THALES

PROJECT WA063A0
RDG DARWIN
REAL TIME TRAIN INFORMATION

OUTPUT PORTS

Push Ports

ActiveMQ Communications Interface Specification

WA063A01624 Issue 1 18th January2017

Originator's signature & date	Approver's signature and date
--	--
D Barnard	T Parkinson

Thales, Ashurst Drive, Bird Hall Lane, Cheadle Heath, Stockport, Cheshire SK3 0XB, UK
Tel: +44 (0)161 491 4001•Fax: +44 (0)161 741 3704•ukinfo@thalesgroup.com•www.thalesgroup.com

DISTRIBUTION

1	Project Master File
2	RDG

ISSUE RECORD

Issue	Date	Purpose
1	18/01/2017	Split the Push Port Interface Specification to separate the data specification from the communications specification. Add clarifications from comments.

CONTENTS

OUTPUT PORTS
PUSH PORTS
ACTIVEMQ COMMUNICATIONS INTERFACE SPECIFICATION

1. Introduction

Push Ports are provided to meet the needs of external systems that deal with high volumes of enquires and that require rapid access to Darwin (RTTI) information. Darwin provides a “push” service of information in real-time that allows the client system to hold a copy of the Darwin database.

This document addresses a communications Interface Specification for the retrieval of information via the Darwin (RTTI) Push Ports Interface. The low-level communications interface to the Push Ports can vary, and the Apache ActiveMQ message queue-based interface is documented in this specification. Even though the Push Ports are accessed by different mechanisms, the data received by a client is the same and is documented in the Push Ports Data Interface Specification (Reference 2).

Darwin makes available via the Push Port creation of, and changes to, train schedule records, together with train running predictions made by Darwin. Note that predictions and changes to schedules are relative to the original schedule as created in Darwin, usually sourced from ITPS. In order to correctly interpret Darwin data, clients must also have access to the ITPS schedule data.

Darwin also supports the download of a complete XML-format timetable for those Clients who do not have access to their own ITPS-generated timetable.

Documentation for the ActiveMQ API is freely available online and can be accessed at the URL “http://activemq.apache.org/”. There are also multiple language-specific API wrappers that are available for most common programming languages.

2. References

1. Common Interface File, End User Specification, Issue 28, Nov. 2012. Issued by Network Rail.

2. P75301004 Push Port Data Interface Specification.

3. ActiveMQ Online Documentation (http://activemq.apache.org/)

3. Communication

Physical network connection between the Client network and Darwin is not specified here, but will typically be either a dedicated Leased WAN connection installed and maintained by an RDG supplier, a pair of VPN connections operating over the Internet, or direct Internet access. The difference between these options related to Client connectivity mainly involves resilience features in the event of a failure within the Darwin System.

When use is made of the dedicated WAN connection, failures within the Darwin system are transparent (at the communications level) to the Client system (other than a brief loss of connectivity). The WAN connection will be automatically re-configured to connect to an active Data Centre.

If VPN connections are used, there must be a separate VPN to each of the Darwin Data Centres and it is the responsibility of the Client to implement any logic and network routing to connect to any active Data Centre.

Direct Internet connectivity will be resilient at the Darwin end, but may not be at the client end, according to the local environment.

Security for the communication link is provided by use of the Leased WAN, or over the Internet by VPN encryption. Direct Internet communication will not be encrypted. The Darwin Firewall will only allow connection from known IP addresses to the ActiveMQ port. Push Port information is not deemed to be highly sensitive, so these measures are considered adequate.

In addition, connection to the ActiveMQ message broker utilises credentials to restrict the message queues that a client is allowed to access.

3.1 Connection

Push Port communication is carried out over one or more ActiveMQ message queues.

Client systems communicate with the Darwin System by connecting to a Darwin ActiveMQ messaging server. There may be multiple instances of the ActiveMQ server, for performance and resilience reasons. The Push Port Client application must specify the following in order to make a connection to Darwin:

- Name(s)/IP Address(es) and port number of the Darwin ActiveMQ server(s).
- Access credentials.
- ClientID allocated to the Push Port Client.

ActiveMQ supports several “wire protocols”, but currently only the native “OpenWire” protocol is supported by Darwin.

3.1.1 Connection Failover

Darwin will support multiple instances of the ActiveMQ service, any of which may be used by clients. If a service becomes unavailable then the client should connect to another available instance.

The recommended way of achieving this failover is to use the ActiveMQ “failover:” transport in the connection request. This transport has the following general syntax:

```
failover:(uri1,…,uriN)?transportOptions

where uri is like: “tcp://host:port”
```

By default, this transport will randomly select one of the configured URIs to connect to, and if that connection fails it will automatically attempt to reconnect to one of the alternatives, until a successful connection is made. Note that it is possible to use the failover protocol even if there is only one ActiveMQ server available (e.g. a staging system). The retry behaviour will still occur, although with just the one URI to try.

3.1.2 Connection Options

Many options can be set on the connection to control or optimise the behaviour of message delivery. Most can be left to their defaults in normal operation. Full details of the available options can be found in the ActiveMQ documentation (ref. 3).

Options can be set on the transport URI (as above in the “`?transportOptions`” value), or directly from code as properties on the `ActiveMQConnection` object.

Options that *must* be set are:

- watchTopicAdvisories=false

3.1.3 Internet Connection

The connection details when communication to Darwin is via a dedicated network connection or VPN will vary according to the setup of those communications links. The IP addresses to use for the connection to the Darwin ActiveMQ servers will be advised at the time of setup.

If communication is via the Internet then the Darwin ActiveMQ servers can be found at:

```
amq1.realtime.nationalrail.co.uk port 61616
amq2.realtime.nationalrail.co.uk port 61616
```

Note that clients that connect to a Darwin Staging system will have separate name(s)/IP addresses specific to that system, but otherwise connection details will be the same.

Using these details, an example minimum Internet ActiveMQ connection URI is:

```
failover:(tcp://amq1.realtime.nationalrail.co.uk:61616,tcp:/
/amq2.realtime.nationalrail.co.uk:61616)?watchTopicAdvisorie
s=false
```

3.1.4 Exclusive Consumers

A feature of ActiveMQ that may be of use to clients is the “exclusive consumer” option. By default, multiple client connections *may* be made to a *queue* and messages will be distributed round-robin between all connected clients. This could be used in a simple “load balancing” scheme, whereby each connected client will process a subset of the messages queued. However, this is not really recommended, as it makes it difficult to correlate the individual messages and their sequence numbers (sec. 5.2), though clients are free to make their own choices in this area.

Alternatively, each client can set the "consumer.exclusive=true" option when opening the queue. In this case, all but one of the client connections will block and only one will succeed. That client will receive all messages that are queued. If the connected client disconnects for any reason, one of the other blocked consumers will be selected as the exclusive consumer and all subsequent messages will be delivered to this client.

This allows the clients to implement a simple failover mechanism, whereby another client instance can automatically take over from a failed instance.

Use of this exclusive consumer option is not a requirement in order to use the ActiveMQ Push Port service, but may be used if useful to the client. Full details can be found in the ActiveMQ online documentation (ref. 3).

3.2 FTP

Darwin provides an FTP service, in order to give access to Timetable Data. The FTP service is only available to clients that access Darwin via a dedicated communications interface or an encrypted VPN (not via the Internet).

Each client that requires FTP access will be provided with the IP address(es) of the FTP server, a username and password. The account will be read-only and clients cannot write or delete files.

3.3 Amazon S3

An Amazon S3 "bucket" is provided to give an alternate means of access to Timetable Data files (section 6). This service will be available over the Internet.

Each client that requires S3 access will be provided with the URL of the "bucket". Access is limited to IP address white-listed clients. The access will be read-only and clients cannot write or delete files.

3.4 Availability

The Push Port service is available continuously, 24x365. Data is provided as soon as it is updated, other than when Darwin is re-building its timetable, when it may be queued for a short time. Darwin typically re-builds its timetable at 02:00 each day, though this may possibly occur at other times due to operational reasons.

4. Configuration

In order to use the ActiveMQ Push Port, various configuration items are required. Some of these are defined by Thales outside the client's control, and some are defined by the client.

The Thales configuration includes:

- Access credentials
- Port type: *Unfiltered* or *Filtered*
- ClientID
- DCIS 'Instance ID' (if applicable)
- Message Filtering restrictions (if applicable)

Each port is identified by its unique ClientID. This is analogous to the TCP port number used in the original socket-based Push Ports. For example, a client that had a socket-based Push Port on port 20000 might be given a ClientID of "Port20000". However, the ClientID is just an arbitrary string, so there is no significance in the actual value used. The ClientID must be used to set the equivalently named ActiveMQ Connection property, and is also used in the names of certain queues.

The Push Port service supports the filtering of data for a client, as controlled by the above "Port type" configuration item. If a client is defined as *Unfiltered*, no further configuration is needed.

For *Filtered* clients, the additional configuration is defined by the client entering the data into a Darwin web site.

4.1.1 Filtered Push Ports

4.1.1.1 Configuration Web Site

The client configuration data for Filtered Push Ports is managed by logging into the following Darwin web site:

```
https://realtime.nationalrail.co.uk/PPConfig/
```

Note. This URL is for live Push Ports only. Staging Push Ports have a similar web site on the applicable staging system.

The credentials used to access the web site are the same as those given to access the ActiveMQ Push Port.

Once logged into the configuration web site, a user may create and edit the following filter criteria:

- Schema Version
- "New" or "Old" reason codes (default "new")

- TIPLOC Filtering
- TD Filtering
- Message Type Filtering

Saving any of the filter criteria will apply that filter to the running Push Port immediately.

4.1.1.2 Schema Version

A filtered client must identify which Push Port data schema version they require. A user of the web site will be presented with a list of all of the supported data schema versions and one of them must be selected. On creation of a new client configuration, the schema version will default to the latest version available at that time.

If the user changes the schema version and saves it, the system will immediately automatically reconfigure to start sending data in the new version. This data will be written to a new version-specific queue and the message sequence number will be reset (see section 5.2). The queue associated with the previous version will continue to exist (for the time being), but no further data will be written to it.

Note that filtering configuration is independent of the schema version requested. Filtering may be used with any supported schema versions and changing schema version will not affect filtering.

4.1.1.3 Reason Codes

For historic compatibility reasons, the Push Ports may be configured to publish delay and cancellation reasons with different coded values. By default, the Push Ports always publish the “new” reason code mappings (>=500), but for certain clients these codes may be translated to be compatible with the “old” reason codes (<500). New clients must always use “new” reason codes.

Note that changing the reason codes setting will also reset the message sequence number (see section 5.2) and will require a snapshot to be performed in order to maintain full synchronisation.

4.1.1.4 TIPLOC Filtering

Push Ports can be filtered by a set of TIPLOC locations.

When a port is filtered by location, the server will only send data to a client that is relevant to one or more of the filter locations, or has been explicitly activated by the client associated with the port using the DCIS Web Service interface. Thus, updates for a service will be sent if that service calls at one of the filter locations (or is associated with a service that calls at a filter location). Station messages (<OW> elements) will only be sent if they apply to a filter location. Note that this applies equally to both “normal” update messages and snapshot messages. However, also note that the download-able Timetable files (see section 6) will not be filtered and will always contain information for all services and locations.

Once a filtered port sends data for a service, it will continue to send updates to that service even if the service no longer meets the filter criteria. Thus, if a service is edited to call at a filter location, it will be sent to the client. If that service is subsequently edited to remove that filter location, the updates to the service will still be sent to the client. Once a service is associated with a filtered port, it will not be un-associated until the service is removed from the Darwin timetable.

By default, when a new *filtered* client configuration is created, no TIPLOC filter criteria will be defined. This means that *no* location-based updates will be sent to the client until the configuration is edited to define the TIPLOC filter criteria. However, any explicit activations by DCIS clients will still be sent to them.

Normally, a client may only filter on calling points (i.e. locations with a passenger activity). Passing services, where a train does not stop at a filtered location, will not be returned. However, it is possible for Thales to enable configuration to allow passing trains to be sent. If a client requires such configuration, then a request should be submitted to RDG.

TIPLOC filter criteria are edited by selecting one or more TIPLOC locations from a list of all TIPLOCs known to Darwin. If a location has more than one TIPLOC associated with it, then all TIPLOC values must be supplied in order for all data to be received.

4.1.1.5 TD Filtering

Push Ports can be filtered by a set of Train Describer (TD) Area identifiers.

When a port is filtered by TD Area, the server will only send data to a client that is relevant to one or more of the filter TD Areas.

The exact data that is received is documented in the Push Port Data Specification (reference 2). However, in order to receive any of this data, a client must explicitly request those TD area codes for which they require data. By default, when a new *filtered* client configuration is created, no TD filter criteria will be defined. This means that *no* TD-based updates will be sent to the client until the configuration is edited to define the TD filter criteria.

TD filter criteria are edited by selecting one or more TD Area codes from a list of all TD Area codes known to Darwin.

4.1.1.6 Message Type Filtering

Push Ports can be filtered by message type. The following message types may be filtered:

- Station (OW) messages
- Train (adhoc) Alert messages
- Train Order messages
- Tracking ID messages
- Alarm messages

Filtering of Forecast (TS) messages is also possible, but requires special Thales configuration to be manually set up. Clients that need such configuration should submit a request, with their usage scenario, to RDG or the Darwin Service Desk.

Note that not all message types are available in all schema versions. If a message type is requested to be sent, but is not supported by the configured schema version, then the configuration will be ignored.

It is possible that certain message filter settings will be restricted in Thales configuration by making them read-only so they cannot be changed by the user.

By default, when a new *filtered* client configuration is created, all non-read-only message types will be sent.

5. Protocol

5.1 Message Queues

The Push Ports will write data messages to each of a set of ActiveMQ message queues. These messages will be written continuously, whether a client is connected or not.

In order to limit resource usage, and prevent clients from receiving very out of date data, each message will have relatively short message lifetime (in the order of 10 minutes).

If a client disconnects from a message queue and reconnects again before the oldest message expires then all of the messages queued in the interim will still be delivered. However, note that the message expiry is based on clock times, so it is important that client clocks have accurate times. For example, if the client clock is 11 minutes fast then no messages will be received as they will appear to have expired when they are delivered to the client.

If a client is normally not interested in the Push Port data (for example, a backup server), it is preferable for it to read the data off the queue anyway and discard it. Not only will this improve resource usage on the Darwin ActiveMQ servers, it will prevent the client receiving a full queue of 10 minute old data when they do finally connect.

The name of the queue that a client will connect to is specific to that client, and depends upon whether the client is configured to be an *Unfiltered* or *Filtered* client.

5.1.1 Unfiltered Clients

The queue name that an *Unfiltered* client will connect to has the general format:

```
Consumer.{CLIENTID}.VirtualTopic.PushPort-v{VERSION}
```

where {CLIENTID} is the ClientID allocated in the Thales configuration for this client, and {VERSION} is the version ID of the Push Port schema version required, e.g. “14”.

So, for example, if a client has a ClientID of “Port20000” and requires data conforming to schema version “9”, they will connect to queue:

```
Consumer.Port20000.VirtualTopic.PushPort-v9
```

Alternatively, if an *Unfiltered* client requires Push Port data with reason codes mapped to “old” values (see section 4.1.1.3) then a connection to the following queue name should be made:

```
Consumer.{CLIENTID}.VirtualTopic.PushPort-OldR-v{VERSION}
```

5.1.2 Filtered Clients

The queue name that a *Filtered* client will connect to has the general format:

```
Filtered.{CLIENTID}.PushPort-v{VERSION}
```

where {CLIENTID} is the ClientID allocated in the Thales configuration for this client, and {VERSION} is the version ID of the Push Port schema version required, e.g. “14”.

So, for example, if a client has a ClientID of “Port20000” and requires data conforming to schema version “9”, they will connect to queue:

```
Filtered.Port20000.PushPort-v9
```

Note that, unlike the case for an *Unfiltered* client, there isn’t a separate queue name for “new” or “old” reason codes. Messages will be queued to this same queue with the appropriate reason code mapping, according to the setting of the configuration data.

5.2 Message Sequence Numbers

Each message added to a message queue will be given a *Message Sequence Number*. The *Message Sequence Number* will be added to the message as an ActiveMQ message property called “PushPortSequence”. To be absolutely clear, this is a property of the *message itself*, not part of the data within the message.

This property will start at zero and wrap back to zero after reaching 9,999,999 (i.e. modulus 10,000,000). The sequence number may also be reset back to zero if the Darwin Push Port services are restarted, or possibly for other reasons.

The purpose of the sequence number is to detect if messages are lost for any reason. If a message is received whose sequence does not follow on from the previous message then it should be assumed that messages may have been lost and a snapshot will be required to re-synchronise.

5.3 Connection and Reconnection

A client will make a connection to the ActiveMQ server, as detailed in section 3.1. Once successfully connected, the client queue should be opened, as detailed in section 5.1.

Connection for the first time, or after a long outage, can be handled with the same procedure as a reconnection after a short outage, as follows:

1. Read the first message from the queue.

2. If the sequence number of that message is the next expected sequence number after the last message successfully received from the previous connection then no messages have been lost during the disconnection and this and subsequent messages can now be processed normally.

   This of course requires that the previous sequence number is remembered by the client application. On the first connection attempt, it is suggested that the “last sequence number” is initialised to an invalid value (such as -2) to force a failed match.

3. If the sequence number does not follow on from the last known one, then a snapshot will be required, as detailed in section 5.4, and the current message can be discarded.

5.4 Snapshots

When a client has determined that a snapshot is required then the following procedure should be performed.

Note that the entire snapshot process must be completed within the expiry time for messages (see section 5.1), otherwise data will be lost and another snapshot will be required – presumably having the same result again.

If this is a problem, it is suggested that snapshots and messages are simply copied to some local queues without any expiry times before being processed as needed. If using ActiveMQ for these local queues, it is suggested that using Apache Camel (which is built into ActiveMQ) as a message bridge be considered.

The ActiveMQ Push Port interface supports only **Standard Snapshots** and does not support snapshots via FTP.

5.4.1 Snapshot Queues

Darwin provides a single, writable *Snapshot Request* queue to request a snapshot. This queue is named:

```
Snapshot.Request
```

In addition, there are a set of separate *Snapshot Response* topics for *Unfiltered* and *Filtered* clients.

The topic name that an *Unfiltered* client will connect to has the general format:

```
Snapshot.Response-v{VERSION}
```

where {VERSION} is the version ID of the Push Port schema version required, e.g. "14".

Alternatively, if an *Unfiltered* client requires Snapshot data with reason codes mapped to "old" values (see section 4.1.1.3) then a connection to the following topic name should be made:

```
Snapshot.Response-OldR-v{VERSION}
```

The topic name that a *Filtered* client will connect to has the general format:

```
Snapshot.{CLIENTID}.Response-v{VERSION}
```

where {CLIENTID} is the ClientID allocated in the Thales configuration for this client, and {VERSION} is the version ID of the Push Port schema version required, e.g. "14".

As in the case of the filtered data queue, there isn't a separate *Snapshot Response* topic name for "new" or "old" reason codes. Messages will be queued to this same queue with the appropriate reason code mapping, according to the setting of the configuration data.

Note that the snapshot responses are written to ActiveMQ topics, not queues. This makes little difference from the interfacing perspective, but there are certain things of which to be aware. Firstly, messages will only be delivered to the client while the client is connected to the topic. If the client disconnects, any messages on the topic will be lost for that client. Secondly, topics can be shared between multiple clients. This means that snapshots may appear on the topic that were requested by another client (or multiple clients).

Unless a client has determined that a snapshot is required, the client *should not connect* to the snapshot queue/topic. This will ensure that the client will not be impacted by snapshots from other clients, which are not of interest.

5.4.2 Snapshot Process

The process to request and receive a snapshot must be performed as follows:

1. Connect to the Snapshot request queue and the appropriate Snapshot Response topic.
2. Send a Push Port “GetSnapshotReq” message to the Snapshot Request queue. Note that the “viaftp” attribute of these messages is not supported by the ActiveMQ service and will be ignored. See the appropriate Push Port data schema for details of these messages.
3. Read messages from the Snapshot Response topic until the snapshot has completed.
4. Read and discard messages from the Push Port data queue until the correct snapshot marker is found.
5. Close the Snapshot request queue and the appropriate Snapshot Response topic.
6. Resume reading and processing the Push Port data queue.

Any message other than a valid Push Port “GetSnapshotReq” message (including a valid “GetFullSnapshotReq” message) that is sent to the Snapshot request queue will be silently ignored.

The Push Port service will synchronise the snapshot with a point in the Push Port data queue stream of messages. The snapshot that is generated will contain all of the data that has been output prior to that point in the stream, and will not contain any data that is generated after that point.

In order to indicate the synchronisation point, the Push Port service will insert a data schema “SnapshotId” message into each Push Port data queue. The SnapshotId message will contain a unique *Snapshot ID*, which is used to correlate each snapshot.

The snapshot itself will be generated to the Snapshot response topics. Each snapshot will begin with a data schema “SnapshotId” message and end with another “SnapshotId” message. Each SnapshotId message will contain the unique *Snapshot ID*. In addition, each message will have two ActiveMQ message properties called “SS_BOUNDARY” and “SS_REQUESTER”.

The SnapshotId message at the start of the snapshot will set SS_BOUNDARY to “Start”, and the one at the end will be set to “End”, to indicate the boundaries of the snapshot.

The SS_REQUESTER property will be set to a space separated list of ClientIDs that have requested this snapshot. A single snapshot response may satisfy multiple requests. A client that has requested a snapshot should look in this property for its own ClientID. If its ClientID is not found, this snapshot should be ignored and the client should discard all of the snapshot messages until another snapshot is found with the correct ClientID, corresponding to their request.

Note that a snapshot will consist of many individual <sR> elements and messages, containing the snapshot data. This is in contrast to other (non-message queue-based) interfaces where the snapshot is generated within a single <sR> element. However, the actual data in the snapshot is the same in all cases.

When a snapshot is requested, a client should read and discard all Push Port data queue messages until the next “SnapshotId” message is found. If the *Snapshot ID* matches the *Snapshot ID* from the snapshot topic, then the synchronisation point has been found and the client should stop reading further messages until the snapshot has been fully processed. If the *Snapshot ID* does not match, then further messages can be discarded until the correct “SnapshotId” message is found.

Messages on the Snapshot response topics are given sequence numbers, in the same way as the data queues. The only difference is that the sequence number is initialised to 0 at the start of each snapshot.

As can be inferred from the above description, it is possible for a client to receive a “SnapshotId” message in the Push Port data when no snapshot has been requested. This will be because another client has requested the snapshot and this is their synchronisation marker. Such unsolicited messages should simply be ignored.

5.5 Darwin Timetable Rebuild

A Darwin timetable data rebuild occurs on a nightly basis. Currently, this is configured to occur at 2:00 am, although this is subject to change and should not be relied upon. During the rebuild, Darwin will not issue any update messages and the heartbeat (section 5.6) will indicate that the database is being re-initialised (HBINIT). Once the database rebuild has completed, Darwin will notify the Client that the database is available via the heartbeat message (HBOK).

It is advantageous for clients to remain connected to the Push Port queue during the rebuild process. After the rebuild completes, Darwin will immediately push out any relevant new or changed data, and remaining connected will remove any need to perform a snapshot.

5.6 Darwin Status Messages

A status message with the current state of the interface (HBOK, HBPENDING, HBINIT or HBFAIL) is sent every 60 seconds if no other data is sent to the client in that time. This message is also sent immediately if the state of the interface has changed.

The status messages that may be received by an ActiveMQ client are:

Code	Type	Text	Description
HBOK	Heartbeat; sent periodically.	System is available	Darwin is running and able to accept requests for data.
HBINIT	Heartbeat; sent periodically.	System is initialising	Darwin is running but is initialising its timetable. Clients should wait until a HBOK message is received.
HBFAIL	Heartbeat; sent periodically.	System is unavailable	Darwin is shutdown (the push port handler is a separate process from the core Darwin process).

HBPENDING	Heartbeat; sent periodically.	System is failing over and data is delayed	Darwin is operating, but part of the system is currently in failover mode. Data may be queued for a short period. Clients may remain connected and data will be delivered when available. This status is only returned in data schema version 11 and later.

Note that additional status messages that can be returned in response to client requests in other Push Port communication protocols will not be sent to ActiveMQ clients and therefore do not appear in the table above.

The general format of Darwin Data Phase status messages is as follows:

```
<?xml version="1.0"?>
<Pport  …>
  <FailureResp code="HBOK">
    System is available
  </FailureResp>
</Pport>
```

From version 11 of the data schema, DCIS clients can request (via the DCIS web service interface) a heartbeat operation, to verify full end-to-end operation. When responding to a heartbeat request, a <FailureResp> message will include optional “requestSource” and “requestID” attributes. These attributes allow a client to detect that this heartbeat message was generated as the result of the DCIS web service request made by that client, verifying end-to-end operation. Only the client that requested the heartbeat, as determined by the “requestSource” attribute, will receive the message. The “requestID” attribute is an optional value provided by the DCIS client with their heartbeat request.

The status returned for a heartbeat will reflect the current state of the system, as will be returned in the next regular status message (assuming the state does not change in the meantime). During some internal Darwin failover scenarios, heartbeat messages may be lost, even though the regular status messages appear to indicate that the system is available.

6. Timetable Files

Darwin makes available timetable files to registered clients. The format and contents of these files is detailed in the Push Port Data Specification (reference 2) and the Timetable XML schema files.

6.1 Timetable IDs

A Timetable ID is used to identify the currently available timetable files.

The ActiveMQ versions of the Push Ports will <u>not</u> wait for Timetable files to be fully available before transitioning from a HBINIT status to HBOK status. When this transition occurs, the Timetable files may still be in the process of being generated. However, note that it is not guaranteed that the Timetable ID will change after every HBINIT status, so there may be no new Timetable files at all.

To indicate that a new timetable is available, the ActiveMQ Push Ports will proactively send a “TimetableId” message, to identify the new Timetable file names. Clients that do not require the timetable should simply ignore this message.

A separate “TimetableId” message will be sent for each individual Timetable file that becomes available. Thus, multiple “TimetableId” messages will be generated in succession, one for each Timetable and Reference file schema version.

Note that due to existing schema limitations, the “TimetableId” message has mandatory attributes for timetable file and timetable reference data file names. Since the “TimetableId” notification message is only reporting the presence of a single file, only one of these attributes will be populated with a valid file name. The other attribute will consist only of white space. In future schema versions, these attributes may become optional, so that they may be omitted when not relevant.

Since all schedules are now “activated” before any other data is published for them, it is safe to process the timetable in parallel with processing real time updates. The only need for the timetable is to gain access to those schedules that will be activated in the future.

If the client has been disconnected for a significant amount of time (such that messages may have been lost and a snapshot is therefore required), or on first connection, the previous “TimetableId” message may have been missed. To check this, the client should look for the latest Timetable files and compare the Timetable ID against what the client expects. If the Timetable ID differs then the latest timetable should be downloaded and processed.

Details of where to download and how to process the timetable files can be found in the Push Port Data Specification (reference 2).

7. Abbreviations and Glossary

ATOC	Association of Train Operating Companies
CIF	Common Interface File. The format of this file defines the format in which ITPS provides schedule information.
CRS	Computerised Reservation System
DCIS	Darwin CIS. Interface between Darwin and CIS systems driving passenger displays at stations.
False Destination	A train destination. Typically used in a circular route to provide a route for the train.
FTP	File Transfer Protocol.
Gzip	A compression tool using the DEFLATE format as defined in RFC 1951. gzip is defined in RFC 1952.
ITPS	Integrated Train Planning System (replacement source of schedule data for TSDB).
LDB	Live Departure Boards. This is the publicly available web interface to the Darwin system.
RDG	Rail Delivery Group (Previously ATOC NRE)
RID	Darwin generated ID. A unique ID held within the Darwin database to identify a journey.
RTTI	Real Time Train Information – database of train running information. Previous name for Darwin.
Snapshot	There are two types of snapshot: 1. Standard Snapshot: information for all train journeys in the Darwin database that are in progress or have yet to commence. 2. Full Snapshots: includes information for all journeys in the Darwin database since the last timetable rebuild. I.e. includes the standard snapshot plus historic schedule information. Snapshot data is defined in reference 2. Full Snapshots are not supported by the ActiveMQ interface.
Theseus	Theseus supplies train-running data received from TOC's to Darwin.
TIPLOC	Timing Point Location
TOC	Train Operating Company
TSDB	Train Services Database (now superseded by ITPS).
Updates	Darwin provides update information to the Client when this information becomes known to Darwin. Update data is defined in reference 2.
UID	Unique Identifier. (However, the UID is not always unique within the Darwin database)