriberRef> NADER</SubscriberRef>
          <SubscriptionIdentifier> MYSUB457</SubscriptionIdentifier>
          <InitialTerminationTime>2004-12-17T15:30:47-
05:00</InitialTerminationTime>
          <xxxRequest version="1.0">>
              …….
          </xxxRequest version="1.0">>
          <ChangeBeforeUpdates>PT2M</ChangeBeforeUpdates>
          <IncrementalUpdates>true</IncrementalUpdates>
       </xxxSubscriptionRequest>
    </SubscriptionRequest>
</Siri>
```

7.1.3 SubscriptionResponse

7.1.3.1 SubscriptionResponse — Element

After the Notification Producer has received the subscription request, it acknowledges with a single ***SubscriptionResponse*** message: this contains a separate ***ResponseStatus*** instance for each individual subscription processed. The response is sent to the [*Subscriber*] endpoint of the request. The response may include an ***Address*** element which indicates the actual subscription manager endpoint.

Table 20 — SubscriptionResponse — Attributes

SubscriptionResponse			*+Structure*	Response from Producer to Consumer to inform whether subscriptions have been created. Answers a previous ***SubscriptionRequest***.
Log	***ResponseTimestamp***	**1:1**	*xsd:dateTime*	Time individual response element was created.
Endpoint	***Address***	0:1	*EndpointAddress*	Address for further communication about subscription, i.e. [Subscription] endpoint for Subscription manager.

properties	***ResponderRef***	0:1	→*ParticipantCode*	Participant reference that identifies responder.
	RequestMessageRef	0:1	→*MessageQualifier*	Reference to an arbitrary unique reference associated with the request which gave rise to this response.
Delegation	***DelegatorAddress***	0:1	*EndpointAddress*	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	→*ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Payload	***ResponseStatus***	**1:***	+*Structure*	Status information about the request, or else error conditions.
	SubscriptionManagerAddress	0:1	*EndpointAddress*	Endpoint address of subscription manager if different from that of the Producer or known default.
	ServiceStartedTime	0:1	*xsd:dateTime*	Time at which service providing the subscription was last started. Can be used to detect restarts. If absent, unknown.
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

The ***ResponseStatus*** element supplies information on whether an individual SIRI Functional Service subscription request could be processed. If a subscription is granted, the response can include information on the maximum possible update rate of the data producing system, as well as the available data horizon. If the subscription could not be created, it should contain an appropriate error condition. The Subscriber may then resubmit a corrected request to create just these subscriptions.

Table 21 — ResponseStatus — Attributes

ResponseStatus				*+Structure*	Contains information about the processing of an individual service subscription – either success info or an error condition.
Log		***ResponseTimestamp***	0:1	*xsd:dateTime*	Time individual response element was created.
Endpoint		***RequestMessageRef***	0:1	*→MessageQualifier*	Reference to a unique message identifier associated with the request which gave rise to this response.
		SubscriberRef	0:1	*→ParticipantCode*	Unique identifier of Subscriber – Participant reference.
		SubscriptionRef	**1:1**	*→SubscriptionQualifier*	Unique identifier of subscription within Service and Subscriber.
Payload		***Status***	0:1	*xsd:boolean*	Whether the request could be processed successfully or not. Default is true.
		ErrorCondition	0:1	*+Structure*	Error conditions that apply to a service request. Choice of one of the following:
	a	***CapabilityNotSupportedError***	**–1:1**	*+Error*	Capability not supported.
	b	***AccessNotAllowedError***		*+Error*	Requestor is not authorised to the service or data requested.
	c	***NoInfoForTopicError***		*+Error*	Valid request was made but service does not hold any data for the requested topic expression.
	d	***AllowedResourceUsageExceededError***		*+Error*	Valid request was made but request would exceed the permitted resource usage of the client.
	e	***OtherError***		*+Error*	Error other than a well-defined category.
		Description	0:1	*→ErrorDescription*	Description of the error in plain text.
Info		***ValidUntil***	0:1	*xsd:dateTime*	End of the data horizon of the data producer; omitted if the request lies completely within the data horizon.
		ShortestPossibleCycle	0:1	*PositiveDurationType*	Minimum separation between two updates depends on the processing cycle of the AVMS.
any		***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

7.1.3.2 SubscriptionResponse — Example

The following is an example of a ***SubscriptionResponse***.

```
<!- =======RESPONSE================================== ->
<SubscriptionResponse>
   <ResponseTimestamp>2004-12-17T09:30:47-05:00</ResponseTimestamp>
   <ResponderRef>KUBRICK</ResponderRef>
   <!- Good request ->
   <ResponseStatus>
      <ResponseTimestamp>2004-12-17T09:30:47-05:01</ResponseTimestamp>
      <SubscriptionRef>0003456</SubscriptionRef>
      <Status>true</Status>
      <ValidUntil>2004-12-17T09:30:47-05:00</ValidUntil>
      <ShortestPossibleCycle>P1Y2M3DT10H30M</ShortestPossibleCycle>
   </ResponseStatus>
   <!- Bad request ->
   <ResponseStatus>
      <ResponseTimestamp>2004-12-17T09:30:47-05:02</ResponseTimestamp>
```

```
            <SubscriptionRef>0003457</SubscriptionRef>
            <Status>false</Status>
            <ErrorCondition>
               <NoInfoForTopicError/>
            </ErrorCondition>
         </ResponseStatus>
   </SubscriptionResponse>
</Siri>
```

7.2 Subscription Validity

A Subscription Request includes a request for an Initial Termination Time, representing the desired lease of the Subscription as an absolute UTC end time. The time stamp should be selected so that it lies beyond the last potential data registration time. The subscription will only be granted if the lease can be met, otherwise an error will be returned.

The lease defines how long the server shall store and manage the subscription. Subscribers should not ask for leases that are longer than necessary, as each subscription takes up system resources. Subscriptions can have validities that exceed the limits of a single operating day.

7.3 Terminating Subscriptions

7.3.1 Introduction

Subscriptions are automatically deleted by the Notification Producer Service after expiration of their validity.

A Subscriber may terminate a subscription before expiration of validity by sending a ***TerminateSubscriptionRequest*** subscription request to the Subscription Manager. The ***TerminateSubscriptionRequest***, contains one or more sub-elements of type ***SubscriptionRequestRef*** (7.1), each of which identifies a subscription to be ended.

In WS-PubSub subscription management is done through a different interface to that of the Notification Producer: although the Notification Producer is the factory for new subscriptions, it does not manage them. Changes to subscriptions are directed at a distinct endpoint ([*Subscription*]: the subscription manager) with a separate interface, allowing a proper separation of concern. In SIRI a separate ***TerminateSubscriptionRequest*** message is used therefore for deleting messages, rather than making subscription changes a type of Functional Service request.

In systems that use data delivery acknowledgement a Notification Producer may terminate a subscription if it fails to receive a delivery acknowledgement within a specified timeout. In this case the Notification Producer may send a ***TerminateSubscriptionResponse*** to the Consumer to inform it of its eviction.

It is normally the Subscribers responsibility to monitor the service stats through heartbeats. However if a Producer service has to terminate it may additionally issue a SubscriptionTerminatedNotifcation to alert subscribers that they need to restart..

7.3.2 The TerminateSubscriptionRequest

7.3.2.1 TerminateSubscriptionRequest — Element

A Subscriber terminates its subscriptions to a service by sending a ***TerminateSubscriptionRequest*** to the Subscription Manager, i.e. the [*Subscriber*] endpoint of the SIRI functional service. A ***TerminateSubscriptionRequest*** may contain either a specific subscription identifier, or a special value of ***All***, indicating that all subscriptions for the subscriber should be terminated.

Table 22 — TerminateSubscriptionRequest — Attributes

TerminateSubscriptionRequest			*+Structure*	Request from Subscriber to Subscription Manager to terminate a subscription. Answered with a *TerminateSubscriptionResponse*.
Id	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Creation time of notice of change message.
Auth.	***AccountId***	0:1	*+Structure*	Account Identifier. May be used to attribute requests to a specific user account for authentication or reporting purposes +SIRI v2.0
	AccountKey	0:1	*+Structure*	Authentication key for request. May be used to authenticate the request to ensure the user is a registered client. +SIRI v2.0
Endpoint properties	***Address***	0:1	*EndpointAddress*	Address of Subscriber.
	RequestorRef	**1:1**	*→ParticipantCode*	Identifies the Requestor.
	RequestMessageIdentifier	0:1	*MessageQualifier*	Arbitrary unique identifier for this message. Can be used to reference it subsequently.
Delegation	***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Topic	***SubscriberRef***	0:1	*➔ParticipantCodeType*	Participant identifier of Subscriber. Subscription ref will be unique within this. SIRI v1.3
			choice	Either All or a named Subscription.
	a ***All***	**–1:1**	*EmptyType*	Terminate all subscriptions for the Subscriber.
	b ***SubscriptionRef***		*SubscriptionQualifier*	Identifies the specific subscription to be terminated.
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

7.3.2.2 TerminateSubscriptionRequest — Example

The following is an example of a ***TerminateSubscriptionRequest*** message:

```
<TerminateSubscriptionRequest version="1.0" lang="en">
   <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
   <RequestorRef>NADER</RequestorRef>
   <All>Text</All>
</TerminateSubscriptionRequest>
```

7.3.3 The TerminateSubscriptionResponse

7.3.3.1 TerminateSubscriptionResponse — Element

The ***TerminateSubscriptionResponse*** is sent back to the [*Subscriber*] endpoint indicated by the request, and will contain an acknowledgment or error code for each subscription terminated.

Table 23 — TerminateSubscriptionResponse — Attributes

TerminateSubscriptionResponse				*+Structure*	Response from Subscription Manager to Consumer to inform whether subscriptions have been removed. Answers a ***TerminateSubscriptionRequest***.
Endpoint properties	***ResponseTimestamp***		**1:1**	*xsd:dateTime*	Creation time of response.
	ResponderRef		0:1	*→ParticipantCode*	Identifies the Producer.
	RequestMessageRef		0:1	*MessageQualifier*	Reference to a message for which this is the response.
Delegation	***DelegatorAddress***		0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Payload	***TerminationResponseStatus***		**1:***	*+Structure*	Status of each response to each subscription termination.
		ResponseTimestamp	0:1	*xsd:dateTime*	Creation time of response status
		SubscriberRef	0:1	*→ParticipantCode*	Unique Identifier of Subscriber.
		SubscriptionRef	**1:1**	*SubscriptionQualifier*	Unique Identifier of Subscription
		Status	0:1	*xsd:boolean*	Whether the subscription could be cancelled. Default is true.
		ErrorCondition	0:1	*+Structure*	Error Condition that applies to a ***TerminateSubscriptionResponse***.
				choice	One of the following error codes.
		a ***CapabilityNotSupportedError***	**–1:1**	*+Error*	Capability not supported.
		b ***UnknownSubscriberError***		*+Error*	Subscriber not found.
		c ***UnknownSubscriptionError***		*+Error*	Subscription not found.
		d ***OtherError***		*+Error*	Other Error.
		Description	0:1	*→ErrorDescription*	Description of the error in plain text.
any	***Extensions***		0:1	*xsd:any**	Placeholder for user extensions.

7.3.3.2 TerminateSubscriptionResponse — Example

The following is an example of a ***TerminateSubscriptionResponse*** message:

```
<TerminateSubscriptionResponse>
   <ResponseTimestamp>2004-12-17T09:30:47-05:00</ResponseTimestamp>
   <ResponderRef>KUBRICK</ResponderRef>
   <!- GOOD Response ->
   <TerminationResponseStatus>
      <ResponseTimestamp>2004-12-17T09:30:47-05:00</ResponseTimestamp>
      <SubscriberRef>NADER</SubscriberRef>
```

```
        <SubscriptionRef>0000456</SubscriptionRef>
        <Status>true</Status>
    </TerminationResponseStatus>
    <!- BAD Response ->
    <TerminationResponseStatus>
        <ResponseTimestamp>2004-12-17T09:30:47-05:00</ResponseTimestamp>
        <SubscriberRef>NADER</SubscriberRef>
        <SubscriptionRef>0000457</SubscriptionRef>
        <Status>false</Status>
        <ErrorCondition>
            <UnknownSubscriptionError/>
        </ ErrorCondition>
    </TerminationResponseStatus>
</TerminateSubscriptionResponse>
```

7.3.4 The SubscriptionTerminatedNotification (SIRI 2.0)

7.3.4.1 SubscriptionTerminatedNotification — Element

The SubscriptionTerminatedNotifcation can be used by a producer to give an explicit warning to a consumer that a subscription has had to be terminated. This allows for a more efficient handling of loss of service errors in some circumstances by giving the client immediate notication that it will need to restart a subscription to a service.

Table 24 — SubscriptionTerminatedNotification — Attributes

SubscriptionTerminatedNotification			*+Structure*	Notification from Subscription Manager to SUbscriber that subscribtion has been terminate a subscription. (+Siri 2.0)
Log	***ResponseTimestamp***	**1:1**	*xsd:dateTime:*	Time Notifcation is issued..
Endpoint	***ProducerRef***	0:1	*→ParticipantCode*	Identifies the Producer whose subscription is being terminated.
	Address	0:1	*EndpointAddress*	Endpoint Address to which acknowledgements to confirm delivery are to be sent.
	ResponseMessageIdentifier	0:1	*MessageQualifier*	An arbitrary unique reference associated with the response which may be used to reference it.
	RequestMessageRef	0:1	*MessageQualifier*	Reference to an arbitrary unique identifier associated with the request which gave rise to this response.
Delegator	***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCodeType*	Identifier of delegating system that originated message. +SIRI 2.0
Subscription	***SubscriberRef***	0:1	*→ParticipantCodeType*	Participant identifier of Subscriber. Whose subscription has been terminated.
	SubscriptionFilterRef	0:1	*→FilterRefStructure*	Unique identifier of Subscription filter to which this subscription is assigned. If there is onlya single filter, then can be omitted.
	SubscriptionRef	**1:1**	*SubscriptionQualifier*	Identifies the specific subscription that has been terminated.

	ErrorCondition	0:1	*+Structure*	Error Condition that applies to a ***Subscription Termination***
			Choice	One of the following error codes.
	q ***OtherError***		*+Error*	Other error.
	Description	0:1	*→ErrorDescription*	Description of the error in plain text.
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

7.3.4.2 SubscriptionTerminatedNotification — Example

The following is an example of a ***SubscriptionTerminatedNotification*** message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (C) Copyright 2005-2014 CEN SIRI -->
<Siri xmlns="http://www.siri.org.uk/siri"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://www.siri.org.uk/siri../../xsd/siri.xsd">
<SubscriptionTerminatedNotification>
<ResponseTimestamp>2004-12-17T09:30:47-05:00</ResponseTimestamp>
<ProducerRef>KUBRICK</ProducerRef>
<SubscriberRef>NADER</SubscriberRef>
<SubscriptionRef>987653</SubscriptionRef>
<ErrorCondition> <OtherError number="123"></OtherError>
<Description>Weekley restart </Description>
</ErrorCondition>
</SubscriptionTerminatedNotification>
</Siri>
```

8 Delivering data

8.1 Direct Delivery

8.1.1 Introduction

For Direct Delivery of subscriptions, a ***ServiceDelivery*** message is sent by the Producer to the [*Consumer*] endpoint of the Consumer for each update of package of updates; see 5.3 on mediation behaviour.

The ***ServiceDelivery*** will include one or more delivery messages for a single concrete SIRI Functional Service as described for each individual service in Part 3. The Consumer may enqueue these messages and process them as it needs. A Direct Delivery is similar in effect to the last step of Fetched Delivery for a subscription

8.1.2 Acknowledging Receipt of Data (DataReceivedAcknowledgement)

8.1.2.1 DataReceivedAcknowledgement — Element

If the system supports the SIRI optional *ConfirmDelivery* capability, and is configured for confirmed delivery, the consumer shall acknowledge that data has arrived. After receiving all the data, the Consumer will return a ***DataReceivedAcknowledgement*** message to the [*GetData*] endpoint of the Producer.

Table 25 — DataReceivedAcknowledgement — Attributes

DataReceivedAcknowledgement				*+Structure*	Response from Consumer to Producer to acknowledge that data has been received. Used as optional extra step if reliable delivery is needed. Answers a ***ServiceDelivery***
Log		***ResponseTimestamp***	**1:1**	*xsd:dateTime*	Time individual response element was created.
Endpoint properties		***ResponderRef***	0:1	*→ParticipantCode*	Consumer who is acknowledging the data delivery.
		RequestMessageRef	0:1	*→MessageQualifier*	Reference to a unique identifier associated with the request which gave rise to this response.
Delegation		***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Payload		***Status***	0:1	*xsd:boolean*	Whether data could be processed or not. Default is *true*. *False* if error.
		ErrorCondition	0:1	*+Structure*	Error Condition that applies to a ***DataReceivedAcknowledgement***.
				choice	One of the following error codes.
	a	***UnknownSubscriptionError***	**–1:1**	*+Error*	Subscriber not found.
	b	***OtherError***		*+Error*	Error other than a well-defined category.
		Description	0:1	*→ErrorDescription*	Description of the error in plain text

8.1.2.2 DataReceivedAcknowledgement — Example

The following is an example of a ***DataReceivedAcknowledgement*** message:

```
<DataReceivedAcknowledgement>
   <ResponseTimestamp>2001-12-17T09:30:47-05:00</ResponseTimestamp>
   <ConsumerRef>NADER</ConsumerRef>
   <RequestMessageRef>012225678</RequestMessageRef>
   <Status>true</Status>
 </DataReceivedAcknowledgement>
```

8.2 Fetched Delivery

8.2.1 Introduction

Fetched Delivery of subscriptions delivers the data in two steps; a notification from the Producer to the Consumer; and then a data supply request from the Consumer to the Producer to fetch the data.

If there is a single Subscription Filter, then no message key is needed on the data supply request: the response will simply include all data for all subscriptions for the subscriber.

If there are multiple Subscription Filters, a '***NotificationRef***' – the notification message identifier issued by the Producer – can be used as a message key to identify the specific Channel of the notification on subsequent steps.

8.2.2 Signalling Data Availability (DataReadyNotification / DataReadyResponse)

8.2.2.1 Procedure

Once the subscription has been set up and data has become available, the data Consumer is notified of the existence of updated data via a ***DataReadyNotification*** message, sent to the [*Notify*] endpoint of the subscription or service request. A message is sent by the Notification Producer for every change to topic data associated with the subscription that meets the subscription policy. Typically the subscription policy will use aggregation and change sensitivity mechanisms to reduce the number of notifications sent – see clause on Mediation earlier. The notification has a return endpoint associated with it.

8.2.2.2 DataReadyNotification— Element

Table 26 — DataReadyNotification — Attributes

DataReadyNotification			*DataReadyRequest Structure*	Request from Producer to Consumer to notify that data update is ready to fetch. Answered with a ***DataReadyResponse***.
Log	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Creation time of notice of change message.
Endpoint properties	***Address***	0:1	*EndpointAddress*	Address to which response is to be sent. This may also be determined from ***ProducerRef*** and preconfigured data.
	ProducerRef	0:1	*→ParticipantCode*	Identifies the Notification Producer. This element is mandatory for Publication/Subscription use.
	RequestMessageIdentifier	0:1	*MessageQualifier*	Arbitrary unique identifier for this message. Can be used to reference it subsequently.
Delegation	***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0

8.2.2.3 DataReadyNotification — Example

The following is an example of a ***DataReadyNotification*** message:

```
<DataReadyNotification>
   <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
   <ProducerRef>KUBRICK</ ProducerRef>
</DataReadyNotification>
```

8.2.2.4 DataReadyAcknowledgement— Element

The data Consumer (client) then confirms reception of the notification signal with a ***DataReadyAcknowledgement*** message sent back to the [*GetData*] endpoint of the notification request. This message contains a Status element to indicate success or failure in processing the notification.

Table 27 — DataReadyAcknowledgement — Attributes

DataReadyAcknowledgement				*DataReadyResponseStructure*	Response from Consumer to Producer to acknowledge that data has been received. Used as optional extra step if reliable delivery is needed. Answers a ***ServiceDelivery***.
Log	***ResponseTimestamp***		**1:1**	*xsd:dateTime*	Time individual data received response element was created.
Endpoint properties	***ConsumerRef***		0:1	*→ParticipantCode*	Consumer who is responding to Data notification.
	RequestMessageRef		0:1	*→MessageQualifier*	Reference to identifier of notification message that this response acknowledges.
Delegation	***DelegatorAddress***		0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Payload	***Status***		0:1	*xsd:boolean*	Whether notification data could be processed or not. Default is *true*. *False* if errors. If *false*, an error condition shall be given.
	ErrorCondition		0:1	*+Structure*	Error Conditions that apply to a ***DataReadyAcknowledgement***.
				choice	One of the following error codes.
	a	***UnknownSubscriptionError***	**–1:1**	*+Error*	Subscriber not found.
	b	***OtherError***		*+Error*	Other error.
		Description	0:1	*→ErrorDescription*	Description of the error in plain text.

8.2.2.5 DataReadyAcknowledgement — Example

The following is an example of a ***DataReadyAcknowledgement*** message:

```
<DataReadyAcknowledgement>
    <ResponseTimestamp>2004-12-17T09:30:47-05:00</ResponseTimestamp>
    <ConsumerRef>NADER</ConsumerRef>
    <Status>true</Status>
</DataReadyAcknowledgement>
```

The data can now be retrieved by the Consumer, acting as a client. If the Consumer does not wish to retrieve data at this particular moment, retrieval can be postponed to a later time. The polling of updated data is done subsequently and independently of the initial notification, i.e. the signalling of the fact that a change has taken place.

8.2.3 Polling Data (DataSupplyRequest/ServiceDelivery)

8.2.3.1 Procedure

Data polling occurs at the initiative of the data Consumer. Only the current real-time information is transmitted – see discussion of mediation earlier. Historical data is not available.

Polling generally occurs after updated data is signalled by a ***DataReadyNotification***, but can also occur at any time after setting up the subscription (the WS-SubPub 'Get Current Message' function).

8.2.3.2 DataSupplyRequest Message — Element

To fetch the data, the Consumer sends a ***DataSupplyRequest*** message to the [*GetData*] endpoint of the Producer, which prompts the Producer to supply the data that has been updated since the last ***DataSupplyRequest***: The [*GetData*] endpoint may either be configured, or specified as the return address on the request.

The data may be; (i) all data for the channel; (ii) all data for a given notification.

Table 28 — DataSupplyRequest — Attributes

DataSupplyRequest			*+Structure*	Request from Consumer to Producer to fetch update previously notified by a Data ready message. Answered with a Service Delivery.
Log	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Time individual data supply request was created.
Auth.	***AccountId***	0:1	*+Structure*	Account Identifier. May be used to attribute requests to a specific user account for authentication or reporting purposes +SIRI v2.0
	AccountKey	0:1	*+Structure*	Authentication key for request. May be used to authenticate the request to ensure the user is a registered client. +SIRI v2.0
Endpoint properties	***Address***	0:1	*EndpointAddress*	Address to which response is to be sent. This may also be determined from RequestorRef and preconfigured data.
	ConsumerRef	0:1	*→ParticipantCode*	Consumer who is requesting data. This element is mandatory for Publication/Subscription use.
	MessageIdentifier	0:1	*MessageQualifier*	Arbitrary unique identifier for this message. Can be used to reference it subsequently.
Delegation	***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Topic	***NotificationRef***	0:1	*→MessageQualifier*	Reference to identifier of notification message that this response acknowledges. If not specified then data for all subscriptions for the Consumer will be fetched.
	AllData	0:1	*xsd:boolean*	Whether all data is to be returned (Get Current Message) or just updates since last recorded delivery date for subscription. Default is *false.*

If ***AllData*** is set to *false*, then the Producer will transmit only the data that has been updated since the last request. The read is destructive; that is, it cannot be repeated because once the data has been fetched the last update flag has been reset. See discussion under mediation for further considerations.

If ***AllData*** is set to *true*, then the Producer will transmit not only the data that has been updated since the last request, but all data for all active subscriptions held by the Consumer. The read is non-destructive; that is, it

can be repeated until the data is stale. This corresponds to the WS-PubSub Get Current message. See discussion under mediation for further considerations.

8.2.3.3 DataSupplyRequest — Example

The following is an example of a ***DataSupplyRequest*** message for a single, well known aggregated channel:

```
<DataSupplyRequest>
    <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
    <AllData>false</AllData>
</DataSupplyRequest>
```

The following is an example of a ***DataSupplyRequest*** message for a referenced notification:

```
<DataSupplyRequest>
    <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
    <ConsumerRef>NADER</ConsumerRef>
    <NotificationRef>2004-12-17T09:30:42-06:00</NotificationRef>
    <AllData>false</AllData>
</DataSupplyRequest>
```

8.2.3.4 ServiceDelivery Message

The Notification Producer for the Service responds to the ***DataSupplyRequest*** with the data updates, contained within of a ***ServiceDelivery*** message. The ***ServiceDelivery*** contains one or more to one or more delivery messages for a single SIRI Functional service. See 6.2.2 earlier.

8.3 Delegated Delivery +SIRI 2.0

Some configurations wish to proxy client requests from a SIRI consumer via an intermediate aggregating system, using SIRI messages both between (i) the client consumer and aggregator (as producer), and (ii) the aggregator (as consumer) and the actual back end producer service. To facilitate this SIRI requests and responses may be tagged with a ***DelegatorAddress*** and a ***DelegatorRef***. These provide tracking information relating to the original requestor that can be echoed back in service responses and used to delegate messages back to the original requestor. This allows the aggregation itself to be stateless.

Delegation may give rise to additional error conditions.

9 Recovery from system failure

9.1 Introduction

A SIRI Client, a SIRI Server or the communications link between them may undergo failure, or any combination thereof. Each of these cases is discussed separately below.

9.2 Recovery after Client Failure

It is the responsibility of the Subscriber to remember the state needed to recreate subscriptions in the event of a failure by the Notification Producer.

When a Subscriber requests a subscription, it creates its own state to represent the subscription and to allow it to associate incoming deliveries with its own and/or its Consumer's data. If the Subscriber loses this state, for example after a crash of the client machine, all the Subscriber's subscriptions with the Producer shall be recreated. The first step is to delete all subscriptions on the Producer (***TerminateSubscriptions*** / ***All***), and then to create them again.

9.3 Recovery after Server Failure

If the Notification Producer loses its subscription data, and a heartbeat *is* available, a significant service failure will become obvious by the failure to arrive of a heartbeat within the prescribed interval. The recovery action will be to recreate a new set of subscriptions.

If the Notification Producer loses its subscription data, and a heartbeat is *not* available, it is not immediately obvious to the Consumer. There will be an absence of ***DataReadyNotification*** messages, but the Consumer is unable to distinguish this condition from a normal period of quiet operation. In order to detect a service failure, ***CheckStatusRequest*** messages (9.5.2) need to be sent periodically to the Producer. A ***CheckStatusResponse*** (9.5.3) should be returned by the Producer, specifying the time stamp of the service start. If the start of the service is after the time at which the Consumer requested the subscription to be set up, loss of the subscription shall be assumed. The recovery procedure is now as for client data loss – delete and recreate all subscriptions.

9.4 Reset after Interruption of Communication

A break in the data connection can be determined by the requestor from the timeout of the HTTP protocol. The effect will be a failure for a message to reach its recipient. If the message is the response to a request, the absence of a response can generally be detected with a timeout.

The currently recognised failure conditions are shown in Table 29.

Table 29 — Error Statuses and Actions in Communication Failure Conditions

Lost message	Lost on way to	Failure Condition	Detection	Recovery action
SubscriptionRequest	Producer	Failed to receive subscription	Subscriber received no reply	Subscriber resends request again.
SubscriptionReply	Subscriber	Failed to inform of new subscription		Subscriber resends request with same reference. Subscription is overwritten.
DataReadyNotification	Consumer	Failed to receive notification.	Producer received no reply	Producer sends notification again.
DataReadyResponse	Producer	Failed to acknowledge notification.		Resend of the request, until reply is received from the client, or a timeout.
DataSupplyRequest	Producer	Failed to receive supply request.	Consumer receives no data supply reply	Consumer shall assume the request has been lost (worst case) and request all data again (***DataSupplyAll***).
DataDelivery	Consumer	Failed to receive data response.		Data lost, renewed polling not possible because the server has reset the update flag of the subscription, i.e. further updates have been detected. Consumer shall make a new ***DataSupplyAll*** request. ***GetCurrentMessage***.
TerminateSubscriptionRequest	Producer	Failed to receive subscription termination.	Subscriber received no reply	Subscriber retransmits message until it receives a reply, or there is an error message concerning an unknown Subscription Identifier or there is a timeout.
TerminateSubscriptionResponse	Consumer	Failed to receive subscription termination reply.		
CheckStatusRequest	Producer	Failed to receive status request.	Consumer received no reply	Sender retransmits up to timeout, after which it assumes Service is no longer available.

Lost message	Lost on way to	Failure Condition	Detection	Recovery action
CheckStatusResponse	Consumer	Failed to receive status response.		Sender retransmits. Producer responds again up to timeout, after which it assumes Service is no longer available.

9.5 Alive Handling

9.5.1 Introduction

Status polling is used to monitor the availability of the SIRI functional services. This can either be on demand Status Check, or an automatic Heartbeat.

9.5.2 CheckStatusRequest

9.5.2.1 CheckStatusRequest Message — Element

If the client wishes to establish whether the service is still “alive”, it sends a ***CheckStatusRequest*** to the server and waits for the reply (***CheckStatusResponse***). The message is sent to the [*CheckStatus*] endpoint for the SIRI functional service.

The ***CheckStatus*** message shall always be implemented.

The ***CheckStatusRequest*** also enables the client to detect whether a service has been started again, and that the subscriptions have been lost. Within ***CheckStatusRequest*** the server specifies the last start time of the service (***ServiceStartedTimestamp***). A start time after the set-up of a subscription indicates that it has been restarted at some point in between (9.3).

Table 30 — CheckStatusRequest — Attributes

CheckStatusRequest			*+Structure*	Request from Consumer to Producer to inform it of current system status. The Endpoint address to which request is sent determines which service is checked.
Attributes	***version***	0:1	*VersionString*	Version Identifier of Functional Service, e.g. ‘1.0c’.
Log	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Time of request.
Auth.	***AccountId***	0:1	*+Structure*	Account Identifier. May be used to attribute requests to a specific user ACCOUNT for authentication or reporting purposes +SIRI v2.0 Note that an ACCOUNT may be shared between more than one consumer device, for example if used to authenticate an application.
	AccountKey	0:1	*+Structure*	Authentication key for request. May be used to authenticate the request to ensure the user is a registered and approved client. +SIRI v2.0.
Endpoint	***Address***	0:1	*EndpointAddress*	Address to which response is to be sent
	RequestorRef	**1:1**	*→ParticipantCode.*	Identifier of requestor – Participant Code.
Identity	***MessageIdentifier***	0:1	*MessageQualifier*	Arbitrary unique identifier for this message. Can be used to reference it subsequently.
Delegator endpoin	***DelegatorAddress***	0:1	*EndpointAddress*	Address of originated system to which delegated response is to be returned. +SIRI 2.0.

t				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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

9.5.2.2 CheckStatusRequest — Example

The following is an example of a ***CheckStatusRequest*** message:

```
<CheckStatusRequest>
    <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
    <RequestorRef>NADER</RequestorRef>
</CheckStatusRequest>
```

9.5.3 CheckStatusResponse

9.5.3.1 CheckStatusResponse Message — Element

Table 31 — CheckStatusResponse — Attributes

CheckStatusResponse				*+Structure*	Response from Producer to Consumer to inform whether services is working. Answers a previous ***CheckStatusRequest***.
Log		***ResponseTimestamp***	**1:1**	*xsd:dateTime:*	Time of Response.
Endpoint		***ResponderRef***	0:1	*→ParticipantCode*	Identifies the Producer or Service whose status is being checked.
		RequestMessageRef	0:1	*MessageQualifier*	Reference to identifier of check status message that this response acknowledges.
Delegator		***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
Payload		***Status***	0:1	*xsd boolean*	Whether the service is available. *False* if not available. Default is *true*.
		DataReady	0:1	*xsd boolean*	Whether data delivery is ready to be fetched +SIRI v2.0
		ErrorCondition	0:1	*+Structure*	Error Condition that applies to a ***CheckStatusResponse.***
				Choice	One of the following error codes.
	a	***ServiceNotAvailableError***	**1:1**	*+Error*	Service is not available.
	b	***UnknownSubscriberError***		*+Error*	Subscriber is not known.
	c	***OtherError***		*+Error*	Other error.
		Description	0:1	*→ErrorDescription*	Description of the error in plain text.
		ValidUntil	0:1	*xsd:dateTime:*	End of data horizon of the data producer.

	ShortestPossibleCycle	0:1	*PositiveDurationType*	Minimum separation between two updates.
	ServiceStartedTime	0:1	*xsd:dateTime:*	Specifies the time of the start of the service. If the service is not available to deliver data, No value should be given here.
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

The ***CheckStatusResponse*** indicates the availability of the SIRI Functional Service. If the System is completely unavailable there will be no reply. The ***CheckStatusResponse*** also provides status information that can be used to establish if there has been an outage. If the Service started time is later than the creation time for the subscription, then it is likely the subscriptions are not current and that the data set of the Consumer may be incomplete.

The ***CheckStatusResponse*** is sent to the [*ReportStatus*] endpoint indicated by the request. The ***CheckStatusResponse*** describes the total availability of all information channels of a service, i.e. messages sent to any endpoint, and shall return false if any endpoint is not working. If a single channel is unavailable, the entire service is no longer considered available, i.e. both data supply, subscription management and shall be unavailable.

9.5.3.2 CheckStatusResponse — Example

The following is an example of a ***CheckStatusResponse*** message:

```
   <CheckStatusResponse>
      <ResponseTimestamp>2001-12-17T09:30:47-05:00</ResponseTimestamp>
      <Status>true</Status>
      <ValidUntil>2004-12-17T19:30:47-05:00</ValidUntil>
      <ShortestPossibleCycle>PT2M</ShortestPossibleCycle>
      <ServiceStartedTimeStamp>2004-12-17T09:30:47-
05:00</ServiceStartedTimeStamp>
   </CheckStatusResponse>
```

9.5.4 HeartbeatNotification

9.5.4.1 Heartbeat Message — Element

Table 32 — HeartbeatNotification — Attributes

HeartbeatNotification			*+Structure*	Response from Producer to Consumer to inform if service is working. Sent at regular intervals.
Log	***RequestTimestamp***	**1:1**	*xsd:dateTime:*	Time of Heartbeat Notification.
Endpoint	***Address***	0:1	*EndpointAddress*	Address to which any life sign response is to be sent. Not currently used, left in for in uniformity.
	ProducerRef	0:1	*→ParticipantCode*	Identifies the Notification Producer or Service whose status is being checked.
	MessageIdentifier	0:1	*MessageQualifier*	Arbitrary unique reference to this message.
Delegation	***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0

Payload	***Status***		0:1	*xsd:boolean*	Whether the service is available. *False* if not available. Default is *true*.
	ErrorCondition		0:1	*+Structure*	Error Conditions that apply to a ***HeartbeatNotification.*** *One of the following:*
				choice	One of the following error codes.
	a	***ServiceNotAvailableError***	**–1:1**	*+Error*	Service is not available.
	b	***OtherError***		*+Error*	Other error.
		Description	0:1	*→ErrorDescription*	Description of the error in plain text.
	ValidUntil		0:1	*xsd:dateTime:*	End of data horizon of the data producer.
	ShortestPossibleCycle		0:1	*PositiveDurationType*	Minimum separation between two updates.
	ServiceStartedTime		0:1	*xsd:dateTime:*	Specifies the time of the start of the service.
any	***Extensions***		0:1	*xsd:any**	Placeholder for user extensions.

The ***HeartbeatNotification*** message reports the availability of the SIRI Functional Service at regular intervals. It is an optional SIRI capability.

A heartbeat is sent if configured as a system feature; at the preconfigured heartbeat interval. A single heartbeat message is sent for each subscriber channel; if the consumer has many subscriptions, it will still get only one heartbeat.

If the System is completely unavailable there will be no heartbeat. The ***HeartbeatNotification*** message content is similar to that of ***CheckStatusResponse***, and is also sent to the [*ReportStatus*] endpoint of the Consumer. If the Service started time is later than the creation time for the subscription, then it is likely the subscriptions are not current and that the data set of the Consumer may be incomplete.

9.5.4.2 HeartbeatNotification — Example

The following is an example of a ***HeartbeatNotification*** message:

```
   <HeartbeatNotification>
      <RequestTimestamp>2001-12-17T09:30:47-05:00</RequestTimestamp >
      <ProducerRef>KUBRICK</ProducerRef>
      <Status>true</Status>
      <ValidUntil>2004-12-17T19:30:47-05:00</ValidUntil>
      <ServiceStartedTimeStamp>2004-12-17T09:30:47-
05:00</ServiceStartedTimeStamp>
   </ HeartbeatNotification>
```

9.6 Additional Failure modes for delegated delivery (+SIRI v2.0)

If an intermediate aggregator is being used to forward messages, additional modes of failure may arise. For example, if the aggregator is unable to deliver a message to the original requestor because the endpoint is invalid, unauthorised or not available. These can be reported with ***UnknownParticipant, EndpointDeniedAccessError*** or ***EndpointNotAvailableAccessError*** Error conditions.

10 Transport of SIRI messages

10.1 Separation of Addressing from Transport Protocol

WS-PubSub requires that the message transport be independent of the binding to the XML used to serialise functional service content. This separation of concerns allows different communication transport methods to be used, and also permits different middleware to be used for queuing and dequeuing messages.

In particular it should be possible to encode the endpoint references used to specify the Web Addresses (that is, URIs and ports) of communicating systems in different ways for different transport protocols. This requires identification and encoding of the different logical endpoints needed for different types of SIRI messages.

The normal way of exchanging SIRI XML messages is via HTTP using the POST method, with an XML document containing the SIRI encoded message as a simple attachment. Messages may also be exchanged by other means, for example within a SOAP envelope.

SIRI allows two different capability levels for endpoint addressing:

Implicit Addressing – the endpoint addresses are not exchanged within the SIRI API. The initial endpoint addresses of the service are supplied as part of the configuration. The return address to which the Functional Service should send responses is taken from the http request, i.e. is bound into the transport protocol. Both responses and subscriptions are sent to the same address of the requestor.

Explicit Addressing – return addresses may be specified as part of the request, allowing protocol independent access to the full endpoint properties. If desired, different addresses can be specified for both Consumer and Subscriber. The Functional Service address can be returned by the Universal Discovery Service.

10.2 Logical Endpoint Addresses

10.2.1 Endpoint Addresses

For each different SIRI Functional Service, SIRI identifies different Logical Service endpoints, which may be distinct addresses, or all be mapped to the same concrete URI. Table 35 shows the endpoints for server functions. These addresses can be configured in the ***ServiceRequestContext***.

Table 33 — Server Logical Endpoints

Server Logical Endpoint Name	Messages sent to endpoint	Description
[CheckStatus]	***CheckStatusRequest***	Address to which send requests to check that the Functional Service is available.
[Subscribe]	***SubscriptionRequest***	Address to which send requests for new subscriptions to the Functional Service.
[ManageSubscriptions]	***TerminateSubscription***	Address to which send requests to change or delete subscriptions to the Functional Service. Normally the same as *Subscribe*.
[GetData]	***DataReadyResponse***	Address to which send requests to fetch data, and confirmation of successful receipt.
	DataSupplyRequest	
	DataReceivedAcknowledgement	

Table 36 shows the endpoints to access client functions, that is, the addresses to which different types of responses to client requests will be sent. If not explicitly specified, the response address may be taken from the http request.

Table 34 — Client Logical Endpoints

Client Logical Endpoint Name	Messages sent to endpoint	Description
[ReportStatus]	***CheckStatusResponse***	Address to which send responses to inform the Consumer that the Functional Service is available.
	HeartBeat	
[Subscriber]	***SubscriptionResponse***	Address to which send responses to requests to create, change or delete subscriptions.
	TerminateSubscriptionResponse	
[Notify]	***DataReadyNotification***	Address to which send notifications of data being ready.
[Consumer]	***ServiceDelivery***	Address to which send data.

10.2.2 Endpoint Address — Examples

The following subscription request for the Stop Monitoring service includes an explicit endpoint address of; (i) the [*Subscriber*] making the request (***Address*** element, *'http://myhost:8080/nader/dpi/subscription.xml'*) to whom the subscription response should be sent to confirm that the subscription has been successfully created, and; (ii) the [*Consumer*] to whom the Stop Monitoring notification and data messages should be despatched (***ConsumerAddress*** element, '*http://myhost:8080/nader/dpi/dataready.xml'*).

The request is sent to the [*Subscriber*] endpoint address configured for the Producer, for example, '*http://yourhost:8080/KUBRICK/dpi/subscription.xml'.*

```
   <SubscriptionRequest>
      <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
      <Address>http://myhost:8080/nader/dpi/subscription.xml</Address>
      <RequestorRef>NADER</RequestorRef>

   <ConsumerAddress>http://myhost:8080/nader/dpi/dataready.xml</ConsumerAddress>
      <StopMonitoringSubscriptionRequest>
         <SubscriberRef>NADER</SubscriberRef>
         <SubscriptionIdentifier>000234</SubscriptionIdentifier>
         <InitialTerminationTime>2004-12-17T09:30:47-
05:00</InitialTerminationTime>
         <StopMonitoringRequest version="1.0">
            <RequestTimestamp>2004-12-17T15:30:47-05:00</RequestTimestamp>
            <MonitoringRef>POIT5678</MonitoringRef>
         </StopMonitoringRequest>
      </StopMonitoringSubscriptionRequest>
   </SubscriptionRequest>
```

If the same subscription was submitted without explicit addresses, the values for the [*Subscriber]* and [*Consumer*] endpoints would instead be taken from the preconfigured values for the requestor participant as in Table 35.

Table 35 — Client Logical Endpoints

Client Logical Endpoint Name	Messages sent to endpoint	Default Response Endpoint Address.
[ReportStatus]	**CheckStatusResponse**	http request.
	HeartBeat	Configured Participant Consumer Address.
[Subscriber]	**SubscriptionResponse**	Configured Participant Subscriber Address.
	TerminateSubscriptionResponse	Configured Participant Subscriber Address.
[Notify]	**DataReadyNotification**	Configured Participant Consumer Address.
[Consumer]	**ServiceDelivery**	Configured Participant Consumer Address.

The following is an example of a ***SubscriptionRequest*** message:

```
   <SubscriptionRequest>
      <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
      <RequestorRef>NADER</RequestorRef>
      <StopMonitoringSubscriptionRequest>
         <SubscriberRef>NADER</SubscriberRef>
         <SubscriptionIdentifier>000234</SubscriptionIdentifier>
         <InitialTerminationTime>2004-12-17T09:30:47-
05:00</InitialTerminationTime>
         <StopMonitoringRequest version="0.1e">
            <RequestTimestamp>2004-12-17T15:30:47-05:00</RequestTimestamp>
            <MonitoringRef>POIT5678</MonitoringRef>
         </StopMonitoringRequest>
      </StopMonitoringSubscriptionRequest>
   </SubscriptionRequest>
```

10.3 Parallelism and Endpoint Addresses

The use of distinct endpoints for different functions allows for a degree of parallelism in implementations: status, subscription and data conversations between participants pair may be conducted simultaneously by different processes. The use of parallel processes does however open the possibility of conflict or race conditions between the different threads: for example a consumer may still be trying to fetch data for a subscription that is in the process of being cancelled. A strict precedence should be followed to resolve conflict, as follows:

1) Status: the status applies to all processes. If a system becomes unavailable, other conversations should cease.

2) Subscription: if a subscription is terminated, its data processing should cease – any attempt to access data will result in an error.

3) Data Supply: should yield to the higher priority.

10.4 Encoding of XML messages

10.4.1 Principles

Consistent conventions are used in the XML for all SIRI services. The following principles are applied to the encoding of SIRI XML messages:

— Use elements for everything except metadata,

— Use structures for all significant elements,

— Modularise into a subschema for each service,

— Use Upper CamelCase for Element names,

— Use lower camelCase for attributes and Enumerated values,

— Use a wrapper element for ‘one-to-many’ relationships,

— Use standard simple types, from XML, other standards and the SIRI type packages,

— Use enumeration rather than choice of tags for lists of options,

— Annotate all elements,

— Use consistent conventions for nouns in names e.g. time versus interval,

— Use names consistently;

10.4.2 Encoding of Errors in XML

Each different error condition is represented by an explicit subtype of a common abstract error type. The relevant error codes are explicitly enumerated on responses.

10.4.3 Character Set

UTF-8 is recommended.

10.4.4 Schema Packages

The SIRI schema is modularised into separate packages, with a strictly linear dependency graph. The packages effectively fall into three groups.

1. The SIRI terminal schemas: The (Siri.xsd) Schema is a convenience artefact that provides an entry point to all SIRI messages. It uses XML choice elements to give an explicit binding. An alternative terminal schema (SiriSG.xsd) is provided (+SIRI v1.3) that is syntactically identical but uses substitution groups and so can be used for arbitrary extension with additional functional services (as say by NeTEx).

2. SIRI Functional Service packages: for each SIRI Functional Service (SIRI-SM, SIRI-VM, etc.) there is a separate subschema package containing all request and responses, including capabilities for the service.

3. Common shared definitions: There are a number of small subschemas providing base type definitions and shared entities.

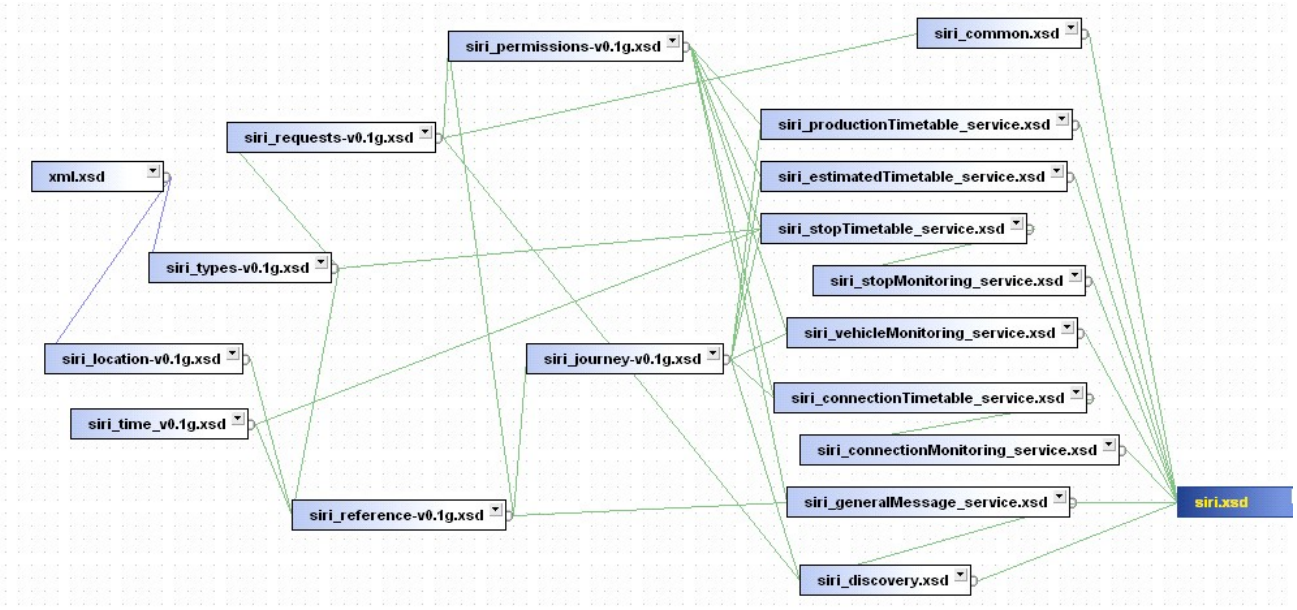

Figure 21 — SIRI Schema Packages

10.4.5 Siri.XSD – Use of XML Choice

Figure 22 shows an extract from the fixed SIRI schema showing the use of an explicitly coded XML choice to group the possible SIRI requests. Additional functional request types require a coded schema change.

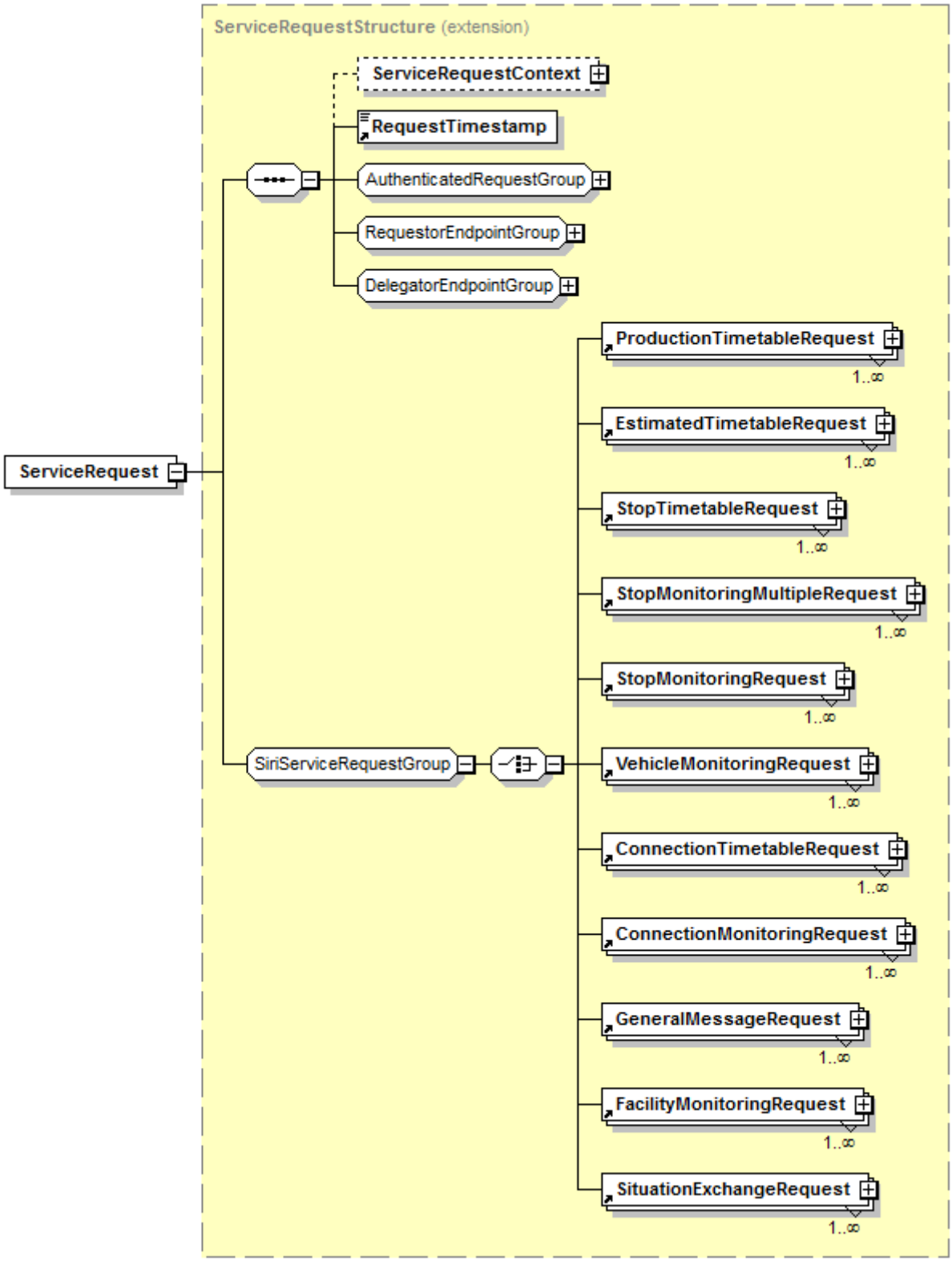

Figure 22 — Example for XML choice

10.4.6 SiriSG.XSD – Use of XML Substitution groups

Figure 23 shows the alternative loosely coupled SIRI schema which makes use of substitution groups. Additional functional request types may be added simply by including additional subschemas with additional request of the correct type.

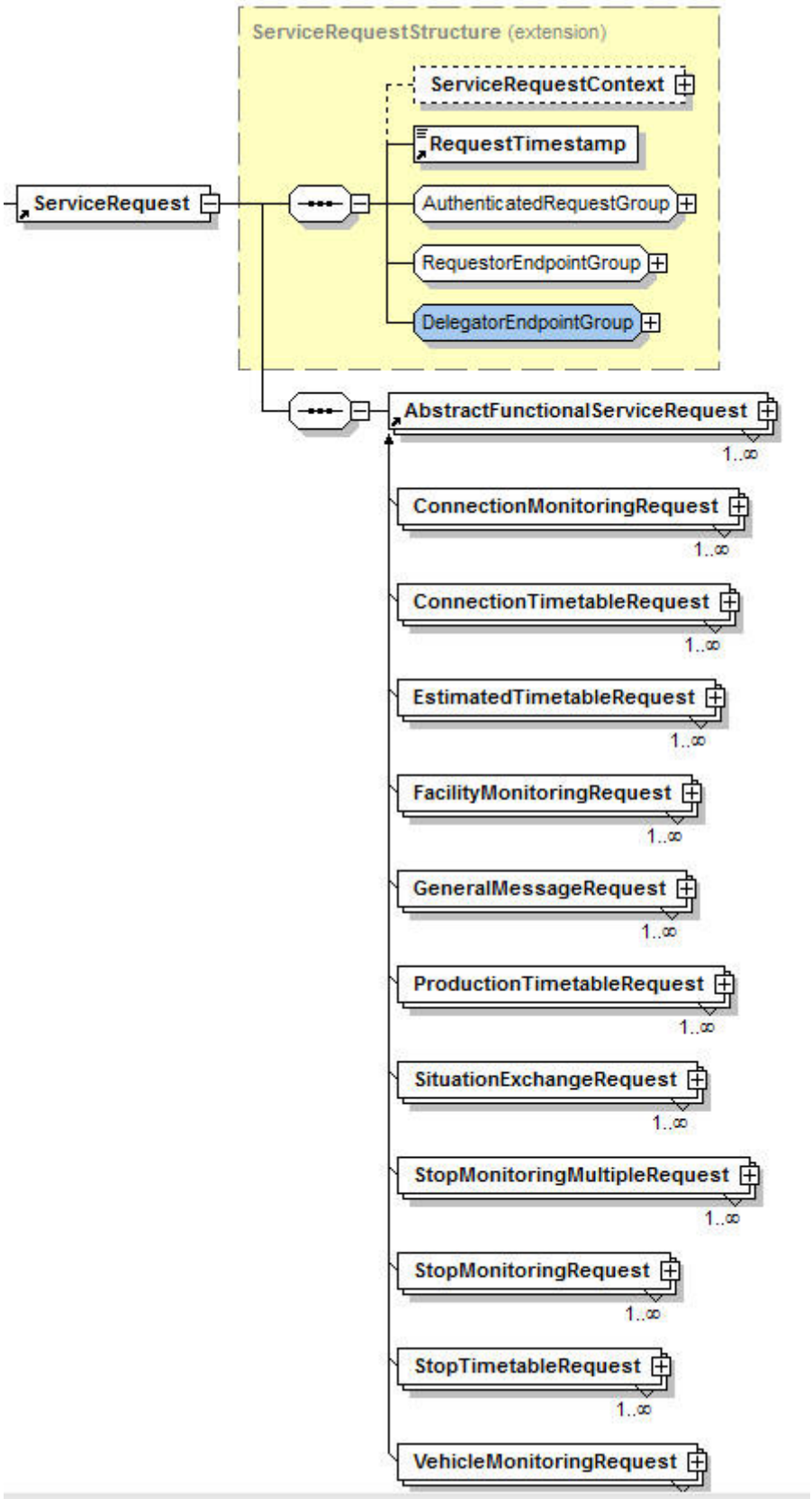

Figure 23 — Example: XML Substitution groups

10.5 Use of SIRI with SOAP / WSDL

10.5.1 Introduction

SIRI messages may also be exchanged wrapped within a SOAP envelope. The SIRI XML set includes example client and server WSDL bindings. This clause provides some discussion of the SIRI SOAP binding.

10.5.2 Web Services

10.5.2.1 General

Web Services are programmable applications made accessible using standard Internet protocols like HTTP, XML and SOAP. Like other software components, Web Services represent black-box functionality that can be reused without knowing in detail how the service is implemented. Web services therefore provide a ready solution for intersystem communication over an Internet/Intranet network, and are a good way of implementing the SIRI transport and communication layer, combining the best aspects of component based development and internet based communication.

Other points to note about Web Services:

— Can work through existing proxies and firewalls,

— Can take advantage of HTTP authentication,

— Can use SSL encryption without any modification,

— Can be incorporated easily with existing XML messaging solutions,

— Can take advantage of XML messaging schemas and offer an easy transition from XML RPC solutions,

— Are not subject to conflict between proprietary component based solutions, like say CORBA and COM,

— Are platform independent and available for a wide variety of clients;

10.5.2.2 SOAP (Simple Object Access Protocol)

Simple Object Access Protocol (SOAP) is a communication protocol specification that defines a uniform way of exchanging XML-encoded data over the internet as messages. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. It also defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol.

Other points about SOAP are:

— SOAP is platform independent,

— SOAP is programming language independent,

— SOAP is simple and extensible,

— SOAP provides a way to traverse firewalls,

— SOAP is being developed as a W3C standard;

10.5.2.3 WSDL (Web Services Definition Language)

Web Services Description Language (WSDL) is used to describe an application encapsulated as a Web Service according to a standard schema that makes it easy for new bindings to be added to use the application. A WSDL binding is written as an XML document that specifies the location of the service and the operations (or methods) the service exposes. WSDL definitions can also be used to build discovery services

WSDL provides a way for service providers to describe the basic format of web service requests over different protocols or encodings. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions of messages, which are abstract descriptions of the data being exchanged, and port types, which are abstract collections of operations. The SIRI WSDL implementation provides three compatible WSDL bindings: two with WSDL 1.1 encoding styles, RPC Literal and Document Literal wrapped (+SIRI v2.0), and one WSDL 2.0. All three use an HTTP transport.

A WSDL 1.1 document uses the following elements in the definition of network services:

— Types – a container for data type definitions using some type system (such as XSD),

— Message – an abstract, typed definition of the data being communicated,

— Operation – an abstract description of an action supported by the service,

— Port Type – an abstract set of operations supported by one or more endpoints,

— Binding – a concrete protocol and data format specification for a particular port type,

— Port – a single endpoint defined as a combination of a binding and a network address,

— Service – a collection of related endpoints;

The SIRI WSDL 1.1 interfaces follow the WS-I Basic Profile, which is a specification from the Web Services Interoperability industry consortium (WS-I, which recently became part of OASIS, providing guidelines and tests for interoperability of Web Services specifications such as SOAP, WSDL, and UDDI). The Version 2.0 of WS-I Basic Profile and corresponding testing tools were used for SIRI.

The WSDL 2.0 standard (formerly called WSDL 1.2 but later renamed WSDL 2.0 because of its substantial differences from WSDL 1.1) was designed to overcome interoperability issues from WSDL 1.x. However few vendors are supporting WSDL 2.0 today, but since SIRI is planned to be a long lasting standard, support for a SIRI WSDL 2.0 interface is also provided;

The main changes in WSDL 2.0 are:

— Addition of new semantics to the description language, including a new component model, making it easier to reuse components.

— Removal of the message constructs: WSDL 1.x messages are now specified using the XML schema type system in the types element allowing richer set of message exchange patterns.

— PortTypes renamed to interfaces.

— Ports renamed to endpoints.

— Support for interface inheritance.

The WSDL 2.0 specification offers better support for RESTful web services, and is much simpler to implement. However support for this specification is still poor in software development kits for Web Services

which often offer tools only for WSDL 1.1. As it is intended to solve main interoperability issues, there is no WS-I profile for WSDL 2.0

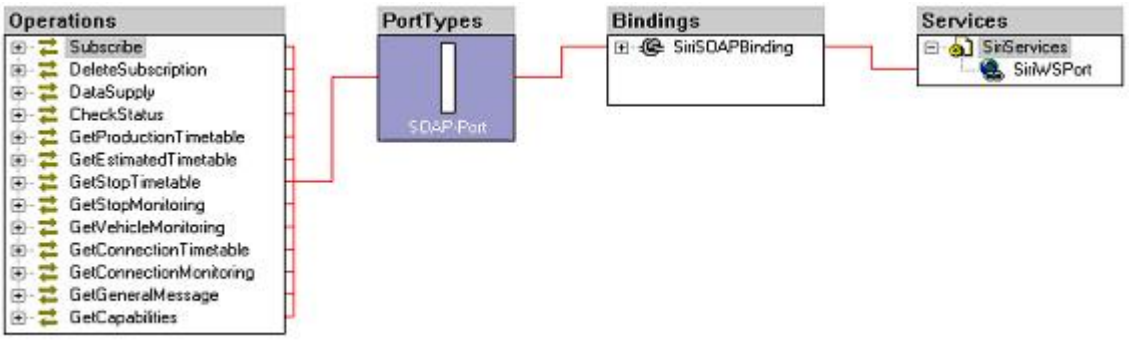

Figure 24 — WSDL 1.1 and 2.0 main concepts *(source http://en.wikipedia.org/wiki/File:WSDL_11vs20.png)*

10.5.3 Use of SOAP

SOAP over HTTP is widely used for intersystem communications and services in the area of business applications and standards and is a candidate of choice for Web Services.

The widespread use of SOAP has been accompanied by extensive tool development to support implanting with SOAP, for example, XML parsers, and automated source code generation services from a WSDL file. This greatly assists the development and implementation of SIRI over SOAP.

The SOAP basic mechanism is quite simple: a call is made to a remote service, passing along any necessary parameters, and the answer is sent back to the caller (in some cases, a one-way request is used, meaning that no answer will be provided). The SOAP stack serializes the request's parameters into XML, moves the data to the destination using a transport protocol such as HTTP, receives the response, deserializes the response back into objects, and returns the results to the calling method. SOAP tools handle all the encoding and decoding, even for very complex data types, and bind to the remote object automatically.

10.5.4 SIRI WSDL

10.5.4.1 SIRI WSDL Definition

The SIRI XML set includes three different sets of client and server WSDL bindings, using the two most widely used SOAP encoding styles: RPC/Literal and Document/Literal for WSDL 1.0; and an additional version for WSDL 2.0. Several different mappings into WSDL of the SIRI schema are potentially possible. In addition there are several different approaches to using SOAP. SIRI therefore sets specific reference bindings: it is important to note that a special care was focused on ensuring compatibility and interoperability between all the SIRI WSDL variants (meaning that a RPC/Literal based client can request a Document/literal based producer, a Document/literal based client can request a WSDL 2.0 based producer, or any other combination).

10.5.4.2 WSDL 1.1 encoding styles

In WSDL the binding describes the protocol and data format specification to be used. A WSDL 1.1 SOAP binding can be either a Remote Procedure Call (RPC) style binding or a Document style binding. The binding also describes an encoded or literal data format. This gives four possible models:

— RPC/Encoded;

— RPC/Literal;

— Document/Encoded;

— Document/Literal.

An additional variant, commonly called the 'Document/Literal Wrapped' pattern (actually a refinement pattern of Document/Literal, being mainly a coding style, rather than a new model), tries to bring together the advantages of both RPC/Literal and Document/Literal), giving a total of five binding styles available for a given WSDL file.

NOTE For more information on SOAP and WSDL, please refer to the numerous external sources available (web, books, etc.).

SIRI provides two fully compatible and interoperable WSDL variants: a RPC/Literal variant, and a second one, added with SIRI v2.0, that is a Document/Literal Wrapped on SOAP and WSDL variant. This choice was made for the following reasons;

— These are the two most widely used encoding styles;

— It is possible to have the RPC/Literal style interoperable with a Document/Literal Wrapped style;

— The approach is fully compliant with SIRI v1.0;

— The approach is compliant with the WS-I (Web Service Interoperability Organization) Basic Profile (Version 2.0).

All SIRI WSDL 1.1 variants are WS-I (Web Service Interoperability Organization) compliant and have been successfully tested against the Basic Profile (Version 2.0).

10.5.5 SIRI WSDL structure

The SIRI WSDL provides a single access point for each SIRI service, plus an additional generic access to any SIRI service (+SIRI v2.0). Every functional service access point is named ***GetXXX*** where ***XXX*** is the name of the SIRI service. Communication management services (subscription, heart beat and check status) don't have the ***Get*** prefix. The notification services are named ***NotifyXXX***. In addition, the two main delivery services are available (Stop and Line Delivery).

Table 36 — SIRI Producer functional services

Service Name	Notes
GetProductionTimetable	Request for daily production timetables based on ***ProductionTimetableRequestStructure.***
GetEstimatedTimetable	Line centric request for information about Stop Visits, i.e. arrival and departure at a stop. Based on ***EstimatedTimetableRequestStructure***.
GetStopTimetable	Request for information about the estimated timetable, based on the ***StopTimetableRequestStructure.***
GetStopMonitoring	Stop centric request for information about Stop Visits, i.e. arrivals and departures at a stop, based on the ***StopMonitoringRequestStructure.***
GetMultipleStopMonitoring	Similar to ***GetStopMonitoring*** but for multiple stops. Deprecated from Siri 2.0 (use ***GetSiriService*** instead).
GetVehicleMonitoring	Request for information about Vehicle Movements. Based on the ***VehicleMonitoringRequestStructure.***
GetConnectionTimetable	Request for information about timetabled connections at a stop. Based on the ***ConnectionTimetableRequestStructure.***

GetConnectionMonitoring	Request for information about changes to connections at a stop for Connection Monitoring service. Based on the ***ConnectionMonitoringRequestStructure.***
GetGeneralMessage	Request for information about general information messages affecting stops, vehicles or services; Based on the ***GeneralMessageRequestStructure.***
GetFacilityMonitoring	Request for information about Facilities status. Based on the ***FacilityMonitoringRequestStructure.***
GetSituationExchange	Request for information about Facilities status. Based on the ***SituationExchangeRequestStructure.***
GetSiriService	Request to any Siri service. Based on the ***ServiceRequestStructure.***

Table 37 — SIRI Producer communication management and utility services

Service Name	Notes
Subscribe	Request from Subscriber to Producer for a subscription. Based on the ***SubscriptionRequestStructure.***
DeleteSubscription	Request from Subscriber to Subscription Manager to terminate a subscription. Based on the ***TerminateSubscriptionRequestStructure.***
DataSupply	Request from Consumer to Producer to fetch update previously notified by a Data ready message. Based on the ***DataSupplyRequestStructure.***
CheckStatus	Request from Consumer to Producer to check whether services is working. Based on the ***CheckStatusRequestStructure***.
GetCapabilities	Requests the current capabilities of the server. Based on the ***CapabilitiesRequestStructure.***
StopPointsDiscovery	Requests for stop reference data for use in service requests. Based on the ***StopPointsDiscoveryRequestStructure.***
LinesDiscovery	Requests for line data for use in service requests. Based on the ***LinesDiscoveryRequestStructure.***
ConnectionLinkDiscovery	Requests for connection data for use in service requests. Based on the ***ConnectionLinkDiscoveryRequestStructure.***

Table 38 — SIRI Consumer notification

Service Name	Notes
NotifyDataReady	Request from Producer to Consumer to notify that data update is ready to fetch.
NotifyHeartbeat	Notification from Producer to Consumer to indicate that the service is running normally.
NotifyProductionTimetable	Notification from Producer to Consumer of updated ***ProductionTimetable*** data***.***
NotifyEstimatedTimetable	Notification from Producer to Consumer of updated ***EstimatedTimetable*** data***.***
NotifyStopTimetable	Notification from Producer to Consumer of updated ***StopTimetable*** data***.***
NotifyStopMonitoring	Notification from Producer to Consumer of updated ***StopMonitoring*** data***.***
NotifyVehicleMonitoring	Notification from Producer to Consumer of updated ***VehicleMonitoring*** data***.***
NotifyConnectionTimetable	Notification from Producer to Consumer of updated ***ConnectionTimetable*** data***.***
NotifyConnectionMonitoring	Notification from Producer to Consumer of updated ***ConnectionMonitoring*** data***.***
NotifyGeneralMessage	Notification from Producer to Consumer of updated ***GeneralMessage*** data***.***
NotifyFacilityMonitoring	Notification from Producer to Consumer of updated ***FacilityMonitoring*** data***.***
NotifySituationExchange	Notification from Producer to Consumer of updated ***SituationExchange*** data***.***

It should be noted that the SOAP Fault defined in SIRI 1.0 is still available, but its usage is now deprecated (+SIRI v2.0). SIRI error mechanism has to be used instead.

The following figure provides an overview of the producer WSDL (at this level there is nearly no difference between the RPC/Literal, Document/Literal and WSDL 2.0 variants). The figure displays:

— **Port Type** (named **Interface** in WSDL 2.0) on the left –an abstract set of operations supported by one or more endpoints;

— **Binding** in the middle – a concrete protocol and data format specification for a particular port type;

— **Service** on the right – a collection of related endpoints.

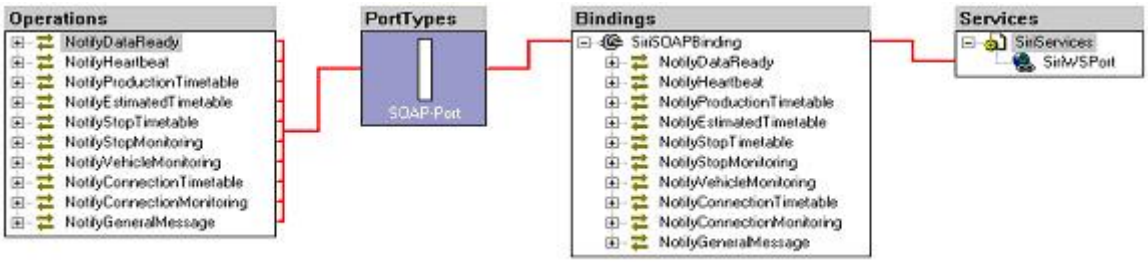

Figure 25 — SIRI SOAP Producer Document/Literal WSDL

The following figure provides an overview of the consumer WSDL.

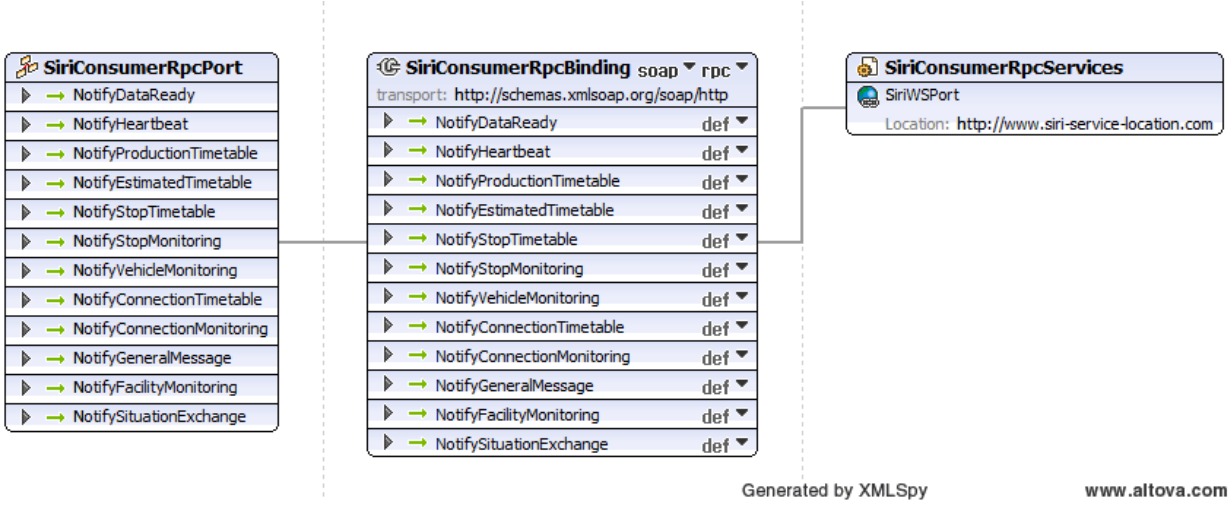

Figure 26 — SIRI SOAP Consumer RPC/Literal WSDL

10.5.6 SIRI RPC WSDL

10.5.6.1 General

The SIRI RPC/Literal WSDL is built directly from types defined in the main XSD file (Siri.xsd): there are no specific extensions for this WSDL definition.

Two SIRI WSDL files are defined, one for the Producer server side, implementing all the supported Producer operations; and one for the Subscriber and Consumer side, implementing all the client notification operations.

Most of the operations have three message parameters: *input*, *output* and *fault*. The *output* parameter is not needed for notification messages (usage of SOAP fault is now deprecated from SIRI 2.0).

Input and output messages have three parts: a generic header (info), the main input or output payload that is specific to the SIRI functional message type, and an extension. The extension is designed to handle parameters which are outside of the scope of SIRI but which may be needed for a specific implementation.

The SIRI RPC WSDL variants were available from SIRI 1.0 and are named *siri_wsProducer.wsdl* for the producer operations, and *siri_wsConsumer.wsdl* for the client operations.

10.5.6.2 WSDL RPC Example: StopTimetable Service

The following figure provides a detailed view of the WSDL for the ***StopTimetable*** service.

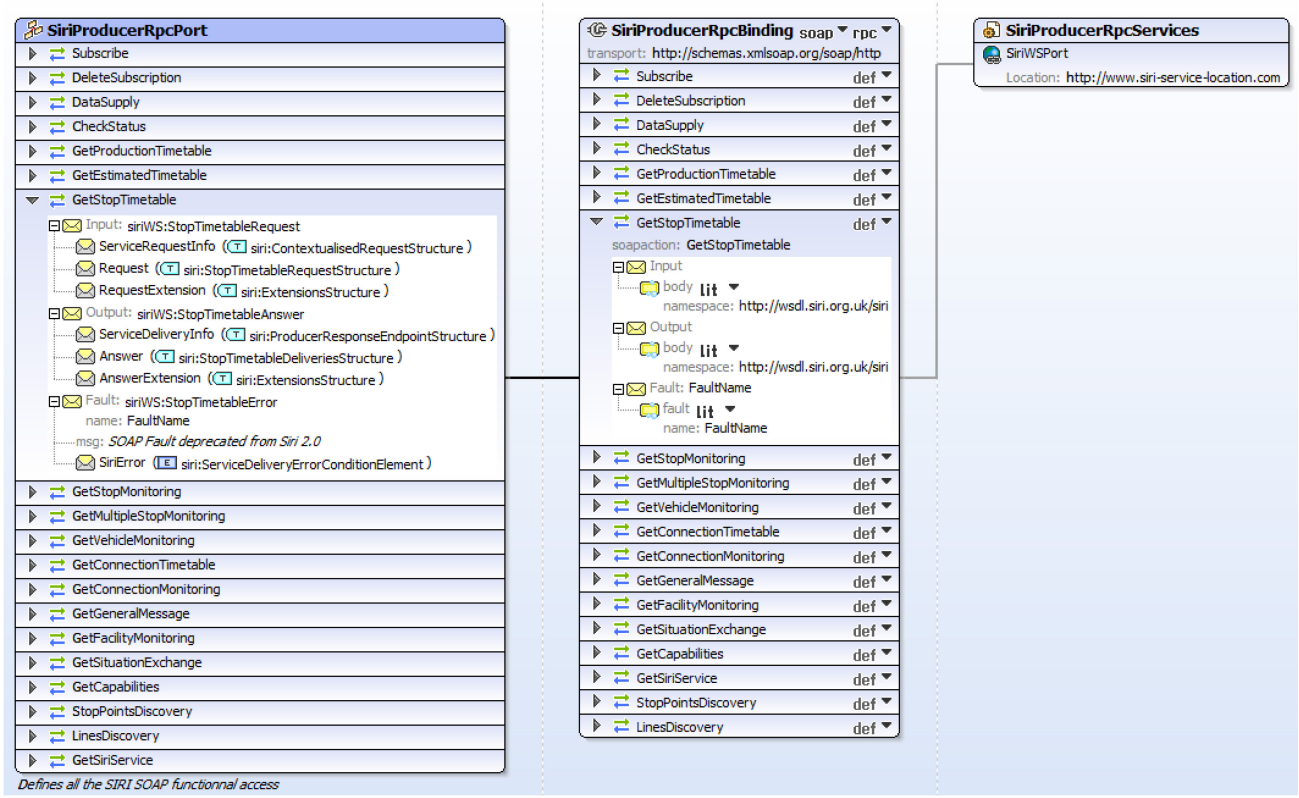

Figure 27 — SIRI SOAP WSDL GetStopTimetable detail

10.5.6.3 SOAP Example: Monitoring Service

The following code fragments provide an example of the application of WSDL for the ***StopMonitoring*** service.

Table 39 — SOAP Example: GetStopMonitoring request

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:siri="http://www.siri.org.uk/siri">
    <SOAP-ENV:Body>
       <GetStopMonitoring xmlns:m="http://wsdl.siri.org.uk/siri">
            <ServiceRequestInfo xmlns="">
              <siri:RequestTimestamp>2010-05-
05T09:17:06.625+02:00</siri:RequestTimestamp>
```

```
            <siri:RequestorRef>CLI_DRYADE</siri:RequestorRef>

   <siri:MessageIdentifier>StopMonitoringClient:Test:0</siri:MessageIdentifier>
          </ServiceRequestInfo>
          <Request version="1.3" xmlns="">
            <siri:RequestTimestamp>2010-05-
05T09:17:06.625+02:00</siri:RequestTimestamp>

   <siri:MessageIdentifier>StopMonitoringClient:Test:0</siri:MessageIdentifier>
            <siri:StartTime>2010-05-05T10:05:00.000+02:00</siri:StartTime>

   <siri:MonitoringRef>DRYADE:StopPoint:BP:15574346:LOC</siri:MonitoringRef>
            <siri:MaximumStopVisits>5</siri:MaximumStopVisits>
            <siri:MaximumNumberOfCalls>
               <siri:Onwards>3</siri:Onwards>
            </siri:MaximumNumberOfCalls>
          </Request>
          <RequestExtension xmlns=""/>
      </GetStopMonitoring>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Table 40 — SOAP Example Message: GetStopMonitoring Answer

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:m0="http://www.siri.org.uk/siri">
   <SOAP-ENV:Body>
      <wsdl:GetStopMonitoringResponse xmlns:wsdl="http://wsdl.siri.org.uk"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
         <ServiceDeliveryInfo>
            <siri:ResponseTimestamp
xmlns:siri="http://www.siri.org.uk/siri">2010-05-
05T09:17:07.657+02:00</siri:ResponseTimestamp>
            <siri:ProducerRef
xmlns:siri="http://www.siri.org.uk/siri">DRYADE</siri:ProducerRef>
            <siri:Address
xmlns:siri="http://www.siri.org.uk/siri">http://localhost:8080/SiriServer</siri:A
ddress>
            <siri:ResponseMessageIdentifier
xmlns:siri="http://www.siri.org.uk/siri">13319</siri:ResponseMessageIdentifier>
            <siri:RequestMessageRef
xmlns:siri="http://www.siri.org.uk/siri">StopMonitoringClient:Test:0</siri:Reques
tMessageRef>
         </ServiceDeliveryInfo>
         <Answer>
            <siri:StopMonitoringDelivery version="1.3"
xmlns:siri="http://www.siri.org.uk/siri">
               <siri:ResponseTimestamp>2010-05-
05T09:17:07.657+02:00</siri:ResponseTimestamp>
               <siri:Status>true</siri:Status>
               <siri:MonitoredStopVisit>
                  <siri:RecordedAtTime>2010-05-
05T09:14:07.666+02:00</siri:RecordedAtTime>
                  <siri:ItemIdentifier>15574363-15574392</siri:ItemIdentifier>
```

```
   <siri:MonitoringRef>DRYADE:StopPoint:BP:15574346:LOC</siri:MonitoringRef>
                  <siri:MonitoredVehicleJourney>
                     <siri:LineRef>DRYADE:Line:15574334:LOC</siri:LineRef>
                     <siri:FramedVehicleJourneyRef>
                        <siri:DataFrameRef>TATROBUS.1.0</siri:DataFrameRef>

   <siri:DatedVehicleJourneyRef>DRYADE:VehicleJourney:15574392:LOC</siri:DatedVe
hicleJourneyRef>
                     </siri:FramedVehicleJourneyRef>

   <siri:JourneyPatternRef>DRYADE:JourneyPattern:15574387:LOC</siri:JourneyPatte
rnRef>
                     <siri:VehicleMode>bus</siri:VehicleMode>
                     <siri:PublishedLineName xml:lang="FR">Ligne 1
BleueBB</siri:PublishedLineName>
                     <siri:DirectionName xml:lang="FR">Les Bucoliques
(A)</siri:DirectionName>
                     <siri:OperatorRef>DRYADE</siri:OperatorRef>

   <siri:OriginRef>DRYADE:StopPoint:SPOR:15574357:LOC</siri:OriginRef>
                     <siri:OriginName>Mairie-1 (A)</siri:OriginName>

   <siri:DestinationRef>DRYADE:StopPoint:SPOR:15574364:LOC</siri:DestinationRef>
                     <siri:DestinationName>Les Bucoliques
(A)</siri:DestinationName>
                     <siri:Monitored>true</siri:Monitored>
                     <siri:InCongestion>false</siri:InCongestion>
                     <siri:InPanic>false</siri:InPanic>
                     <siri:MonitoredCall>

   <siri:StopPointRef>DRYADE:StopPoint:SPOR:15574363:LOC</siri:StopPointRef>
                        <siri:Order>7</siri:Order>
                        <siri:StopPointName xml:lang="FR">Orques et Trolls
(A)</siri:StopPointName>
                        <siri:VehicleAtStop>false</siri:VehicleAtStop>
                        <siri:PlatformTraversal>false</siri:PlatformTraversal>
                        <siri:DestinationDisplay xml:lang="FR">Les Bucoliques
(A)</siri:DestinationDisplay>
                        <siri:AimedDepartureTime>2010-05-
05T10:05:00.000+02:00</siri:AimedDepartureTime>
                        <siri:ExpectedDepartureTime>2010-05-
05T10:08:13.000+02:00</siri:ExpectedDepartureTime>
                        <siri:DepartureStatus>delayed</siri:DepartureStatus>
                     </siri:MonitoredCall>
                     <siri:OnwardCalls>
                        <siri:OnwardCall>

   <siri:StopPointRef>DRYADE:StopPoint:SPOR:15574364:LOC</siri:StopPointRef>
                           <siri:Order>8</siri:Order>
                           <siri:StopPointName xml:lang="FR">Les Bucoliques
(A)</siri:StopPointName>
                           <siri:VehicleAtStop>false</siri:VehicleAtStop>
                           <siri:AimedDepartureTime>2010-05-
05T10:20:00.000+02:00</siri:AimedDepartureTime>
                           <siri:ExpectedDepartureTime>2010-05-
05T10:19:49.000+02:00</siri:ExpectedDepartureTime>
```

```
                                <siri:DepartureStatus>onTime</siri:DepartureStatus>
                            </siri:OnwardCall>
                        </siri:OnwardCalls>
                    </siri:MonitoredVehicleJourney>
                </siri:MonitoredStopVisit>
                <siri:MonitoredStopVisit>
                    <siri:RecordedAtTime>2010-05-
05T09:14:07.667+02:00</siri:RecordedAtTime>
                    <siri:ItemIdentifier>15574370-15574422</siri:ItemIdentifier>

   <siri:MonitoringRef>DRYADE:StopPoint:BP:15574346:LOC</siri:MonitoringRef>
                    <siri:MonitoredVehicleJourney>
                        <siri:LineRef>DRYADE:Line:15574334:LOC</siri:LineRef>
                        <siri:FramedVehicleJourneyRef>
                            <siri:DataFrameRef>TATROBUS.1.0</siri:DataFrameRef>

   <siri:DatedVehicleJourneyRef>DRYADE:VehicleJourney:15574422:LOC</siri:DatedVe
hicleJourneyRef>
                        </siri:FramedVehicleJourneyRef>

   <siri:JourneyPatternRef>DRYADE:JourneyPattern:15574389:LOC</siri:JourneyPatte
rnRef>
                        <siri:VehicleMode>bus</siri:VehicleMode>
                        <siri:PublishedLineName xml:lang="FR">Ligne 1
BleueBB</siri:PublishedLineName>
                        <siri:DirectionName xml:lang="FR">Les Bucoliques
(A)</siri:DirectionName>
                        <siri:OperatorRef>DRYADE</siri:OperatorRef>

   <siri:OriginRef>DRYADE:StopPoint:SPOR:15574365:LOC</siri:OriginRef>
                        <siri:OriginName>La Celeste (A)</siri:OriginName>

   <siri:DestinationRef>DRYADE:StopPoint:SPOR:15574371:LOC</siri:DestinationRef>
                        <siri:DestinationName>Les Bucoliques
(A)</siri:DestinationName>
                        <siri:Monitored>true</siri:Monitored>
                        <siri:InCongestion>false</siri:InCongestion>
                        <siri:InPanic>false</siri:InPanic>
                        <siri:MonitoredCall>

   <siri:StopPointRef>DRYADE:StopPoint:SPOR:15574370:LOC</siri:StopPointRef>
                            <siri:Order>6</siri:Order>
                            <siri:StopPointName xml:lang="FR">Orques et Trolls
(A)</siri:StopPointName>
                            <siri:VehicleAtStop>false</siri:VehicleAtStop>
                            <siri:PlatformTraversal>false</siri:PlatformTraversal>
                            <siri:DestinationDisplay xml:lang="FR">Les Bucoliques
(A)</siri:DestinationDisplay>
                            <siri:AimedDepartureTime>2010-05-
05T10:15:00.000+02:00</siri:AimedDepartureTime>
                            <siri:ExpectedDepartureTime>2010-05-
05T10:13:36.000+02:00</siri:ExpectedDepartureTime>
                            <siri:DepartureStatus>early</siri:DepartureStatus>
                        </siri:MonitoredCall>
                    </siri:MonitoredVehicleJourney>
                </siri:MonitoredStopVisit>
            </siri:StopMonitoringDelivery>
```

```
            </Answer>
            <AnswerExtension/>
        </wsdl:GetStopMonitoringResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

10.5.7 SIRI Document WSDL *(+SIRI v2.0)*

One of the drawbacks of the Document/Literal encoding is that the operation name in the SOAP message is lost. Without the name, dispatching can be difficult or sometimes impossible and compatibility with RPC/Literal is also broken. A workaround, based on Document/Literal Wrapped style, is to have more structured message definitions, including the operation names. Therefore five XSD files have been added.

Table 41 — SOAP Message Structures; XSD files

siri_wsConsumer-Framework.xsd	Structures for consumer communication management.
siri_wsConsumer-Services.xsd	Structures for consumer services (notifications).
siri_wsProducer-DiscoveryCapability.xsd	Structures for discovery services.
siri_wsProducer-Framework.xsd	Structures for producer communication management.
siri_wsProducer-Services.xsd	Structures for producer services.

The SIRI Document WSDL variant was added with SIRI 2.0 and is named *siri_wsProducer-Document.wsdl* for the producer operations, and *siri_wsConsumer-Document.wsdl* for the client operations.

10.5.8 SIRI WSDL 2.0 *(+SIRI v2.0)*

The SIRI WSDL 2.0 variant was generated from the Document/Literal WSDL. It uses the same addition XSD files.

The SIRI WSDL 2.0 variant was added with SIRI 2.0 and the files are named *siri_wsProducer-WSDL2.wsdl* for the producer operations, and *siri_wsConsumer-WSDL2.wsdl* for the client operations.

10.5.9 SIRI WSDL Status

Implementing SIRI as a SOAP Web Service is not mandatory. But if a SIRI SOAP/WSDL implementation is provided, it shall be compliant with the WSDL files provided by SIRI.

11 Capability Discovery Requests

11.1 General

If a SIRI Functional Service supports the Capability Discovery capability, then it is possible to make a ***CapabilityRequest*** to the service and obtain a ***CapabilityResponse*** back with the service's capabilities, detailing exactly which optional SIRI features are supported and which are not.

11.2 Capability Request

Contained in the ***CapabilityRequest*** is a separate service capability request type for each SIRI Functional Service, of the form ***xxxCapabilityRequest***, which returns a response of the form ***xxxCapabilityResponse***.

Table 42 — CapabilityDiscoveryRequest

CapabilityRequest				
Log	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Timestamp on request.
Auth.	***AccountId***	0:1	*+Structure*	Account Identifier. May be used to attribute requests to a particular application provider and authentication key. The account may be common to all users of an application, or to an individual user. Note that to identify an individual user the RequestorRef can be used with an anonymised token. +SIRI v2.0
	AccountKey	0:1	*+Structure*	Authentication key for request. May be used to authenticate requests from a particular account. +SIRI v2.0
Endpoint Properties	***Address***	0:1	*EndpointAddress*	Address to which response is to be sent: [Notify] endpoint. If omitted, this may also be determined from ***RequestorRef*** and preconfigured data, or the http request.
	RequestorRef	**1:1**	*→ParticipantCode*	Identifier of Requestor
	DelegatorAddress	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
	MessageIdentifier	0:1	*MessageQualifier*	Arbitrary identifier that may be given to message.
	Concrete service subscription			
Payload	***ProductionTimetableCapabilityRequest***	0:1	*CapabilityRequestStructure*	Return the capabilities for the Production Timetable Service. See Part 3.
	EstimatedTimetableCapabilityRequest	0:1		Return the capabilities for the Estimated Timetable Service. . See Part 3.
	StopTimetableCapabilityRequest	0:1		Return the capabilities for the Stop Timetable Service. . See Part 3.
	StopMonitoringCapabilityRequest	0:1		Return the capabilities for the Stop Monitoring Service. See Part 3.
	VehicleMonitoringCapabilityRequest	0:1		Return the capabilities for the Vehicle Monitoring Service. See Part 3.
	ConnectionTimetableCapabilityRequest	0:1		Return the capabilities for the Connection Timetable Service. See Part 3.
	ConnectionMonitoringCapabilityRequest	0:1		Return the capabilities for the Connection Monitoring Service. See Part 3.
	GeneralMessage CapabilityRequest	0:1		Return the capabilities for the General Message Service. See Part 3.
	FacilityMonitoringCapabilityRequest	0:1		Return the capabilities for the Facility Monitoring Service. . See Part 4. SIRI v1.3
	SituationExchange CapabilityRequest	0:1		Return the capabilities for the Situation Exchange Service. See Part 5..SIRI v1.3

11.3 Service Capability Discovery

11.3.1 Service Capability Discovery Request — Element

For each SIRI Functional Service, there is a separate request message to discover the exact capabilities of the implemented service. All capability discovery request messages have the same parameters (see Table 47): they differ only in that they have different names of the form ***xxxCapabilityRequest*** where xxx is the Functional Service name.

Table 43 — SIRI Service CapabilityDiscoveryRequest — Attributes

xxxRequest			*+Structure*	SIRI Functional service request for service xxx.
Attributes	***version***	**1:1**	*VersionString*	Version Identifier of Functional Service, e.g. '1.0c'.
Endpoint Properties	***RequestTimestamp***	**1:1**	*xsd:dateTime*	Time of Request.
	MessageIdentifier	0:1	*MessageQualifier*	Arbitrary unique reference to this message.
Options	***ParticipantPermissions***	0:1	*xsd:boolean*	If ***AccessControl*** is supported, whether to include the requesting participant's permissions in the response. Default is *false*.
any	***Extensions***	0:1	*xsd:any**	Placeholder for user extensions.

11.3.2 Service Capability Discovery Response — Element

For each SIRI Functional Service, there is a separate response message to return the service's specific capabilities.

Table 44 — CapabilityDiscoveryResponse — Attributes

CapabilityResponse			*+Structure*	Description of capabilities
Log	***ResponseTimestamp***	**1:1**	*xsd:dateTime*	Timestamp on response.
Endpoint Properties	***ProducerRef***	0:1	*→ParticipantCode*	Participant reference that identifies producer of data. May be available from context.
	Address	0:1	*EndpointAddress*	Address to which any acknowledgment should be sent. Only needed if ***ConfirmDelivery*** specified.
	ResponseMessageIdentifier	0:1	*MessageQualifier*	An arbitrary unique reference associated with the response which may be used to reference it.
	RequestMessageRef	0:1	*→MessageQualifier*	Reference to a unique message identifier associated with the request which gave rise to this response.
Discovery	***DelegatorAddress***	0:1	*EndpointAddress*	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	*→ParticipantCode*	Identifier of delegating system that originated message. +SIRI 2.0
	Concrete service response			
Payload	***ProductionTimetableCapabilityResponse***	0:1	*+Structure*	Capabilities for the Production Timetable Service. See SIRI Part 3.

	EstimatedTimetableCapabilityResponse	0:1	*+Structure*	Capabilities for the Estimated Timetable Service. See SIRI Part 3.
	StopTimetableCapabilityResponse	0:1	*+Structure*	Capabilities for the Stop Timetable Service. See SIRI Part 3.
	StopMonitoringCapabilityResponse	0:1	*+Structure*	Capabilities for the Stop Monitoring Service. See SIRI Part 3.
	VehicleMonitoringCapabilityResponse	0:1	*+Structure*	Capabilities for the Vehicle Monitoring Service. See SIRI Part 3.
	ConnectionTimetableCapabilityResponse	0:1	*+Structure*	Capabilities for the Connection Timetable Service. See SIRI Part 3.
	ConnectionMonitoringCapabilityResponse	0:1	*+Structure*	Capabilities for the Connection Monitoring Service. See SIRI Part 3.
	GeneralMessageCapabilitResponse	0:1	*+Structure*	Capabilities for the General Message Service. See SIRI Part 3.
	FacilityMonitoringCapabilityResponse	0:1	*+Structure*	Capabilities for the Facility Monitoring Service. . See Part 4. SIRI v1.3
	SituationExchange CapabilityResponse	0:1	*+Structure*	Capabilities for the Situation Exchange Service. See Part 5. SIRI v1.3

11.3.3 Functional Service Capability Discovery Response — Element

For each SIRI Functional service there is a separate ***xxxCapabilityResponse*** message to return the service's capabilities. There is different message content for the exact capabilities of each different service; a number of capabilities are common to all services.

Table 45 — SIRI Common Capability Responses

xxxCapabilityResponse			*+Structure*	Response with capabilities for implementation of SIRI XXX service.
Attributes	***version***	**1:1**	*VersionString*	Version Identifier of XXX Service, e.g. '1.0c'.
Log	***ResponseTimestamp***	**1:1**	*xsd:dateTime*	Time individual response element was created.
Endpoint	***RequestMessageRef***	0:1	*→MessageQualifier*	Arbitrary unique reference to the request which gave rise to this message.
Status	***Status***	0:1	*xsd:boolean*	Whether the request could be processed successfully or not. Default is *true*.
	ErrorCondition	0:1	*+Structure*	Description of any error or warning condition.
Payload	***xxxServiceCapabilities***	**1:1**	*+Structure*	Functional Service specific response.
General	***GeneralInteraction***	0:1	*CapabilityGeneralInteractionStructure*	General capabilities common to all SIRI Functional Service request types.
	Interaction	**1:1**	*+Structure*	Interaction capabilities.
	RequestResponse	**1:1**	*xsd:boolean*	Whether the service supports Request Response Interaction. Default is true.
	PublishSubscribe	**1:1**	*xsd:boolean*	Whether the service supports Publish Subscribe Interaction. Default is *true*.
	Delivery	**1:1**	*+Structure*	Interaction capabilities.
	DirectDelivery	**1:1**	*xsd:boolean*	Whether the service supports Direct delivery.
	FetchedDelivery	**1:1**	*xsd:boolean*	Whether the service supports Fetched delivery.

		MultipartDespatch	**1:1**	*xsd:boolean*	Whether the service supports multiple part despatch with MoreData flag. Default is true.
		MultipleSubscriberFilter	**1:1**	*xsd:boolean*	Whether the service supports multiple Subscriber Filters.
		HasConfirmDelivery	**1:1**	*xsd:boolean*	Whether Supports Delivery confirm.
		HasHeartbeat	**1:1**	*xsd:boolean*	Whether the service has a heartbeat message. Default is *false*.
		VisitNumberIsOrder	0:1	*xsd:boolean*	Whether ***VisitNumber*** can be used as a strict order number within JOURNEY PATTERN Default is *false*.
Implementation	***TransportDescription***		0:1	*+Structure*	Nature of Communications Transport protocol
		CommunicationsTransportMethod	**1:1**	*CommunicationsTransportMethodEnum*	Communications Transport method used to exchange messages. Default is '*httpPost*'.
		CompressionMethod	**1:1**	*CompressionMethodEnum*	Method of compression used to optimise transmission of messages.
Payload	***xxxCapabilities***		0:1	*+Structure*	Service Specific Functional capabilities - See Part 3.
	xxxPermissions		0:1	*+Structure*	Service Specific permissions - See Part 3.
any	***Extensions***		0:1	*xsd:any**	Service Specific Capabilities.

11.3.3.1 *CommunicationsTransportMethod* — Allowed values

Allowed values for ***CommunicationsTransportMethod*** (*CommunicationsTransportMethodEnumeration*).

Table 46 — *CommunicationsTransportMethod* — Allowed Values (SIRI 2.0)

Value	Description
httpPost	Transport is using http with XML attachments.
wsdlSoap	Transport is using http with WSDL SOAP RPC.
wsdlSoapDocumentLiteral	Transport is using http with WSDL SOAP Document Literal. +SIRI v2.0.
other	Transport is using other method.
httpUrlJson	Transport is using http with simple URL parameters and JSON response +SIRI v2.0.
httpUrlProtoBuffers	Transport is using http with simple URL parameters and ProtoBuffers response +SIRI v2.0.

11.3.3.2 *CompressionMethod — Allowed values*

Allowed values for ***CompressionMethod*** (*CompressionMethodEnumeration*).

Table 47 — *CompressionMethod* — Allowed Values (SIRI 2.0)

Value	Description
none	No compression method other than that inherent in the protocol is used.
gzip	GZIP compression method is used.
other	Other compression method is used.

11.3.4 Service Capability Response — Example

The following is an example of a Service Capability Response for the SIRI ***Stop Monitoring*** service:

```
<StopMonitoringCapabilitiesResponse …. version="1.0">
    <ResponseTimestamp>2005-11-17T09:30:47-05:00</ResponseTimestamp>
    <RequestMessageRef>12536</RequestMessageRef>
    <StopMonitoringServiceCapabilities>
    <GeneralInteraction>
        <Interaction>
            <RequestResponse>true</RequestResponse>
            <PublishSubscribe>true</PublishSubscribe>
        </Interaction>
        <Delivery>
            <DirectDelivery>true</DirectDelivery>
            <FetchedDelivery>false</FetchedDelivery>
        </Delivery>
        <MultipartDespatch>true</MultipartDespatch>
        <MultipleSubscriberFilter>true</MultipleSubscriberFilter>
        <HasConfirmDelivery>false</HasConfirmDelivery>
        <HasHeartbeat>false</HasHeartbeat>
    </GeneralInteraction>
    <TopicFiltering>
        <DefaultPreviewInterval>PT60M</DefaultPreviewInterval>
        <ByStartTime>true</ByStartTime>
        <FilterByMonitoringRef>true</FilterByMonitoringRef>
        <FilterByLineRef>true</FilterByLineRef>
        <FilterByDirectionRef>true</FilterByDirectionRef>
        <FilterByDestination>false</FilterByDestination>
        <FilterByVisitType>true</FilterByVisitType>
    </TopicFiltering>
    <RequestPolicy>
        <NationalLanguage>en-uk</NationalLanguage>
        <NationalLanguage>de</NationalLanguage>
        <GmlCoordinateFormat>epsg:4326</GmlCoordinateFormat>
        <UseReferences>true</UseReferences>
        <UseNames>false</UseNames>
        <HasDetailLevel>false</HasDetailLevel>
        <DefaultDetailLevel>normal</DefaultDetailLevel>
        <HasMaximumVisits>true</HasMaximumVisits>
        <HasMinimumVisitsPerLine>true</HasMinimumVisitsPerLine>
        <HasNumberOfOnwardsCalls>false</HasNumberOfOnwardsCalls>
        <HasNumberOfPreviousCalls>false</HasNumberOfPreviousCalls>
    </RequestPolicy>
    <SubscriptionPolicy>
        <HasIncrementalUpdates>true</HasIncrementalUpdates>
        <HasChangeSensitivity>true</HasChangeSensitivity>
    </SubscriptionPolicy>
    <AccessControl>
        <RequestChecking>false</RequestChecking>
        <CheckOperatorRef>true</CheckOperatorRef>
        <CheckLineRef>true</CheckLineRef>
        <CheckMonitoringRef>true</CheckMonitoringRef>
    </AccessControl>
    <ResponseFeatures>
        <HasLineNotices>true</HasLineNotices>
    </ResponseFeatures>
```

```
   </StopMonitoringServiceCapabilities>
   <StopMonitoringPermissions>
<!—Block General use -->
      <StopMonitoringPermission>
         <AllParticipants/>
         <GeneralCapabilities>
            <RequestResponse>true</RequestResponse>
            <PublishSubscribe>true</PublishSubscribe>
         </GeneralCapabilities>
         <OperatorPermissions>
            <AllowAll>false</AllowAll>
         </OperatorPermissions>
         <LinePermissions>
            <AllowAll>false</AllowAll>
         </LinePermissions>
         <StopMonitorPermissions>
            <AllowAll>true</AllowAll>
         </StopMonitorPermissions>
      </StopMonitoringPermission>
   <!-- Enable NADER to line 22, 46 & operator 101 -->
      <StopMonitoringPermission>
         <ParticipantRef>NADER</ParticipantRef>
         <GeneralCapabilities>
            <RequestResponse>true</RequestResponse>
            <PublishSubscribe>true</PublishSubscribe>
         </GeneralCapabilities>
         <OperatorPermissions>
            <OperatorPermission>
            <Allow>true</Allow>
            <OperatorRef>101</OperatorRef>
            </OperatorPermission>
         </OperatorPermissions>
         <LinePermissions>
            <LinePermission>
               <Allow/>
               <LineRef>22</LineRef>
            </LinePermission>
            <LinePermission>
               <Allow/>
               <LineRef>46</LineRef>
            </LinePermission>
         </LinePermissions>
         <StopMonitorPermissions>
            <AllowAll>true</AllowAll>
         </StopMonitorPermissions>
      </StopMonitoringPermission>
   </StopMonitoringPermissions>
   </StopMonitoringCapabilitiesResponse>
</ CapabilityResponse>
```

11.4 Functional Service Capability Permission Matrix

11.4.1 Introduction

The ***xxxCapabilityResponse*** message may include a permission matrix indicating the rights of the client to use the functional service. The ***xxxPermission*** element has some common parameters that are the same for

all services, and then specific permissions that apply to the content available through the service. For example, the Stop Monitoring service has permissions for stops, lines and operators.

The permission element can also be used for configuration as part of an access matrix of permissions for all participants.

Table 48 — SIRI Functional Service Common Permission — Attributes

xxxServicePermissions				*+Structure*	SIRI Permission response for service xxx.
Attributes	***PermissionVersion***		**1:1**	*VersionString*	Version Identifier of Permission data set.
Payload	***xxxPermission***		**1:***	*+Structure*	Permission for a service type.
Identity				*choice*	***AllParticipants*** or named participant(s)
	a	***AllParticipants***	**–1:1**	*EmptyType*	Permissions apply by default to All participants. May be overridden by other separate permissions for individual.
	b	***ParticipantRef***		*→ParticipantCode*	Permission applies to specified participant.
General	***GeneralCapability***		**0:1**	*+Structure*	Allowed patterns of interaction.
		RequestResponse	**1:1**	*xsd:boolean*	Participant may make direct requests for data. Default is *true*.
		PublishSubscribe	**1:1**	*xsd:boolean*	Participant may create subscriptions. Defaults to true.
Permissions	{Depends on Specific SIRI Functional Service – See Part 3.}				
any	***Extensions***		0:1	*xsd:any**	Placeholder for user extensions.

11.4.2 OperatorPermissions — Element

The ***OperatorPermissions*** specifies the operators for which a given participant is allowed to access data as subscriber or consumer. The precise usage allowed will depend on the service. For example, on the SIRI Timetable service the permission indicates the operators for whom timetables can be returned.

Table 49 — OperatorPermissions — Attributes

OperatorPermissions				*+Structure*	SIRI Permissions to access operators.
				choice	Choice of ***AllowAll*** or named Operators.
a	***AllowAll***		**–1:1**	*EmptyType*	Participant may access data for all operators.
b	***OperatorPermission***		**–1:***	*+Structure*	Permission to access a specific operator.
		Allow	**1:1**	*xsd:boolean*	Whether the participant is allowed or not allowed to access operator (*true*) or not allowed to access (*false*).
		OperatorRef	**1:1**	*→OperatorCode*	Identifier of operator whose data participant is allowed to access.

11.4.3 LinePermissions — Element

The ***LinePermissions*** specifies the Lines for which a given participant is allowed to access data as subscriber or consumer. The precise usage allowed will depend on the service. For example, on the SIRI Timetable service the permission indicates the lines for which timetables can be returned.

Table 50 — LinePermissions — Attributes

LinePermissions				*+Structure*	SIRI Permissions to access Lines.
				choice	Choice of ***AllowAll*** or named Line(s).
a	***AllowAll***		**–1:1**	*EmptyType*	Participant may access data for all lines.
b	***LinePermission***		**–1:***	*+Structure*	Permission to access a specific line.
		Allow	**1:1**	*xsd:boolean*	Whether the participant is allowed or not allowed to access line (*true*) or not allowed to access (*false*).
		LineRef	**1:1**	*→LineCode*	Identifier of line whose data participant is allowed to access.
		DirectionRef	**0:***	*→DirectionCode*	Identifier of direct of line that participant is allowed to access.

11.4.4 ConnectionLinkPermissions — Element

The ***ConnectionLinkPermissions*** specifies the Connection Links for which a given participant is allowed to access data as subscriber or consumer. The precise usage allowed will depend on the service. For example, on the SIRI Connection Timetable service the permission indicates the Connection Links for which timetables can be returned.

Table 51 — ConnectionLinkPermissions — Attributes

ConnectionLinkPermissions				*+Structure*	SIRI Permissions to access connection links.
				choice	One of ***AllowAll*** or named Connection Link(s).
a	***AllowAll***		**–1:1**	*EmptyType*	Participant may access data for all connection links.
b	***ConnectionLinkPermission***		**–1:***	*+Structure*	Permission to access a specific connection link.
		Allow	**1:1**	*xsd:boolean*	Whether the participant is allowed or not allowed to access connection link (true) or not allowed to access (*false*).
		ConnectionLinkRef	**1:1**	*→Connection LinkCode*	Identifier of connection link whose data participant is allowed to access.

11.4.5 StopMonitorPermissions — Element

The ***StopMonitorPermissions*** specifies the Monitoring points (LOGICAL DISPLAYs) for which a given participant is allowed to access data as subscriber or consumer. The precise usage allowed will depend on the service. For example, on the SIRI Stop Timetable service the permission indicates the SCHEDULED STOP POINTs or LOGICAL DISPLAYs for which Stop Timetables can be returned.

Table 52 — StopMonitorPermissions — Attributes

StopMonitorPermissions			*+Structure*	SIRI Permissions to access monitoring point (LOGICAL DISPLAY).
			choice	One of ***AllowAll*** or named Monitoring point(s).
a	***AllowAll***	**–1:1**	*EmptyType*	Participant may access data for all monitoring points.
b	***StopMonitorPermission***	**–1:***	*+Structure*	Permission to access a specific monitoring point.
	Allow	**1:1**	*xsd:boolean*	Whether the participant is allowed or not allowed to access monitoring point (true) or not allowed to access (*false*).
	MonitoringRef	**1:1**	*→MonitoringCode*	Identifier of monitoring point whose data participant is allowed to access.

11.4.6 VehicleMonitorPermissions — Element

The ***VehicleMonitorPermissions*** specifies the Vehicle Monitoring references for which a given participant is allowed to access data as subscriber or consumer. On the SIRI Vehicle Monitoring service the permission indicates the Vehicle Monitoring references for which responses can be returned.

Table 53 — VehicleMonitorPermissions — Attributes

VehicleMonitorPermissions			*+Structure*	SIRI Permissions to access vehicle monitoring references.
			choice	One of ***AllowAll*** or named Monitoring reference(s).
a	***AllowAll***	**–1:1**	*EmptyType*	Participant may access data for all monitoring references.
b	***VehicleMonitorPermission***	**–1:***	*+Structure*	Permission to access a specific vehicle monitoring reference.
	Allow	**1:1**	*xsd:boolean*	Whether the participant is allowed or not allowed to access monitoring reference (true) or not allowed to access (*false*).
	VehicleMonitoringRef	**1:1**	*→VehicleMonitoringCode*	Identifier of vehicle monitoring reference whose data participant is allowed to access.

11.4.7 InfoChannelPermissions — Element

The ***InfoChannelPermissions*** specifies the ***InfoChannel*** instances for which a given participant is allowed to access data as subscriber or consumer. On the SIRI General message service the permission indicates the Info Channels for which responses can be returned.

Table 54 — InfoChannelPermissions — Attributes

InfoChannelPermissions			*+Structure*	SIRI Permissions to access info channel references.
			choice	One of ***AllowAll*** or named Channels.
a	***AllowAll***	**–1:1**	*EmptyType*	Participant may access data for all info channels.
b	***InfoChannelPermission***	**–1:***	*+Structure*	Permission to access a specific info channel reference.
	Allow	**1:1**	*xsd:boolean*	Whether the participant is allowed or not allowed to access info channel (true) or not allowed to access (*false*).
	InfoChannelRef	**1:1**	*→InfoChannelCode*	Identifier of info channel whose data participant is allowed to access.

12 SIRI for Simple Web Services – SIRI Lite (+SIRI v2.0)

12.1 Introduction

12.1.1 General

Version 1.0 of SIRI was intended primarily for server-to-server communication between the real-time operations systems of transport companies. The growth of public internet use – and in particular of smartphones using the mobile internet – has led to the additional requirement for direct delivery of data to end user devices, as well as a need for massive scalability. The SIRI Simple Web services ('SIRI Lite') address these needs by providing an alternative transport protocol that uses a payload model derived automatically from SIRI.

SIRI Lite has the following objectives:

— To improve scalability by decreasing bandwidth consumption,

— To improve scalability by decreasing the processing power required to process requests on both the server and the user device,

— To lower the cost of integration for application developers by making it easier to develop with SIRI using mainstream technology platforms for mobile devices,

— To support common end user application use cases,

— To minimize the need, if any, to change the existing SIRI schema and data model,

— To enable the use of automatic transforms drawing on the structure of the SIRI model;

The SIRI Simple Web services are summarised as follows:

1) In principle a SIRI-Lite protocol can be used with any SIRI Functional Service, and any interaction pattern, but it is envisaged that it is particularly relevant for Request/Response interactions for the SIRI-SM, SIRI-VM and the SIRI-SX services (and for the Stop & Line discovery services which can be used to provide reference data to support the use of those services).

2) The SIRI-Lite protocol communication transport protocol is used as follows:

— **Requests are specified as http URL parameters**, the endpoint indicating the format to use for the response.

— **Responses are returned as http responses**, encoded in the lightweight format specified on the request endpoint, for example JSON, or binary XML. See alternative response encodings below.

3) **Normal SIRI request filtering is supported.** It is important to be able to minimize the data that needs to be transmitted; especially for wireless applications and so it is useful to support server-side filtering of the response data.

4) **The normal SIRI XML data types are used**, simplifying the transforms needed to create alternative responses from an existing SIRI service.

— There is one additional refinement permitted: to simplify processing for mobile devices an optional request parameter to map duration and timestamp formats to UNIX formats is supported.

5) The **normal exception conditions** are supported for when data or the service are not available and are returned in the responses

6) **Additional authentication elements** on the request are supported to identify the requesting service. These are added to all requests the SIRI v2.0 framework and are available for use in the SIRI Lite services.

12.1.2 Existing Implementations

The design of the SIRI Lite services is informed by the experience of a number of members of the world-wide SIRI community, in particular that of the Metropolitan Transportation Authority (MTA), the primary public transport operator for New York City. MTA is using SIRI with similar extensions described in this document to support all distribution of a large real-time bus tracking and customer information system.

12.1.3 Using SIRI-LITE services in combination

12.1.3.1 General

It is envisaged that a Consumer application will use a sequence of different SIRI-LITE requests to successively obtain just the data of specific interest to the user. This is illustrated in the following two use cases for obtaining passenger information on an end user device.

NOTE The examples assume that the Consumer application will have previously discovered the endpoints for accessing the required SIRI services. Discovery is outside the SIRI standard but can be done routinely with generic discovery services.

12.1.3.2 Providing real-time Stop Arrivals & Departures – Use Case for SIRI LITE

a) Use device location or address finder service to locate user.

b) Use SIRI Stop Point discovery service to find available stops in an area.

c) Display stops to user.

d) User selects a stop.

e) Use SIRI-SM Stop Monitoring services to get visits for a selected stop.

f) Display the arrivals and departures at stop to the user.

g) User selects a journey.

h) Use SIRI-VM Vehicle Monitoring to get detailed calling pattern for a selected journey.

i) Display calling pattern of selected journey to user.

12.1.3.3 Vehicle positions – Use Case for SIRI LITE

a) Use device location or address finder service to locate user.

b) Use SIRI Line discovery service to find available lines in an area.

c) Display lines to user.

d) User selects a line.

e) Use SIRI-VM Vehicle Monitoring services to get just vehicle positions for a selected line.

f) Display positions of vehicles to user on a spatial visualisation on the device.

g) User selects a vehicle.

h) Use SIRI-VM Vehicle Monitoring services to get journey with predicted times for a selected line.

i) Display calling pattern of selected journey to user.

12.1.4 Alternative Response Encoding

The response of a SIRI Lite service is encoded in an agreed technology format chosen for speed and efficiency for serialization as payload of an http request. In this document we use JSON (*JavaScript Object Notation*) as the preferred format for examples, but the SIRI LITE need not be restricted to a single technology.

Table 55 – Alternative Response Encodings for SIRI Simple Web Services

Response Encoding	Description	Reference
XML	Plain XML	http://www.w3.org/XML
XML EXI *XML Efficient XML Interchange*	Encodings and/or transformations of XML to enhance processing speed and efficiency	http://www.w3.org/XML/EXI/
JSON: *JavaScript Object Notation*	A loosely typed format. More space efficient than XML and very commonly used for public facing web services, including those used by in-browser and native mobile phone applications. Some XML binding libraries have the ability to output as JSON as well as XML.	http://www.json.org/
Google ***Protocol Buffers:***	A strongly typed and bound encoding that is very space and speed efficient for serializing structured data	http://code.google.com/p/protobuf/
Apache **Thrift**	An Open source version of buffer.	http://thrift.apache.org/
Fast Infoset	The Fast Infoset technology provides an alternative to W3C XML syntax as a means of representing instances of the W3C XML Information Set.	http://www.itu.int/ITU-T/asn1/xml/finf.htm

12.1.5 Lossless transforms

Under each of the above cases, the data model and types *do not change*. When SIRI is decoded from a rendering any of the above formats, the original SIRI data model and types are retained in full.

12.1.6 Simple transforms

An encoding style that favours straightforward and automated mapping of the XML structures is preferred (e.g. for JSON nested structures rather than arrays).

12.2 Encoding of URL Requests

12.2.1 Complete Request Encoding in HTTP URL's

Any SIRI request may be formulated under SIRI Simple Web Services entirely within the scope of an HTTP URL - rather than as an XML document attached to the http request, as in the normal SIRI usage.

For example:

```
http://server.siri.com/path/to/api/vehicle-monitoring.xml?LineRef=Line1
```

The SIRI LITE request includes:

— Which service is being requested (e.g. ***StopMonitoring***, ***VehicleMonitoring***)

— Parameters of the request (i.e. topics and policies)

— Desired response encoding (when alternatives to XML are available).

12.2.2 General format of SIRI Lite request URL

SIRI LITE requests follow a generalized URL structure of the form:

endpoint?query_string

Where endpoint is made up of the normal http components,

scheme://host:port/path?query_string

Thus the overall elements are:

— scheme: http or https.

— host: the hostname or IP address of the service.

— port: the IP port of the service (default 80 for http, 443 for https).

— path: the hierarchical portion of the URL indicating the SIRI functional service and version

— query string: parameters of the request

For example:

http://www.tfl.gov.uk/siri/2.0/stop-monitoring.json?MonitoringRef=ABCD

12.2.3 Endpoints and Service Identification

The path part of the request is used to indicate both the functional service (SIRI-SM, SIRI-VM, etc.) being requested, the API version, and the encoding to be used.

For example, Stop Monitoring encoded with JSON:

http://www.tfl.gov.uk/siri/2.0/stop-monitoring.json?MonitoringRef=ABCD

For example, Vehicle Monitoring encoded with XML EXI:

http://www.tfl.gov.uk/siri/2.0/vehicle-monitoring.exi?LineRef=Line1

Thus for each SIRI functional service and encoding combination, a separate endpoint shall be provided: the actual structure of the path used does not need to be prescribed by the standard, but it is recommended that it also includes a version number to allow for concurrent support of different versions.

12.2.4 Encoding of Service Parameters on http request

Request parameters from a SIRI functional request are included directly in the query string of the request URL.

Information about any XML groups used to organize the parameters within the XML request, such as Topics or Policies group should be omitted.

For example, for a ***StopMonitoringRequest***, a valid URL might be:

http://www.vbb.de/siri/2.0/stop-monitoring.json?MonitoringRef=4565&LineRef=X56&StopMonitoringDetailLevel=calls

In this example, ***MonitoringRef*** and ***LineRef*** both come from the ***StopMonitoringTopicGroup***, while ***StopMonitoringDetailLevel*** comes from the ***StopMonitoringRequestPolicyGroup***.

NOTE Older implementations of HTTP clients or servers may have limits on how long a valid URI can be.

12.2.5 Naming of Request Parameters with Hierarchy

Most request topics or policies parameters are “simple”, comprised of a single element, but a few are complex i.e. have nested sub-elements. In the case of a complex element, the parameter name should be constructed by concatenating the element names, separating the two by a period.

12.2.6 Naming of Parameters with Plural Cardinality

Some request topics or policies can have a cardinality of more than one. In this case, the parameter should be repeated for each element.

For example, ***SituationExchange*** can have multiple ***LineRef*** topics (in the ***SituationNetworkFilterGroup***). A valid URL that specifies multiple ***LineRef*** elements might be:

http://ratp.fr/2.0/situation-exchange.xml?LineRef=M13&LineRef=M8&LineRef=M13

12.2.7 Handling of invalid request combinations

The normal XML encoding for a SIRI functional service includes validation of data types and also enforces constraints that prevent certain combinations of parameters that are not allowed or meaningless. The direct encoding does not prevent such combinations. A SIRI-LITE function service shall enforce the same constraints and return an appropriate error message if it detects an illegal combination (for example ***ParametersIgnored, UnknownExtensions, InvalidDataReferences)***.

12.2.8 Specifying the encoding of the Response

The encoding of the response format is given by the choice of endpoint (and may be indicated as part of the path component of the URL), **not** as a parameter in the query string.

For example, an XML response might be retrieved using an endpoint:

http://server.siri.com/path/to/api/vehicle-monitoring.xml?LineRef=Line1

Or a JSON response:

http://server.siri.com/path/to/api/vehicle-monitoring.json?LineRef=Line1

12.3 Examples

12.3.1 General

In the following examples we show SIRI-LITE encodings for XML and JSON.

12.3.2 SIRI-SM Simple Stop Monitoring request to fetch stop departures – SIRI LITE Examples

12.3.2.1 General

The following example shows the request response pair to produce a JSON response show to basic Stop Departures for a stop using the SIRI-SM Stop Monitoring Service.

12.3.2.2 Simple Stop Monitoring request to fetch stop departures – XML Example

The following code fragment shows the XML attachment for a SIRI XML request for a SIRI-SM service.

```
<ServiceRequest>
   <RequestTimestamp>2012-06-15T12:46:05-04:00</RequestTimestamp>
   <StopMonitoringRequest version="2.0">
      <RequestTimestamp>2012-06-15T12:46:05-04:00</RequestTimestamp>
      <!--=======TOPIC ==================================== -->
      <MonitoringRef>305453</MonitoringRef>
   </StopMonitoringRequest>
</ServiceRequest>
```

12.3.2.3 Simple Stop Monitoring request to return stop departures – JSON Example

The following code fragment shows the equivalent request translating the XML parameters into request parameters for an SIRI Lite SIRI-SM service.

http://bustime.mta.info/api/siri/stop-monitoring.json?MonitoringRef=305453

12.3.2.4 Simple Stop Monitoring response to return stop departures – XML Example

The following code fragment shows the XML document for a SIRI XML response for a SIRI-SM service.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Siri xmlns:ns2="http://www.ifopt.org.uk/acsb"
xmlns:ns4="http://datex2.eu/schema/1_0/1_0"
xmlns:ns3="http://www.ifopt.org.uk/ifopt" xmlns="http://www.siri.org.uk/siri">
   <ServiceDelivery>
      <ResponseTimestamp>2012-06-15T12:47:24.675-04:00</ResponseTimestamp>
      <StopMonitoringDelivery version="2.0">
         <ResponseTimestamp>2012-06-15T12:47:24.675-04:00</ResponseTimestamp>
         <ValidUntil>2012-06-15T12:48:24.675-04:00</ValidUntil>
         <MonitoredStopVisit>
            <RecordedAtTime>2012-06-15T12:46:57-04:00</RecordedAtTime>
            <MonitoredVehicleJourney>
               <LineRef>B63</LineRef>
               <DirectionRef>1</DirectionRef>
               <FramedVehicleJourneyRef>
                  <DataFrameRef>2012-06-15</DataFrameRef>

   <DatedVehicleJourneyRef>20120408EA_071100_B63_0113_B35_5</DatedVehicleJourney
Ref>
               </FramedVehicleJourneyRef>
               <JourneyPatternRef>B630113</JourneyPatternRef>
               <PublishedLineName>B63</PublishedLineName>
               <OperatorRef>MTA NYCT</OperatorRef>
               <OriginRef>801131</OriginRef>
               <DestinationRef>801042</DestinationRef>
               <DestinationName>BAY RIDGE SHORE RD via 5 AV</DestinationName>
```

```
                <Monitored>true</Monitored>
                <VehicleLocation>
                   <Longitude>-74.012167</Longitude>
                   <Latitude>40.64333</Latitude>
                </VehicleLocation>
                <VehicleRef>7589</VehicleRef>
                <MonitoredCall>
                   <StopPointRef>305453</StopPointRef>
                   <VisitNumber>1</VisitNumber>
                   <StopPointName>5 AV - BAY RIDGE AV</StopPointName>
                   <ExpectedDepartureTime>2012-06-15T12:49:57-
04:00</ExpectedDepartureTime>
                </MonitoredCall>
             </MonitoredVehicleJourney>
          </MonitoredStopVisit>
          <MonitoredStopVisit>
             <RecordedAtTime>2012-06-15T12:47:00-04:00</RecordedAtTime>
             <MonitoredVehicleJourney>
                <LineRef>B63</LineRef>
                <DirectionRef>1</DirectionRef>
                <FramedVehicleJourneyRef>
                   <DataFrameRef>2012-06-15</DataFrameRef>

   <DatedVehicleJourneyRef>20120408EA_072300_B63_0113_B70_204</DatedVehicleJourn
eyRef>
                </FramedVehicleJourneyRef>
                <JourneyPatternRef>B630113</JourneyPatternRef>
                <PublishedLineName>B63</PublishedLineName>
                <OperatorRef>MTA NYCT</OperatorRef>
                <OriginRef>801131</OriginRef>
                <DestinationRef>801042</DestinationRef>
                <DestinationName>BAY RIDGE SHORE RD via 5 AV</DestinationName>
                <Monitored>true</Monitored>
                <VehicleLocation>
                   <Longitude>-74.000672</Longitude>
                   <Latitude>40.654379</Latitude>
                </VehicleLocation>
                <VehicleRef>7583</VehicleRef>
                <MonitoredCall>
                   <StopPointRef>305453</StopPointRef>
                   <VisitNumber>1</VisitNumber>
                   <StopPointName>5 AV - BAY RIDGE AV</StopPointName>
                   <ExpectedDepartureTime>2012-06-15T12:55:57-
04:00</ExpectedDepartureTime>
                </MonitoredCall>
             </MonitoredVehicleJourney>
          </MonitoredStopVisit>
       </StopMonitoringDelivery>
   </ServiceDelivery>
</Siri>
```

12.3.2.5 Simple Stop Monitoring response to return stop departures – JSON Example

The following code fragment shows the equivalent response translating the SIRI-SM XML document into JSON value pairs.

```
{
```

```
"Siri":{
   "ServiceDelivery":{
       "ResponseTimestamp":"2012-06-15T12:47:24.675-04:00",
       "StopMonitoringDelivery":[
            {
                "MonitoredStopVisit":[
                     {
                          "MonitoredVehicleJourney":{
                               "LineRef":"B63",
                               "DirectionRef":"1",
                               "FramedVehicleJourneyRef":{
                                   "DataFrameRef":"2012-06-15",
"DatedVehicleJourneyRef":"20120408EA_071100_B63_0113_B35_5"
                               },
                               "JourneyPatternRef":"B630113",
                               "PublishedLineName":"B63",
                               "OperatorRef":"MTA NYCT",
                               "OriginRef":"801131",
                               "DestinationRef":"801042",
                               "DestinationName":"BAY RIDGE SHORE RD via 5
AV",
                               "Monitored":true,
                               "VehicleLocation":{
                                   "Longitude":-74.018271,
                                   "Latitude":40.637455
                               },
                               "VehicleRef":"7589",
                               "MonitoredCall":{
                                   "StopPointRef":"305453",
                                   "VisitNumber":1,
                                   "StopPointName":"5 AV - BAY RIDGE AV",
                                   "ExpectedDepartureTime": "2012-06-
15T12:49:57-04:00"
                               }
                          },
                          "RecordedAtTime":"2012-06-15T12:46:57-04:00"
                     },
                     {
                          "MonitoredVehicleJourney":{
                               "LineRef":"B63",
                               "DirectionRef":"1",
                               "FramedVehicleJourneyRef":{
                                   "DataFrameRef":"2012-06-15",

"DatedVehicleJourneyRef":"20120408EA_072300_B63_0113_B70_204"
                               },
                               "JourneyPatternRef":"B630113",
                               "PublishedLineName":"B63",
                               "OperatorRef":"MTA NYCT",
                               "OriginRef":"801131",
                               "DestinationRef":"801042",
                               "DestinationName":"BAY RIDGE SHORE RD via 5
AV",
                               "Monitored":true,
                               "VehicleLocation":{
                                   "Longitude":-74.010629,
                                   "Latitude":40.644808
```

```
                                    },
                                    "VehicleRef":"7583",
                                    "MonitoredCall":{
                                        "StopPointRef":"305453",
                                        "VisitNumber":1,
                                        "StopPointName":"5 AV - BAY RIDGE AV"
                                        "ExpectedDepartureTime": "2012-06-
15T12:55:57-04:00"
                                    }
                                },
                            "RecordedAtTime":"2012-06-15T12:56:15.000-04:00"
                            },
                    ],
                     "ResponseTimestamp":"2012-06-15T12:47:24.675-04:00",
                     "ValidUntil":"2012-06-15T12:48:24.675-04:00"
                }
            ]
    }
}
}
```

12.3.3 SIRI-VM Simple Vehicle Monitoring request to fetch vehicle positions – SIRI Lite Examples

12.3.3.1 General

The following example shows the request response pair to produce a JSON response for a SIRI-VM service to fetch a MONITORED VEHICLE JOURNEY.

The request also includes authentication parameters.

— ***AccountId: AppDev123 - An account for a given device application provider.***

— ***AccountKey: 123ABC - An authentication key specific to the account for a given device application provider.***

The authentication details may be used to track requests by account. To support the tracking of requests by individual users, subject to user consent, may be used.

— ***RequestorRef*** :X4573 - an anonymised token identifying the user.

12.3.3.2 Simple Vehicle Monitoring request to fetch vehicle positions – XML Example

The following code fragment shows the XML attachment for a SIRI XML request for a SIRI-VM service.

```
<ServiceRequest>
    <!--======ENDPOINT REFERENCES================================-->
    <RequestTimestamp>2012-06-15T13:23:04-04:00</RequestTimestamp>
    <AccountId>Client1</AccountKey>
    <AccountKey>123ABC</AccountKey>
    <VehicleMonitoringRequest version="2.0">
        <RequestTimestamp>2012-06-15T13:23:04-04:00</RequestTimestamp>
        <!--========TOPIC ==================================== -->
        <VehicleRef>7574</VehicleRef>
        <!--========POLICY=========================================-->
        <VehicleMonitoringDetailLevel>basic</VehicleMonitoringDetailLevel>
    </VehicleMonitoringRequest>
```

12.3.3.3 Simple Vehicle Monitoring request to fetch vehicle positions – JSON Example

The following code fragment shows the equivalent request translating the XML parameters into request parameters for an SIRI Lite SIRI-VM service.

```
http://www.bahn.nl/siri/2.0/vehicle-monitoring.json?VehicleMonitoringRef=7574
&VehicleMonitoringDetailLevel=basic&AccountId=AppDev123&AccountKey=123ABC
```

12.3.3.4 Simple Vehicle Monitoring response to return vehicle positions – XML Example

The following code fragment shows the XML document for a SIRI XML response for a SIRI-VM service.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Siri xmlns:ns2="http://www.ifopt.org.uk/acsb"
xmlns:ns4="http://datex2.eu/schema/1_0/1_0"
xmlns:ns3="http://www.ifopt.org.uk/ifopt" xmlns="http://www.siri.org.uk/siri">
    <ServiceDelivery version="2.0">
        <ResponseTimestamp>2012-06-15T13:23:05.627-04:00</ResponseTimestamp>
        <VehicleMonitoringDelivery>
            <ResponseTimestamp>2012-06-15T13:23:05.627-04:00</ResponseTimestamp>
            <ValidUntil>2012-06-15T13:24:05.627-04:00</ValidUntil>
            <VehicleActivity>
                <RecordedAtTime>2012-06-15T13:22:47-04:00</RecordedAtTime>
                <MonitoredVehicleJourney>
                    <LineRef>B63</LineRef>
                    <DirectionRef>1</DirectionRef>
                    <FramedVehicleJourneyRef>
                        <DataFrameRef>2012-06-15</DataFrameRef>

    <DatedVehicleJourneyRef>20120408EA_075900_B63_0113_B70_213</DatedVehicleJourn
eyRef>
                    </FramedVehicleJourneyRef>
                    <JourneyPatternRef>B630113</JourneyPatternRef>
                    <PublishedLineName>B63</PublishedLineName>
                    <OperatorRef>MTA NYCT</OperatorRef>
                    <OriginRef>801131</OriginRef>
                    <DestinationRef>801042</DestinationRef>
                    <DestinationName>BAY RIDGE SHORE RD via 5 AV</DestinationName>
                    <Monitored>true</Monitored>
                    <VehicleLocation>
                        <Longitude>-73.992923</Longitude>
                        <Latitude>40.661826</Latitude>
                    </VehicleLocation>
                    <Bearing>223.65405</Bearing>
                    <ProgressRate>normalProgress</ProgressRate>
                    <VehicleRef>7574</VehicleRef>
                    <MonitoredCall>
                        <StopPointRef>308335</StopPointRef>
                        <VisitNumber>1</VisitNumber>
                        <StopPointName>5 AV - 21 ST</StopPointName>
                        <ExpectedDepartureTime>2012-06-15T13:23:15-
04:00</ExpectedDepartureTime>
                    </MonitoredCall>
                </MonitoredVehicleJourney>
            </VehicleActivity>
        </VehicleMonitoringDelivery>
    </ServiceDelivery>
</Siri>
```

12.3.3.5 Simple Vehicle Monitoring response to return vehicle positions – JSON Example

The following code fragment shows the equivalent response translating the SIRI-VM XML document into JSON value pairs.

```
{
"Siri":{
    "ServiceDelivery":{
        "ResponseTimestamp":"2012-06-15T13:23:06.591-04:00",
        "VehicleMonitoringDelivery":[
              {
                  "VehicleActivity":[
                        {
                            "MonitoredVehicleJourney":{
                                  "LineRef":"B63",
                                  "DirectionRef":"1",
                                  "FramedVehicleJourneyRef":{
                                      "DataFrameRef":"2012-06-15",

"DatedVehicleJourneyRef":"20120408EA_075900_B63_0113_B70_213"
                                  },
                                  "JourneyPatternRef":"B630113",
                                  "PublishedLineName":"B63",
                                  "OperatorRef":"MTA NYCT",
                                  "OriginRef":"801131",
                                  "DestinationRef":"801042",
                                  "DestinationName":"BAY RIDGE SHORE RD via 5
AV",
                                  "Monitored":true,
                                  "VehicleLocation":{
                                      "Longitude":-73.992923,
                                      "Latitude":40.661826
                                  },
                                  "Bearing":223.65405,
                                  "ProgressRate":"normalProgress",
                                  "VehicleRef":"7574",
                                  "MonitoredCall":{
                                      "StopPointRef":"308335",
                                      "VisitNumber":1,
                                      "StopPointName":"5 AV - 21 ST",
                                      "ExpectedDepartureTime":"2012-06-
15T13:23:15-04:00"
                                  }
                            },
                            "RecordedAtTime":"2012-06-15T13:22:47.000-04:00"
                        }
                  ],
                  "ResponseTimestamp":"2012-06-15T13:23:06.591-04:00",
                  "ValidUntil":"2012-06-15T13:24:06.591-04:00"
              }
        ]
    }
}
}
```

12.3.4 SIRI-VM Complex Vehicle Monitoring to obtain journeys – SIRI Lite Examples

12.3.4.1 General

The following example shows the request response pair to produce a JSON response of vehicle predictions, including position and onward calling pattern, for a LINE. It limits the number of onward calls to 2. There are also additional Situation Exchange elements in the same delivery.

12.3.4.2 Complex Vehicle Monitoring request to fetch monitored journeys – XML Example

The following code fragment shows the XML attachment for a SIRI XML request for a SIRI-VM service.

```
    <ServiceRequest>
        <!--=======ENDPOINT REFERENCES================================-->
        <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
        <VehicleMonitoringRequest version="2.0">
            <RequestTimestamp>2004-12-17T09:30:47-05:00</RequestTimestamp>
            <!--=======TOPIC ==================================== -->
            <LineRef>B63</ LineRef >
            <DirectionRef>1</DirectionRef>
            <!--=======POLICY=============================================-->
            <VehicleMonitoringDetailLevel>calls</VehicleMonitoringDetailLevel>
            <MaximumNumberOfCalls>
<Onwards>2</Onwards>
            </MaximumNumberOfCalls>
        </VehicleMonitoringRequest>
```

12.3.4.3 Complex Vehicle Monitoring request to fetch monitored journeys – RESTful Example

http://bustime.mta.info/api/siri/vehicle-monitoring.xml?LineRef=B63&DirectionRef=1&VehicleMonitoringDetailLevel=calls&MaximumNumberOfCalls.Onwards=2

12.3.4.4 Complex Vehicle Monitoring response to return monitored journeys – XML Example

The following code fragment shows the XML document for a SIRI XML response for a SIRI-VM service.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Siri xmlns:ns2="http://www.ifopt.org.uk/acsb"
xmlns:ns4="http://datex2.eu/schema/1_0/1_0"
xmlns:ns3="http://www.ifopt.org.uk/ifopt" xmlns="http://www.siri.org.uk/siri"
version="2.0">
    <ServiceDelivery>
        <ResponseTimestamp>2012-06-15T13:40:20.738-04:00</ResponseTimestamp>
        <VehicleMonitoringDelivery version="2.0">
            <ResponseTimestamp>2012-06-15T13:40:20.738-04:00</ResponseTimestamp>
            <ValidUntil>2012-06-15T13:41:20.738-04:00</ValidUntil>
            <VehicleActivity>
                <RecordedAtTime>2012-06-15T13:40:05.900-04:00</RecordedAtTime>
                <MonitoredVehicleJourney>
                    <LineRef>S78</LineRef>
                    <DirectionRef>1</DirectionRef>
                    <FramedVehicleJourneyRef>
                        <DataFrameRef>2012-06-15</DataFrameRef>

    <DatedVehicleJourneyRef>20120408EA_079500_S78_0470_MISC_890</DatedVehicleJour
neyRef>
```

```
                </FramedVehicleJourneyRef>
                <JourneyPatternRef>S780470</JourneyPatternRef>
                <PublishedLineName>S78</PublishedLineName>
                <OperatorRef>MTA NYCT</OperatorRef>
                <OriginRef>805164</OriginRef>
                <DestinationRef>905179</DestinationRef>
                <DestinationName>BRICKTOWN MALL</DestinationName>
                <SituationRef>
                   <SituationSimpleRef>36926</SituationSimpleRef>
                </SituationRef>
                <Monitored>true</Monitored>
                <VehicleLocation>
                   <Longitude>-74.076856</Longitude>
                   <Latitude>40.607764</Latitude>
                </VehicleLocation>
                <Bearing>227.4601</Bearing>
                <ProgressRate>normalProgress</ProgressRate>
                <VehicleRef>6226</VehicleRef>
                <MonitoredCall>
                   <StopPointRef>201092</StopPointRef>
                   <VisitNumber>1</VisitNumber>
                   <StopPointName>HYLAN BL - DONLEY AV</StopPointName>
                   <EstimatedDepartureTime>2012-06-15T13:41:00-
04:00</EstimatedDepartureTime>
                </MonitoredCall>
                <OnwardCalls>
                   <OnwardCall>
                      <StopPointRef>905018</StopPointRef>
                      <VisitNumber>1</VisitNumber>
                      <StopPointName>HYLAN BL - NARROWS RD S</StopPointName>
                      <EstimatedDepartureTime>2012-06-15T13:44:00-
04:00</EstimatedDepartureTime>
                   </OnwardCall>
                   <OnwardCall>
                      <StopPointRef>201093</StopPointRef>
                      <VisitNumber>1</VisitNumber>
                      <StopPointName>OLGA PL - POUCH TER</StopPointName>
                      <EstimatedDepartureTime>2012-06-15T13:47:00-
04:00</EstimatedDepartureTime>
                   </OnwardCall>
                </OnwardCalls>
             </MonitoredVehicleJourney>
          </VehicleActivity>
          <VehicleActivity>
             <RecordedAtTime>2012-06-15T13:40:04.749-04:00</RecordedAtTime>
             <MonitoredVehicleJourney>
                <LineRef>S78</LineRef>
                <DirectionRef>1</DirectionRef>
                <FramedVehicleJourneyRef>
                   <DataFrameRef>2012-06-15</DataFrameRef>

   <DatedVehicleJourneyRef>20120408EA_072000_S78_0470_MISC_835</DatedVehicleJour
neyRef>
                </FramedVehicleJourneyRef>
                <JourneyPatternRef>S780470</JourneyPatternRef>
                <PublishedLineName>S78</PublishedLineName>
                <OperatorRef>MTA NYCT</OperatorRef>
```

```
                <OriginRef>805164</OriginRef>
                <DestinationRef>905179</DestinationRef>
                <DestinationName>BRICKTOWN MALL</DestinationName>
                <SituationRef>
                   <SituationSimpleRef>36926</SituationSimpleRef>
                </SituationRef>
                <Monitored>true</Monitored>
                <VehicleLocation>
                   <Longitude>-74.189972</Longitude>
                   <Latitude>40.522584</Latitude>
                </VehicleLocation>
                <Bearing>199.91507</Bearing>
                <ProgressRate>normalProgress</ProgressRate>
                <VehicleRef>6214</VehicleRef>
                <MonitoredCall>
                   <StopPointRef>201183</StopPointRef>
                   <VisitNumber>1</VisitNumber>
                   <StopPointName>HYLAN BL - CORNELIA AV</StopPointName>
                   <EstimatedDepartureTime>2012-06-15T13:40:30-
04:00</EstimatedDepartureTime>
                </MonitoredCall>
                <OnwardCalls>
                   <OnwardCall>
                      <StopPointRef>201184</StopPointRef>
                      <VisitNumber>1</VisitNumber>
                      <StopPointName>HYLAN BL - SEGUINE AV</StopPointName>
                      <EstimatedDepartureTime>2012-06-15T13:43:30-
04:00</EstimatedDepartureTime>
                   </OnwardCall>
                   <OnwardCall>
                      <StopPointRef>201185</StopPointRef>
                      <VisitNumber>1</VisitNumber>
                      <StopPointName>HYLAN BL - INEZ ST</StopPointName>
                      <EstimatedDepartureTime>2012-06-15T13:47:30-
04:00</EstimatedDepartureTime>
                   </OnwardCall>
                </OnwardCalls>
             </MonitoredVehicleJourney>
          </VehicleActivity>
       </VehicleMonitoringDelivery>
       <SituationExchangeDelivery>
          <Situations>
             <PtSituationElement>
                <SituationNumber>36926</SituationNumber>
                <PublicationWindow>
                   <StartTime>2012-03-26T00:00:00-04:00</StartTime>
                   <EndTime>2012-06-29T23:59:00-04:00</EndTime>
                </PublicationWindow>
                <Severity>undefined</Severity>
                <Summary xml:lang="EN">S74 and S78 buses detoured due to the
Water Main Reconstruction</Summary>
                <Description xml:lang="EN">S74 and S78 buses detoured at all
times from Arthur Kill Rd between Veterans Rd and Boscombe Av</Description>
                <Affects>
                   <VehicleJourneys>
                      <AffectedVehicleJourney>
                         <LineRef>S74</LineRef>
```

```
                        <DirectionRef>1</DirectionRef>
                     </AffectedVehicleJourney>
                     <AffectedVehicleJourney>
                        <LineRef>S74</LineRef>
                        <DirectionRef>0</DirectionRef>
                     </AffectedVehicleJourney>
                     <AffectedVehicleJourney>
                        <LineRef>S78</LineRef>
                        <DirectionRef>0</DirectionRef>
                     </AffectedVehicleJourney>
                     <AffectedVehicleJourney>
                        <LineRef>S78</LineRef>
                        <DirectionRef>1</DirectionRef>
                     </AffectedVehicleJourney>
                  </VehicleJourneys>
               </Affects>
               <Consequences>
                  <Consequence>
                     <Condition>diverted</Condition>
                  </Consequence>
               </Consequences>
            </PtSituationElement>
         </Situations>
      </SituationExchangeDelivery>
   </ServiceDelivery>
</Siri>
```

12.3.4.5 Complex Vehicle Monitoring response to return monitored journeys – JSON Example

The following code fragment shows the equivalent response translating the SIRI-VM XML document into JSON value pairs.

```
{
"Siri":{
    "ServiceDelivery":{
         "ResponseTimestamp":"2012-06-15T13:40:20.884-04:00",
         "VehicleMonitoringDelivery":[
               {
                    "VehicleActivity":[
                          {
                               "MonitoredVehicleJourney":{
                                     "LineRef":"S78",
                                     "DirectionRef":"1",
                                     "FramedVehicleJourneyRef":{
                                          "DataFrameRef":"2012-06-15",

"DatedVehicleJourneyRef":"20120408EA_079500_S78_0470_MISC_890"
                                     },
                                     "JourneyPatternRef":"S780470",
                                     "PublishedLineName":"S78",
                                     "OperatorRef":"MTA NYCT",
                                     "OriginRef":"805164",
                                     "DestinationRef":"905179",
                                     "DestinationName":"BRICKTOWN MALL",
                                     "SituationRef":[
                                          {
                                                "SituationSimpleRef":"36926"
                                          }
                                     ],
```

```
                                    "Monitored":true,
                                    "VehicleLocation":{
                                         "Longitude":-74.076856,
                                         "Latitude":40.607764
                                    },
                                    "Bearing":227.4601,
                                    "ProgressRate":"normalProgress",
                                    "VehicleRef":"6226",
                                    "MonitoredCall":{
                                         "StopPointRef":"201092",
                                         "VisitNumber":1,
                                         "StopPointName":"HYLAN BL - DONLEY AV",
                                         "EstimatedDepartureTime":"2012-06-15T13:41:00-
04:00"
                                    },
                                    "OnwardCalls":{
                                         "OnwardCall":[
                                               {
                                                    "StopPointRef":"905018",
                                                    "VisitNumber":1,
                                                    "StopPointName":"HYLAN BL - NARROWS
RD S",
                                                    "EstimatedDepartureTime":"2012-06-
15T13:44:00-04:00"
                                               },
                                               {
                                                    "StopPointRef":"201093",
                                                    "VisitNumber":1,
                                                    "StopPointName":"OLGA PL - POUCH
TER",
                                                    "EstimatedDepartureTime":"2012-06-
15T13:47:00-04:00"
                                               }
                                         ]
                                    }
                               },
                               "RecordedAtTime":"2012-06-15T13:40:05.900-04:00"
                          },
                          {
                               "MonitoredVehicleJourney":{
                                    "LineRef":"S78",
                                    "DirectionRef":"1",
                                    "FramedVehicleJourneyRef":{
                                         "DataFrameRef":"2012-06-15",

"DatedVehicleJourneyRef":"20120408EA_072000_S78_0470_MISC_835"
                                    },
                                    "JourneyPatternRef":"S780470",
                                    "PublishedLineName":"S78",
                                    "OperatorRef":"MTA NYCT",
                                    "OriginRef":"805164",
                                    "DestinationRef":"905179",
                                    "DestinationName":"BRICKTOWN MALL",
                                    "SituationRef":[
                                         {
                                               "SituationSimpleRef":"36926"
                                         }
                                    ],
                                    "Monitored":true,
                                    "VehicleLocation":{
                                         "Longitude":-74.189972,
                                         "Latitude":40.522584
```

```
                                    },
                                    "Bearing":199.91507,
                                    "ProgressRate":"normalProgress",
                                    "VehicleRef":"6214",
                                    "MonitoredCall":{
                                         "StopPointRef":"201183",
                                         "VisitNumber":1,
                                         "StopPointName":"HYLAN BL - CORNELIA AV",
                                         "EstimatedDepartureTime":"2012-06-15T13:40:30-
04:00"
                                },
                                    "OnwardCalls":{
                                         "OnwardCall":[
                                            {
                                                    "StopPointRef":"201184",
                                                    "VisitNumber":1,
                                                    "StopPointName":"HYLAN BL - SEGUINE
AV",
                                                    "EstimatedDepartureTime":"2012-06-
15T13:43:30-04:00"
                                                },
                                            {
                                                    "StopPointRef":"201185",
                                                    "VisitNumber":1,
                                                    "StopPointName":"HYLAN BL - INEZ ST"
                                                    "EstimatedDepartureTime":"2012-06-
15T13:46:30-04:00"
                                                }
                                         ]
                                    }
                               },
                               "RecordedAtTime":"2012-06-15T13:40:04.749-04:00"
                          },
                    ],
                    "ResponseTimestamp":"2012-06-15T13:40:20.884-04:00",
                    "ValidUntil":"2012-06-15T13:41:20.884-04:00"
               }
          ],
          "SituationExchangeDelivery":[
               {
                    "Situations":{
                         "PtSituationElement":[
                              {
                                    "PublicationWindow":{
                                         "StartTime":"2012-03-26T00:00:00.000-04:00",
                                         "EndTime":"2012-06-29T23:59:00.000-04:00"
                               },
                               "Summary":"S74 and S78 buses detoured due to the Water Main
Reconstruction",
                                    "Description":"S74 and S78 buses detoured at all
times from Arthur Kill Rd between Veterans Rd and Boscombe Av",
                                    "Affects":{
                                         "VehicleJourneys":{
                                               "AffectedVehicleJourney":[
                                                    {
                                                          "LineRef":"S74",
                                                          "DirectionRef":"1"
                                                    },
                                                    {
                                                          "LineRef":"S74",
                                                          "DirectionRef":"0"
                                                    },
```

```
                                                {
                                                      "LineRef":"S78",
                                                      "DirectionRef":"0"
                                                },
                                                {
                                                      "LineRef":"S78",
                                                      "DirectionRef":"1"
                                                }
                                          ]
                                    }
                              },
                              "Consequences":{
                                    "Consequence":[
                                          {
                                                "Condition":"diverted"
                                          }
                                    ]
                              },
                              "SituationNumber":"36926"
                        }
                  ]
            }
      }
   ]
}
}
```


12.3.5 SIRI-SM Stop Monitoring failed request with Exception – SIRI LITE Examples

12.3.5.1 General

The following example shows a JSON response reporting an error condition on a SIRI-SM StopMonitoringRequest to fetch departures request for a stop.

12.3.5.2 Simple Stop Monitoring response to return exceptions – XML Example

The following code fragment shows the XML document with an error condition for a SIRI XML response for a SIRI-SM service.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Siri xmlns:ns2="http://www.ifopt.org.uk/acsb"
xmlns:ns4="http://datex2.eu/schema/1_0/1_0"
xmlns:ns3="http://www.ifopt.org.uk/ifopt" xmlns="http://www.siri.org.uk/siri">
    <ServiceDelivery version=”2.0”>
        <ResponseTimestamp>2012-06-15T14:32:27.962-04:00</ResponseTimestamp>
        <StopMonitoringDelivery>
            <ResponseTimestamp>2012-06-15T14:32:27.962-04:00</ResponseTimestamp>
            <ErrorCondition>
                <OtherError>
                    <ErrorText>No such stop: XYZ</ErrorText>
                </OtherError>
                <Description> No such stop: XYZ </Description>
            </ErrorCondition>
        </StopMonitoringDelivery>
    </ServiceDelivery>
</Siri>
```

12.3.5.3 Simple Stop Monitoring response to return exceptions – JSON Example

The following code fragment shows the equivalent response translating the SIRI-SM XML document with an error condition into JSON value pairs.

```
{
    "Siri":{
        "ServiceDelivery":{
            "ResponseTimestamp":"2012-06-15T14:32:38.712-04:00",
            "StopMonitoringDelivery":[
                {
                    "ResponseTimestamp":"2012-06-15T14:32:38.712-04:00",
                    "ErrorCondition":{
                        "OtherError":{
                            "ErrorText":"No such stop: XYZ"
                        },
                        "Description":"No such stop: XYZ"
                    }
                }
            ]
        }
    }
}
```

12.4 Mapping of SIRI XML to Alternative encodings

12.4.1 Use of syntactic features of alternative rendering formats

A guiding principle is that a rendering should be chosen that can be produced unambiguously and simply from the XML, and that is as easy as possible to understand. For example:

1) When JSON is used, the root element of the JSON response should be a JSON object with the sole member “Siri” (just as the root element of any XML response is the <Siri> element);

2) When JSON is used, the nesting of repeated elements should be used, rather than arrays;

12.4.2 Mapping of SIRI data types to alternative encodings

Data types are normally rendered as in the normal SIRI XML. With one optional variant: a further transform parameter **#TimeFormat=Unix** option may be supported to make the handling of dates and durations on devices easier.

http://www.vbb.de/siri/2.0/stop-monitoring.json?MonitoringRef=4565
&StopMonitoringDetailLevel=calls&#TimeFormat=Unix

If #TimeFormat=Unix is chosen:

— Timestamps (xsd:dateTime) are given in Unix time, defined as the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of Thursday, January 1, 1970 (Unix times are defined, but negative, before that date), not counting leap seconds;

— Durations (xsd:duration) are given in seconds.

12.5 Recommendations for the use of SIRI Simple Web Services

12.5.1 General

SIRI is an extensive standard, many aspects of which are optional; some parts of it are more applicable to end user (i.e. web and mobile) passenger information applications than others. This clause provides basic guidance on implementing SIRI Lite so as to maximize the benefit passenger information applications.

12.5.2 Services useful for device Passenger Information Services

Of the many SIRI functional services, the following are most likely to be used by end user applications: ***StopMonitoring***, ***VehicleMonitoring***, ***SituationExchange.*** Any SIRI implementation of the simple web services should strive to implement these three services, whenever possible (assuming the data underlying the services is available).

The following two SIRI discovery services are also useful to allow applications to obtain reference data that is synchronised with that used by the SIRI Functional services: ***StopPointsRequest*** and ***LinesRequest.***

12.5.3 Response filtering

For maximum efficiency, support the parameters that allow the Consumer to limit the amount, type and detail of data returned:

— Allow the client to filter, at minimum, by ***OperatorRef*** (in a multi-operator system), ***LineRef***, ***DirectionRef***, ***MonitoringRef*** (for StopMonitoring), and ***VehicleRef*** (for VehicleMonitoring);

— Allow the client to set the detail of responses with ****DetailLevel***={normal, | calls};

— Allow the client to set quantity moderators, e.g. ***MaxNumberOfCalls.Onwards***, and **MaximumStopVisits**;

12.5.4 Incorporation of reference data in responses

SIRI LITE provides data that can be delivered straight to end-users: responses shall include stop names and other explanatory data from the reference data sets so that the data is self-explanatory and does not need further integration with other reference datasets in order to be understandable by users.

— Include human-readable names for stops (***StopPointName*** in ***OnwardCall***), lines (***PublishedLineName***), destinations (***DestinationName***), etc.

— When human-readable names are not possible or desired (e.g. ****Ref***), use identifiers that are consistent with other static descriptions (e.g. GTFS, TransXChange, NeTEx) that are available to application developers.

12.5.5 Multiple functional service deliveries in the same response

When applicable, the ***ServiceDelivery*** returned in responses for one type of request should have supporting responses of other types as necessary, to minimize the number of calls required from an end-user application to obtain a complete set of data suitable for end-user presentation. For example, a ***StopMonitoringDelivery*** returned in response to a ***StopMonitoringRequest*** might be accompanied by a ***SituationExchangeDelivery*** containing incident or service diversion information for the stops and/or lines covered by that ***StopMonitoringDelivery***.

SIRI v2.0 includes an explicit parameter ***IncludeSituations***, which can be used to request this.

12.5.6 Support a choice of response encodings

Wherever possible, provide a variety of response encodings (a suggested minimum would be XML and JSON) in order to lower the barrier-to-entry for developers. Doing so is less of a technical requirement, and more of a strategy to complement skills a developer may or may not have, or the frameworks supported by different development platforms.

12.5.7 Provide reporting identifiers

It is useful for a service provider to be able to track requests by application and by user in order to be able to analyse traffic patterns and better understand user needs.

The authentication details may be used to track requests by account using the ***AccountId***. This will typically be common to all the users of a particular application or applications provided by a particular application provider.

To support the tracking of requests by individual users, the ***RequestorRef*** may be used to hold an anonymised token identifying the user. This is populated by the individual device. The application should ask the user for consent before enabling this feature.

13 Common SIRI elements & Data Types

13.1 General

SIRI makes use of a number of reference data elements (for example LOGICAL DISPLAYs, LINEs etc.) that typically will have been exchanged previously between systems using asynchronous services. The Transmodel based NeTEx data model shows the relationship of these elements to the underlying public transport information system elements. The NeTEx data format may be used to exchange such elements.

Table 56 SIRI – NETEX equivalents

SIRI Reference / Element	Transmodel / NeTEx entity	Note
BlockRef	BLOCK	
Call	CALL	
ConnectionLinkRef	CONNECTION	
CourseOfJourneyRef	COURSE OF JOURNEYS	
DatedVehicleJourneyRef / DatedVehicleJourney, TargetedVehicleJourney	DATED VEHICLE JOURNEY, NORMAL DATED VEHICLE JOURNEY	
DestinationDisplay	DESTINATION DISPLAY	Well defined name of a destination used on vehicle and sign displays.
DirectionRef / Direction	DIRECTION	
FacilityRef	EQUIPMENT, FACILITY	
InterchangeRef / ServiceJourneyInterchange	SERVICE JOURNEY INTERCHANGE	
JourneyPartRef	JOURNEY PART	
JourneyPatternRef	JOURNEY PATTERN	
LineRef / Line	LINE	
MonitoringRef /	LOGICAL DISPLAY |	NeTEx LOGICAL DISPLAY relates to a

MonitoringPoint	SCHEDULED STOP POINT	SCHEDULED STOP POINT and relevant JOURNEY PATTERNs
MonitoredVehicleJourney	MONITORED VEHICLE JOURNEY	A VEHICLE JOURNEY that is being monitored
OperatorRef / Operator	OPERATOR, AUTHORITY	
SituationRef	SITUATION	
StopPointRef / StopPoint	SCHEDULED STOP POINT	See also SIRI MonitoringPoint
TimetableVersionRef / Timetable	TIMETABLE FRAME	NeTEx TIMETABLE FRAME groups a version of a timetable.
TrainNumberRef	TRAIN NUMBER	UIC train number
VehicleMode	VEHICLE MODE	Enumerated values only
VehicleRef	VEHICLE	

13.2 Introduction

Some elements and groups of elements are shared by many different SIRI Functional Service Deliveries. This clause describes some common definitions that are referenced by services in SIRI Functional Service Specifications (Part3, Part4, Part5).

13.3 Base Data Types

13.3.1 W3C Simple Types

SIRI uses a number of standard W3C XML data types.

Table 57 — W3C XML simple data types used in SIR

Data Type
xsd:anyType
xsd:anyURI
xsd:base64Binary
xsd:boolean
xsd:date
xsd:dateTime
xsd:decimal
xsd:duration
xsd:float
xsd:integer
xsd:language
xsd:normalizedString
xsd:nonNegativeInteger
xsd:NMTOKEN
xsd:NMTOKENS
xsd:positiveInteger

xsd:string
xsd:time

13.3.2 SIRI Simple Types

SIRI defines a number of common simple data types.

Table 58 — SIRI simple data types used in SIRI

Data Type	Description	Example
AbsoluteBearingType	Bearing in compass degrees (0-360). North equates to 0 degrees, while east is 90 degrees.	<***Bearing***>180</***Bearing***>
DurationType	Limited version of xsd:duration that allows for precise time arithmetic. Only Month, Day, Hour, Minute Second or Millisecond terms shall be used.	<***Delay***>-PT1H2M</***Delay***>-
EmailAddressType	Email address type.	<***Email***>-info@siri.org.uk</***Email***>-
EmptyType	Type that has no contained value.	<***All***/>
LongitudeTyype	Longitude from Greenwich Meridian 180° (East) to +180° (West). Decimal degrees.	<***Longitude***>2.356</***Longitude***>
LatitudeType	Latitude from equator. -90° (South) to +90° (North). Decimal degrees.	<***Latitude***>56.356.</***Latitude*** >
PopulatedStringType	A restriction of W3C XML Schema's string that requires at least one character of text.	<***Name***>Paris</***Name***>
PositiveDurationType	Limited version of duration. Has to be positive.	<***TransferTime***>PT1H2M</***TransferTime***>
PhoneType	International phone number.	<***Phone***>+41675601</***Phone***>
VersionString	A string indicating the version of a SIRI data structure.	<***Version***>2.1</***Version***>

13.3.3 NationalLanguageStringStructure — Element

Most text elements in SIRI may be specified in a designated language. If no language is specified the default language from the context should be assumed

NationalLanguageStringStructure		0:1	*+Structure*	A populated string in a designated national language.
	lang	0:1	*xml:lang*	Language for string ISO language code

13.4 Shared Elements & Structures

13.4.1 FramedVehicleJourneyRef — Element

The ***FramedVehicleJourneyRef*** identifies a DATED VEHICLE JOURNEY within the data horizon of the referencing system. It may be that two data systems that share information about connecting journeys use the same ***VehicleJourneyCode*** for different journeys in their respective systems. To ensure that the journeys can be uniquely identifier a data frame reference can be used as a qualifier. In practice the ***OperationalDayType*** may be used as a unique qualifier of the data frame.

Table 59 — FramedVehicleJourneyRef

FramedVehicleJourneyRef		0:1	*+Structure*	A reference to the DATED VEHICLE JOURNEY that the vehicle is making. Unique with the data horizon of the service.
	DataFrameRef	0:1	*DataFrameQualifier*	Unique identifier of data frame within participant service. Used to ensure that the ***DatedVehicleJourneyRef*** is unique with the data horizon of the producer. Often the ***OperationalDayType*** is used for this purpose.
	DatedVehicleJourneyRef	0:1	*→DatedVehicleJourneyCode*	A reference to the DATED VEHICLE JOURNEY that the VEHICLE is making.

13.4.2 Location — Element

The ***Location*** structure provides a standard way of defining a geospatial coordinate, for example a vehicle position. When ***Location*** is used for a vehicle, for example for a ***VehicleLocation***, the front of the vehicle shall be taken as the measurement point.

Table 60 — Location

LocationStructure			0:1	*+Structure*	Geospatial Location
Attributes	***id***		0:1	*xsd:NMTOKEN*	Arbitrary identifier associated with the point.
	srsName		0:1	*xsd:string*	Identifier of the GML coordinate system used in any ***Coordinates*** parameter. Defaults to overall value for document.
Coordinates				*choice*	Location in one of two alternate formats.
	a	***Longitude***	**–1:1**	*LongitudeType*	Longitude from Greenwich Meridian.180° (East) to +180° (West). Decimal degrees. e.g. 2,356.
		Latitude	**–1:1**	*LatitudeType*	Latitude from equator. -90° (South) to +90° (North). Decimal degrees. e.g. 56,356.
	b	***Coordinates***	**–1:1**	*xsd:string*	Coordinates of points in a GML compatible format, as indicated by ***srsName*** attribute.
	Precision		0:1	*DistanceType*	Precision for point measurement. In meters.

13.4.3 Error — Element

13.4.3.1 General

The ***xxxError*** structure provides a standard way of representing a specific error code that can be used for fault or exception handling. Error codes are explicitly reified as elements, for example ***CapabilityNotSupportedError***, ***AccessNotAllowedError***, ***NoInfoForTopicError***, etc.

Table 61 — Error Code

ErrorStructure		0:1	*+Structure*	Error Type.
	ErrorNumber	0:1	*xsd:integer*	Number for error type (+SIRI v2.0)
	ErrorText	0:1	*xsd:string*	Additional description of error.

13.4.3.2 Error Conditions — Elements

Table 62 — Error Conditions

The following specific error conditions can arise – these are all specialisations of ***Error***.

Error	Explanation	From
AccessNotAllowedError	Requestor is not authorised to the service or data requested.	
AllowedResourceUsageExceededError	Valid request was made but request would exceed the permitted resource usage of the client.	
BeyondDataHorizon	Data period or subscription period is outside of period covered by service.	
CapabilityNotSupportedError	Service does not support the requested capability.	
EndpointDeniedAccessError	Endpoint to which a message is to be distributed did not allow access by the client.	+v2.0
EndpointNotAvailableAccessError	Recipient of a message to be distributed is not available.	+v2.0
InvalidDataReferencesError	Request contains references to identifiers that are not known.	+v2.0
NoInfoForTopicError" type	Valid request was made but service does not hold any data for the requested topic expression.	
OtherError	Error type other than the well-defined codes.	
ParametersIgnoredError	Request contained parameters that were not supported by the producer. A response has been provided but some parameters have been ignored.	+v2.0
ServiceNotAvailableError	Functional service is not available to use (but it is still capable of giving this response).	
UnknownEndpointError	Recipient for a message to be distributed is unknown.	+v2.0
UnknownExtensionsError	Request contained extensions that were not supported by the producer. A response has been provided but some or all extensions have been ignored.	
UnknownParticipantError	Recipient for a message to be distributed is unknown.	+v2.0
UnknownSubscriberError	Subscriber not found.	
UnknownSubscriptionError	Subscription not found.	
UnapprovedKeyAccessError	Recipient of a message to be distributed is not available.	+v2.0

13.5 Shared groups of elements

13.5.1 ServiceInfoGroup — Group

The ***ServiceInfoGroup*** provides optional data about descriptive attributes of a VEHICLE JOURNEY.

Table 63 — ServiceInfoGroup — Elements

Service Info	***OperatorRef***	0:1	*→OperatorCode*	OPERATOR of journey.
	ProductCategoryRef	0:1	*→ProductCategory Code*	PRODUCT CATEGORY of journey – classifies, for example; *express, local.*
	ServiceFeatureRef	0:*	*→ServiceFeatureCode*	Classification of service into arbitrary Service Features, e.g. *school bus.*
	VehicleFeatureRef	0:*	*→VehicleFeatureCode*	Feature of VEHICLE. E.g. '*suitableForWheelChairs'*.

13.5.2 JourneyInfoGroup — Group

The ***JourneyInfoGroup*** provides optional data about a JOURNEY of any sort.

Table 64 — JourneyInfoGroup — Elements

Journey Info	***VehicleJourneyName***	0:*	*NLString*	Name of VEHICLE JOURNEY. (One per language (Unbounded 0:* since +SIRI 2.0).
	JourneyNote	0:*	*NLString*	Additional descriptive text associated with journey. One per language (Unbounded 0:* since +SIRI 2.0).
	PublicContact	0:1	*+Structure*	Contact details for use by members of public. +SIRI v2.0
	PhoneNumber	0:1	*PhoneType*	Phone number for Public to contact OPERATOR of journey. +SIRI v2.0
	Url	0:1	*xsd:anyUri*	Public URL to contact OPERATOR of journey. +SIRI v2.0
	OperationsContact	0:1	*+Structure*	Contact details for use by operational staff. +SIRI v2.0
	PhoneNumber	0:1	*PhoneType*	Phone number for operational contact. Not for Public use. +SIRI v2.0
	Url	0:1	*xsd:anyUri*	URL number for operational contact. Not for Public use. +SIRI v2.0

13.5.3 VehicleJourneyInfoGroup — Group

The ***VehicleJourneyInfoGroup*** provides optional data about a VEHICLE JOURNEY. It includes a ***ServiceInfoGroup*** and a ***JourneyInfoGroup.***

Table 65 — VehicleJourneyInfoGroup — Elements

Service Info	*:::*	0:1	*ServiceInfoGroup*	See 12.3 *ServiceInfoGroup*.
Service End Point Names	***OriginRef***	0:1	*→JourneyPlaceCode*	The identifier of the origin of the journey; used to help identify the VEHICLE JOURNEY on arrival boards.
	OriginName	0:*	*NLString*	The name of the origin of the journey; used to help identify the VEHICLE to the public. One per language (Unbounded 0:* since +SIRI 2.0).
	OriginShortName	0:*	*NLString*	The short name of the origin of the journey; used to help identify the VEHICLE to the public. (One per language (Unbounded 0:* since +SIRI 2.0).
	DestinationDisplayAtOrigin	0:*	*NLString*	DESTINATION DISPLAY name shown for journey at the origin. +SIRI v2.0. (One per language
	Via	0:*	*+Structure*	Description of a VIA point on a journey.
	PlaceRef	0:1	*→JourneyPlaceCode*	Identifier of a VIA point of the journey.
	PlaceName	0:*	*NLString*	The name of a VIA point of the journey, used to help identify the LINE. (One per language (Unbounded 0:* since +SIRI 2.0).
	PlaceShortName	0:*	*	Short name of a VIA point of the journey, used to help identify the LINE. (One per language (Unbounded 0:* since +SIRI 2.0).
	ViaPriority	0:1	*xsd:integer*	Relative priority to give to VIA name in displays. 1=high. Default is 2. +SIRI v2.0.
	DestinationRef	0:1	*→JourneyPlaceCode*	The identifier of the destination of the journey; used to help identify the VEHICLE to the public.
	DestinationName	0:*	*NLString*	The name of the destination of the journey; used to help identify the VEHICLE to the public. (One per language (Unbounded 0:* since +SIRI 2.0).
	DestinationShortName	0:*	*NLString*	The name of the destination of the journey; used to help identify the VEHICLE to the public. (One per language (Unbounded 0:* since +SIRI 2.0).
	OriginDisplayAtOrigin	0:*	*NLString*	Origin DISPLAY name shown for journey at the destination . +SIRI v2.0. (One per language
Journey Info	*JourneyInfo Group*	***:::***	0:1	*JourneyInfoGroup* See above.
End Times	***HeadwayService***	0:1	*xsd:boolean*	Whether this is a HEADWAY INTERVAL Service, that is one shown as operating at a prescribed interval rather than to a fixed timetable.
	OriginAimedDepartureTime	0:1	*xsd:dateTime*	Timetabled departure time of VEHICLE from Origin.
	DestinationAimedArrivalTime	0:1	*xsd:dateTime*	Timetabled arrival time of VEHICLE at Destination.
	FirstOrLastJourney	0:1	*FirstOrLastJourneyEnumeration*	Whether journey is first or last journey of day. +SIRI v2.0.

13.5.3.1 *FirstOrLastJourney* — Allowed values

The table below provides allowed values for ***FirstOrLastJourney*** (*FirstOrLastJourneyEnumeration*).

Table 66 — *FirstOrLastJourney* —Allowed Values (SIRI 2.0)

Value	Description
unspecified	Unspecified whether first or last
firstServiceOfDay	Service is first of day.
lastServiceOfDay	Service is last of day.
other	Service is neither first nor last of day.

13.5.4 JourneyPatternInfoGroup — Group

The ***JourneyPatternInfoGroup*** provides optional data about the ROUTE and LINE of a VEHICLE JOURNEY that originates with the ***JourneyPattern*** associated with the ***VehicleJourney***.

Table 67 — JourneyPatternInfoGroup — Elements

Journey Pattern Info	***JourneyPatternRef***	0:1	→*JourneyPatternCode*	Identifier of JOURNEY PATTERN that journey follows.
	JourneyPatternName	0:1	*NLString*	Name or Number by which the JOURNEY PATTERN is known to the public. (+SIRI 2.0).
	VehicleMode	0:1	*VehicleMode*	A method of transportation such as bus, rail, etc.
	RouteRef	0:1	→*RouteCode*	Identifier of ROUTE or SERVICE PATTERN that journey follows.
	PublishedLineName	0:*	*NLString*	Name or Number by which the LINE is known to the public. One per language (Unbounded 0:* since +SIRI 2.0).
	GroupOfLinesRef	0:1	→*GroupOfLinesCode*	Identifier of GROUP OF LINEs to which service belongs (+SIRI V2.0)
	DirectionName	0:*	*NLString*	Name of the relative direction the VEHICLE is running along the LINE, for example, "inbound" or "outbound". One per language (Unbounded 0:* since +SIRI 2.0).
	ExternalLineRef	0:1	→*LineCode*	Alternative identifier of LINE that an external system may associate with journey.

13.5.4.1 *VehicleMode* — Allowed values

The table below provides the allowed values for ***VehicleMode*** (*VehicleModeEnumeration*).

Table 68 — *VehicleMode* — Allowed Values (SIRI 2.0)

Value	Description	NeTEx Mode
air	*Air transport*	*Air*
bus	*Bus transport*	*Bus*
coach	*Coach transport*	*Coach*
ferry	*Ferry transport*	*Ferry*
metro	*Metro transport*	*Metro*
rail	*Rail transport*	*Rail*
tram	*Tram transport*	*Tram*
underground	*Metro – underground*	*Metro: Submode undergound*

13.5.5 DisruptionGroup — Group

13.5.5.1 General

The ***DisruptionGroup*** provides optional data about the real-time disruptions of a VEHICLE JOURNEY or stop, or the EQUIPMENT or facilities associated with it. The group can include a reference to a SITUATION that can be used to associate the disrupted element with a structured incident description, fetched separately using the SIRI-SX service (See SIRI Part 5).

Table 69 — DisruptionGroup — Elements

Disruption	***FacilityConditionElement***	0:1	+*Structure*	Information about a change of Equipment availability at stop that may affect access or use of the Facility. See SIRI-FM Part 4 for further details.
FacilityChange	***FacilityChangeElement***	0:1	+*Structure*	Information about a change of EQUIPMENT availability at stop that may affect access or use. See below
Situation	***SituationRef***	0:*	→*SituationCode*	Reference to a SITUATION associated with the element. See SIRI-SX Part 5 for further details.

13.5.5.2 *FacilityChangeElement* — Element

The ***FacilityChangeElement*** provides optional data about the real-time changes to a piece of EQUIPMENT at a stop or on a VEHICLE. The causes and consequence of the equipment change can be further explained by structured SITUATION description (See SIRI-SX Part 5), referenced by a situation which can be used to tie the element together with a structured incident description that further, and which can be fetched separately with the SIRI-SX service.

Table 70 — FacilityChangeElement — Elements

FacilityChangeElement				*+Structure*	Information about a change of EQUIPMENT availability at stop that may affect access or use.
Equipment	***EquipmentAvailability***		0:1	*+Structure*	Availability change for EQUIPMENT item.
		EquipmentRef	0:1	*→EquipmentCode*	Identifier of the EQUIPMENT.
		Description	0:*	*NLString*	Description of EQUIPMENT. (One per language (Unbounded 0:* since +SIRI 2.0).
		EquipmentStatus	**1:1**	*EquipmentStatusEnum*	Status of the EQUIPMENT availability. Enumeration. Default is '*notAvailable'*. *unknown* | *available* | *notAvailable*
		Description	0:*	*NLString*	Description of EQUIPMENT. (One per language (Unbounded 0:* since +SIRI 2.0).
		ValidityPeriod	0:1	*+Structure*	Period for which Status Change applies. If omitted, indefinite period.
			StartTime	**1:1**	*xsd:dateTime*	The (inclusive) start time stamp.
			EndTime	0:1	*xsd:dateTime*	The (inclusive) end time stamp. If omitted, the range end is open-ended, that is, it should be interpreted as "forever".
		EquipmentTypeRef	0:1	*→EquipmentTypeCode*	Reference to EQUIPMENT type identifier.
		Features	0:1	*+Structure*	Service Features associated with EQUIPMENT.
			Feature	**1:***	*ServiceFeature*	Service or Stop features associated with equipment. Recommended values based on TPEG are given in SIRI documentation and enumerated in the siri_facilities package.
Situation	***SituationRef***		0:*	*→SituationCode*	Reference to a Situation associated with the ***FacilityChangeElement*** that explains the equipment change. See SIRI-SX Part 5 for further details.
Mobility Effect	***MobilityDisruption***		0:1	*+Structure*	Effect of change on impaired access users.
		MobilityImpairedAccess	0:1	*xsd:boolean*	Whether stop or service is accessible to mobility impaired users. This may be further qualified by one ore more MobilityFacility instances to specify which types of mobility access are available (true) or not available (*false*). For example *'suitableForWheelChair'*, or 'stepFreeAccess'.
		MobilityFacility	0:1	*MobilityFacilityEnum*	Classification of Mobility Facility type - Based on TPEG pti23. *suitableForWheelChairs* | *lowFloor* | *stepFreeAccess* | *boardingAssistance* | *onboardAssistance* | *unaccompaniedMinorAssistance* | *audioInformation* | *visualInformation* | *displaysForVisuallyImpaired* | *audioForHearingImpaired* | *tactileEdgePlatforms*

13.5.5.3 *EquipmentStatus* — Allowed values

The table below provides allowed values for ***EquipmentStatus*** (*ProgressRateEnumeration*)

Table 71 — *EquipmentStatus* —Allowed Values

Value	Description
unknown	Status is unknown
available	Status is available
notAvailable	Status is not available

13.5.5.4 *MobilityFacility* — Allowed values

The table below provides allowed values for ***MobilityFacility*** (*MobilityFacility Enumeration*)

Table 72 — *MobilityFacility* —Allowed Values

Value	Description
suitableForWheelChairs	*Suitable for Wheelchairs*
lowFloors	*Low Floors*
stepFreeAccess	*Step Free Access*
boardingAssistance	*Boarding assistance available.*
onboardAssistance	*On-board assistance available.*
unaccompaniedMinorAssistance	*Unaccompanied Minor Assistance*
audioInformation	*Audio Information available.*
visualInformation	*visual Information available.*
displaysForVisuallyImpaired	*Special displays for the Visually Impaired*
audioForHearingImpairedt	*Audio facilities for the Hearing Impaired*
actileEdgePlatform	*Tactile Edge on Platforms*
audioForHearingImpaired	*Audio for Hearing impaired*
tactileEdgePlatform	*Tactile Edge Platform*

13.5.6 JourneyProgressGroup — Group

The ***JourneyProgressGroup*** provides optional data about the real-time status of a vehicle journey of a ***MonitoredVehicleJourney***.

Table 73 — JourneyProgressGroup — Elements

Status	***Monitored***	0:1	*xsd:boolean*	Whether there is real-time information available for journey, if not present, not known.
	MonitoringError	0:1	*MonitoringErrorEnum*	If ***Monitored*** is *false*, an optional reason for non-availability of real-time data. *GPS* | *GPRS* | *Radio*
Progress Data Quality	***InCongestion***	0:1	*xsd:boolean*	Whether the vehicle is in congestion. If not, present, not known.
	InPanic	0:1	*xsd:boolean*	Whether the vehicle alarm is on. Likely to indicate unpredictable progress. If not, present, false.
	PredictionInaccurate	0:1	*xsd:boolean*	Whether the prediction should be judged as inaccurate.
	DataSource	0:1	*xsd:string*	System originating real-time data, if other than producer. Can be used to make judgements of relative quality and accuracy of a proxied source compared to other feeds.
	ConfidenceLevel	0:1	*QualityIndexEnum*	A confidence level associated with data. *certain* | *veryReliable* | *reliable* | *probablyReliable* | *unconfirmed*
Progress Data	***VehicleLocation***	0:1	*LocationStructure*	Current location of VEHICLE. Measured to front of bus.
	Bearing	0:1	*AbsoluteBearingType*	Bearing in degrees in which VEHICLE is heading.
	ProgressRate	0:1	*ProgressRateEnum*	Classification of the rate of progress of VEHICLE. *noProgress* | *slowProgress* | *normalProgress* | *fastProgress* | *unknown*
	Velocity	0:1	*VelocityType*	Velocity of vehicle in specified units (See ServiceRequestContext). Either actual speed or average speed may be used. Default units are metres per second. +SIRI v2.0.
	Occupancy	0:1	*OccupancyEnum*	How full VEHICLE is. Enumeration. If omitted, not known.
	Delay	0:1	*DurationType*	Delay to a precision in seconds. Early times are shown as negative values.
	ProgressStatus	0:1	*NatuaralLanguageStringStructure*	A non-displayable status describing the running of this VEHICLE.
	VehicleStatus	0:1	*VehicleStatusEnum*	A classification of the progress state of the VEHICLE JOURNEY. +SIRI 2.0 expected | notExpected | cancelled | assigned | signedOn | atOrigin | inProgress | aborted | offRoute | completed | assumedCompleted | notRun

13.5.6.1 *MonitoringError* — Allowed values

The table below provides allowed values for ***MonitoringError*** (*MonitoringErrorEnumeration*).

Table 74 — *MonitoringError* —Allowed Values

Value	Description
GPS	Failure is in GPS locating system
GPRS	Failure is in GPRS connection
Radio	Failure is in Radio connection

13.5.6.2 *ConfidenceLevel* — Allowed values

The table below provides allowed values for ***ConfidenceLevel*** (*QualityIndexEnumeration*).

Table 75 — *ConfidenceLevel* —Allowed Values

Value	Description
certain	Data is certain.
veryReliable	Data is very reliable.
reliable	Data is reliable.
probablyReliable	Data is probably reliable.
unconfirmed	Data is unconfirmed.

13.5.6.3 *ProgressRate* — Allowed values

The table below provides allowed values for ***ProgressRate*** (*ProgressRateEnumeration*)

Table 76 — *ProgressRate* —Allowed Values

Value	Description
noProgress	Vehicle is not moving.
slowProgress	Vehicle is making slower than normal progress.
normalProgress	Vehicle is making normal progress.
fastProgress	Vehicle is making better than normal progress.
unknown	Rate of progress is unknown.

13.5.6.4 *Occupancy* — Allowed values

The table below provides allowed values for ***Occupancy*** (*OccupancyEnumeration*)

Table 77 — *Occupancy* —Allowed Values

Value	Description
full	Service is full.
standingAvailable	Standing space is available.
seatsAvailable	Seats are available.

13.5.6.5 *CallStatus* — Allowed values

The table below provides allowed values for ***CallStatus*** (*CallStatusEnumeration*)

Table 78 — *CallStatus* —Allowed Values

Value	Description
onTime	Service is on time.
early	Service is earlier than expected.
delayed	Service is delayed.
cancelled	Service is cancelled.
arrived	Service has arrived.
Departed	Service has departed.
Missed	Arrival or departure was not recorded at this stop but the service has subsequently been detected further down the route.
noReport	There is no data available on the service status.

13.5.6.6 *VehicleStatus* — Allowed values

The table below provides allowed values for *Vehicle**Status*** (*VehicleStatusEnumeration*)

Table 79 — *VehicleStatus* —Allowed Values

Value	Description	Allowed transitions to
notExpected	A vehicle journey that is scheduled to be run only if ordered and has not yet been ordered.	*expected, assigned, not run*
expected	The VEHICLE JOURNEY is scheduled to run.	*signedOn, assigned, cancelled, assumedCompleted*
assigned	There is a vehicle assigned to run the dated vehicle journey. Assignments can be made in advance by the operations control to allocate a specific vehicle to work certain dated vehicle journeys.	*assigned, cancelled*
cancelled	A VEHICLE JOURNEY that was scheduled to run has been cancelled.	*expected*

signedOn	The assignment of a vehicle journey has been confirmed by the driver.	*atOrigin, inProgress*
atOrigin	Service has arrived at the first stop.	*inProgress, aborted*
inProgress	Service has departed from the first or later stop.	*completed, offRoute, aborted*
offRoute	System has detected that vehicle is not following the expected.	*inProgress, aborted*
aborted	A journey that has already been started has been aborted, e.g. because of a breakdown. If an aborted journey is resumed, a new vehicle journey instance will be created.	
completed	Journey has been completed	
assumedCompleted	If an expected vehicle journey is not cancelled and never becomes in progress, at some point in time be considered as assumed completed.	
notRun	If a not expected dated vehicle journey is never run, it should at some point in time be considered as not run. This could for instance be the time when the deadline for ordering	

13.6 OperationalBlockGroup — Group

The ***OperationalBlockGroup*** provides optional data to identify operational entities associated with a vehicle journey.

Table 80 — OperationalBlockGroup — Elements

Operational Block group	***BlockRef***	0:1	→*BlockCode*	BLOCK that VEHICLE is running.
	CourseOfJourneyRef	0:1	→*RunCode*	Run that VEHICLE is running.

13.7 OperationalInfoGroup — Group

The ***OperationalInfoGroup*** provides optional data to identify operational entities associated with a vehicle making a journey. It includes an ***OperationalBlockGroup.***

Table 81 — OperationalInfoGroup — Elements

Operational Info group	***:::***	0:1	*Operational BlockGroup*	See *Operational BlockGroup* Above.
	VehicleJourneyRef	0:1	→*VehicleJourneyCode*	Reference to VEHICLE JOURNEY
	VehicleRef	0:1	→*VehicleCode*	A reference to the specific VEHICLE making a journey.
	AdditionalVehicleJourneyRef	0:*	→*VehicleCode*	Refercence to a other VEHICLE Journeys (+SIRI v2.0)
	DriverRef	0:1	→*DriverCode*	A reference to the DRIVER or Crew currently logged in to operate a monitored VEHICLE. May be omitted if real-time data is not available - i.e. it is timetabled data. +SIRI v2.0
	DriverName	0:1	*Xsd:normalizedString*	The name of the Driver or Crew +SIRI v2.0

Bibliography

[1] ISO 24531:2013, *Intelligent transport systems — System architecture, taxonomy and terminology — Using XML in ITS standards, data registries and data dictionaries*

[2] EN 12896, *Road transport and traffic telematics - Public transport - Reference data model.*

[3] EN 28701, *Intelligent transport systems - Public transport - Identification of Fixed Objects in Public Transport (IFOPT)*

[4] CEN/TS 16614-1:2014, *NeTEx — Network and Timetable Exchange — Part 1: Public transport network topology exchange format*

[5] CEN/TS 16614-2:2014, *NeTEx — Network and Timetable Exchange — Part 2: Network Timing Information*

[6] *WSDL 11vs20,* available from: http://en.wikipedia.org/wiki/File:WSDL_11vs20.png