Adapter Remoting Infrastructure

Network Protocol Specification

ARI Protocol version: 1.9.1
Target: Lightstreamer Server v. 7.3 or greater
Last updated: 14/2/2023

lightStreamer

Table of contents

1 The ARI Protocol....3
1.1 Architecture....3
1.2 Protocol Basics....3
1.3 Packet Structure....4
1.4 Data Types and Encodings....5
1.5 Metadata Provider Protocol Packets....8
1.6 Data Provider Protocol Packets....24
1.7 Common Packets....29
1.8 Notes on Protocol Method Sequence....31

1 The ARI Protocol

1.1 Architecture

For an overview of the architecture of the Adapter Remoting Infrastructure and the necessary server configurations, please refer to the "Adapter Remoting Infrastructure.pdf" file, available in the Lightstreamer distribution under the "docs" folder (or the "adapter_remoting_infrastructure" folder for Server version earlier than 7.2).

The ARI protocol actually consists in two protocols, that apply to two different communications:

- between a Proxy Metadata Adapter and a Remote Metadata Adapter (the **Metadata Provider Protocol**);
- between a Proxy Data Adapter and a Remote Data Adapter (the **Data Provider Protocol**).

The two protocols share the same syntax and rules (hence, we refer generically to a single ARI protocol) but differ in the types of packets to be exchanged.

1.2 Protocol Basics

The following are the basic principles of the ARI protocol:

- There are two unidirectional channels, named requests and replies; the **requests channel** hosts protocol packets sent from a Proxy Adapter; the **replies channel** hosts protocol packets sent from the counterpart (i.e.: the Remote Adapter).
- Protocol packets belong to various categories: **request** and corresponding **reply** packets; **notification** packets, which are originated by a previous request; and **keepalive** packets.
- There are various types of request packets, each with a corresponding reply type, and of notification packets. There are also a few special cases of request types with no corresponding reply type and vice-versa.
- In general, requests pertain to the requests channel, whereas replies and notifications pertain to the replies channel. However, there are also a few **backward requests**, that can be issued by the Remote Adapter in the replies channel and may require a reply from the Proxy Adapter to be issued in the requests channel. Keepalive packets pertain to either channel.
- The stream associated to each channel is made by UTF-8 characters; since Server version 7.4, a further BOM character at the beginning of the reply stream is tolerated and discarded, but not recommended.
- Every request type and the corresponding reply type are characterized by a **method tag**, which refers, more or less directly, to methods of the Java In-Process Adapter SDK interface. Similarly for notification types.
- Every request is identified by a unique alphanumeric ID, that must be repeated as the ID of its corresponding reply. This also holds for backward requests issued by the Remote Adapter and the corresponding replies; in this case, the Remote Adapter is responsible for the choice of a unique alphanumeric ID, which is independent from the IDs used by the Proxy Adapter. For notifications, the ID of the request that originates the notification may be reported.

- Each notification is time stamped, for statistical purposes, with a Java compatible millisecond resolution date-time (i.e. System.currentTimeMillis).
- All protocol packets are simple pipe-separated text lines, terminated by a line-feed or carriage-return & line-feed pair; all text is encoded in UTF-8.
- Requests and replies are completely asynchronous, i.e.: there is no need to answer a request before reading, and possibly answering, the next one.
- Keepalive packets, originated by a Remote Adapter, are available, in order to prevent network intermediate nodes from dropping the connections because of inactivity. The keepalive packets are simpler than request, reply and notification packets.

This implies that:

- in order to ensure that data can be transported by text lines, they must be encoded accordingly;
- in order to ensure that correct decoding can be applied for each data segment, corresponding data types must be specified;
- data values that represent text may need to be properly escaped;
- in order to ensure asynchronous request/reply, a pair of queue & de-queuing-thread must be created for each channel;
- in order to ensure correct statistical treatment of notification timestamps, in case the counterpart runs on a separated machine the two machines must be synchronized through NTP or equivalent network time protocol (this is actually optional, statistics can be ignored and timestamp set to 0).

1.3 Packet Structure

The general structure of a protocol packet is as follows:

- request and reply packets:

```
<ID>|<method>|<data type 1>|<data segment 1>|...|<data type N>|<data segment N>\r\n
```

- notification packets:

```
<timestamp>|<method>|<data type 1>|<data segment 1>|...|<data type N>|<data segment N>\
r\n
```

- keepalive packets:

```
KEEPALIVE\r\n
```

Where the lines may also terminate with just `\n` , possibly depending on the system the originating peer runs on.

Each data segment typically represents the content of a field or argument of a method, and the corresponding data type represents the native type in which it is expressed. There are a few cases where

the structure above is not respected, particularly in the case of handling of counterpart-generated exceptions. See on for more details.

1.4 Data Types and Encodings

String

Data type: `S`
Two different types of encoding are possible:

- "Smart Encoding", used in most of the packets:
  - `#` if null
  - `$` if empty
  - in any other case, a sequence of characters encoded in UTF-8 with possible percent-encoded characters;
    percent-encoding is also based on UTF-8, but, actually, only the following ascii characters require percent-encoding:
    - \r
    - \n
    - |
    - %
    - +

    moreover, the following characters require percent-encoding only when they are in a string of length 1:
    - $
    - #

- "Backward-Compatibility Encoding", used in the initialization packets:
  - `#` if null
  - `$` if empty
  - standard WWW UTF-8 url-encoding in any other case

Note that the two encodings are similar, as the Smart Encoding just supports both UTF-8-percent-encoded and plain UTF-8 form for most characters for which the Backward-Compatibility Encoding only requires UTF-8-percent-encoding. This allows for the use of shorter representations.
However, the Backward-Compatibility Encoding is not just a special case of the Smart Encoding, since the WWW UTF-8 url-encoding specifications also require that the space character is represented as a '+' character, which is not part of Smart Encoding.
Nevertheless, since the '+' character is percent-encoded by Smart Encoding, a string encoded in this way can be decoded by a standard URLDecode algorithm, provided that it accepts also unencoded characters.

As said, the need for one or the other encoding is related with the underlying packet type. We will specify when a packet type requires the Backward-Compatibility Encoding. Where not specified, the Smart Encoding will be required.

Boolean

Data type: `B`
Encoding:

- `0` if false
- `1` (or any other value) if true

Integer (32 bit, signed)

Data type: `I`
Encoding: simple string representation of the integer

Long (64 bit, signed)

Data type: `L`
Encoding: simple string representation of the long integer

Double (64 bit, signed, IEEE 754 formatted)

Data type: `D`
Encoding: simple string representation of the double, with decimal point (not comma)

In addition to these simple native types there are structured types, like exceptions and mode arrays, and the “no data” type:

Void (i.e.: no data)

Data type: `V`
Encoding: not encoded, data segment is simply not present

Mode array

Data type: `M`
Encoding:

- `#` if null
- `$` if empty
- a character sequence in any other case, where each character represents a mode as per the following table:

Character	Mode
`R`	Raw
`M`	Merge
`D`	Distinct
`C`	Command

Diff Algorithm array

Data type: `F`
Encoding:

- `#` if null
- `$` if empty
- a character sequence in any other case, where each character represents a “diff” algorithm as per the following table:

Character	Algorithm
`J`	JSON Patch
`M`	diff-match-patch

Exception

Exceptions can be of generic type or of different subtypes, as per the following table:

Type of exception	Data type	Encoding
Generic	`E`	string-encoding of detail message
Failure	`EF`	string-encoding of detail message
Metadata	`EM`	string-encoding of detail message
Data	`ED`	string-encoding of detail message
Subscription	`EU`	string-encoding of detail message
Access	`EA`	string-encoding of detail message
Items	`EI`	string-encoding of detail message
Schema	`ES`	string-encoding of detail message
Notification	`EN`	string-encoding of detail message
Credits	`EC`	3 subsequent data segments with: • string-encoding of detail message • integer-encoding of client error code • string-encoding of user message
Conflicting session	`EX`	4 subsequent data segments with: • string-encoding of detail message • integer-encoding of client error code • string-encoding of user message • string-encoding of conflicting session ID
Resource unavailable	`ER`	string-encoding of detail message

String-encoding can be based on either Smart Encoding or Backward-Compatibility Encoding, depending on the underlying packet type.

Mobile Platform Type

Data type: `P`
Encoding: a character as per the following table:

Character	Platform
A	Apple Platforms (iOS, macOS, tvOS, etc.)
G	Google Platforms (Android, Chrome, etc.)

The available cases refer to the platform types currently supported.

1.5 Metadata Provider Protocol Packets

Metadata Provider protocol aims at exposing the methods of the MetadataProvider class, of the Java In-Process Adapter SDK interface, through the protocol. To reduce the protocol overhead, much of the original Java interface small methods have been aggregated in bigger protocol packets, carrying more information that are then cached on the Proxy Metadata Adapter side.
The callback methods provided by the Java interface are mapped to backward requests.

For more explanations regarding the semantics of each information, please consult the interface API documentation, either on Java In-Process Adapter SDK API docs or (for an implementation example) on .Net Remote Adapter SDK API docs.

Not all methods available in the Java interface have been exposed for Remote Adapters. The Selectors and Customizers, which involve invocations of *isSelected* and *customizeUpdate* in the inner loop of data dispatching, cannot be used. As a consequence, *isSelectorAllowed* and *enableUpdateCustomization* are implemented by the Proxy Metadata Adapter to always return false.

The Metadata Provider protocol only consists in requests and backward requests and the corresponding replies; hence no notifications are included. We will refer to its channels as the **Metadata requests channel** and the **Metadata replies channel**.
In practice, the Proxy Metadata Adapter and the Remote Metadata Adapter communicate through a TCP connection carrying the Metadata requests channel and the Metadata replies channel as the two streams.

Metadata Init

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `MPI`

Data segments:

- N *initialization parameter name-value pairs*, both names and values as string; all strings should be encoded with the Backward-Compatibility Encoding.

  The parameters are supplied by the Proxy Adapter based on its own configuration in an unspecified order.
  A parameter named “ARI.version” will always be supplied, to carry the maximum protocol version supported by the Proxy Adapter.
  A parameter named “keepalive_hint.millis”, if present, specifies the inactivity time after which a keepalive packet should be issued on the replies channel to prevent the

Proxy Adapter from closing the connection for inactivity. Note that the value is a number, but encoded as a string.

```
<ID>|MPI|S|<param 1>|S|<param 1 value>|...|S|ARI.version|S|<version>|...|S|
<param N>|S|<param N value>
```

Expected reply data segments:

- N *initialization parameter name-value pairs*, both names and values as string; all strings will be encoded with the Backward-Compatibility Encoding.

  The supplied parameters are handled by the Proxy Adapter based on its own configuration and regardless of their order.
  A parameter named “ARI.version” is mandatory, to carry the protocol version that the Remote Metadata Adapter is willing to use. Based on it, the Proxy Adapter will either proceed or interrupt the connection. If it is lower than the request’s ARI.version, the Proxy Adapter may still support it and proceed. If it is higher, the Proxy Adapter will interrupt (the Proxy Adapter will recognize the response, regardless of its protocol version); in this case, returning a Generic Exception (see below) is also an option.

```
<ID>|MPI|S|<param 1>|S|<param 1 value>|...|S|ARI.version|S|<version>|...|S|
<param N>|S|<param N value>
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* or *Metadata* type (based on the Backward-Compatibility Encoding for strings). This will cause the Proxy Adapter to interrupt the connection. The Proxy Adapter will recognize such response, regardless of its protocol version.

Examples:

```
→      10000010c3e4d0462|MPI|S|ARI.version|S|1.8.2|S|keepalive_hint.millis|S|8000|S|
adapters_conf.id|S|DEMO|S|proxy.instance_id|S|hewbc3ikbbctyui|...
←      10000010c3e4d0462|MPI|S|ARI.version|S|1.8.2
```

```
→      10000010c3e4d0462|MPI|S|ARI.version|S|1.8.2|S|adapters_conf.id|S|DEMO|S|prox-
y.instance_id|S|hewbc3ikbbctyui|...
←      10000010c3e4d0462|MPI|S|ARI.version|S|1.8.2|S|user|S|remote1|S|password|S|fd-
hjkslghak
```

```
→      20000010c3e4d0462|MPI|S|ARI.version|S|1.8.2|S|custom:feed_name|S|feed1|S|cus-
tom:feed_port|S|8080|...
←      20000010c3e4d0462|MPI|EM|Authentication+Feed+unavailable
```

Notify User

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NUS`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)

- 1 *user password*, as string (can be null if no password is specified)
- N *http header name-value pairs*, both names and values as string
  The last pair is added by the Server; the name is “REQUEST_ID” and the value is a unique id assigned to the client request.

```
<ID>|NUS|S|<user name>|S|<user password>|S|<header 1>|S|<header value 1>|...|S|
<header N>|S|<header value N>|S|REQUEST_ID|S|<id>
```

Expected reply data segments:

- 1 *allowed max bandwidth*, as double
  Edition note: Bandwidth Control is an optional feature, available depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).
- 1 *wants table (i.e. subscription) notifications flag*, as boolean

```
<ID>|NUS|D|<allowed max bandwidth>|B|<wants table notifications flag>
```

As you can see, this request also integrates the *getAllowedMaxBandwidth* and *wantsTablesNotification* methods of the Java interface.

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Access, ResourceUnavailable,* or *Credits* type; for Credits exceptions, the error code must be either 0 or negative.

Examples:

```
→       10000010c3e4d0462|NUS|S|user1|S|password|S|host|S|www.mycompany.com|...
←       10000010c3e4d0462|NUS|D|40|B|0

→       20000010c3e4d0462|NUS|S|#|S|#|S|connection|S|Keep-Alive|...
←       20000010c3e4d0462|NUS|EC|Anonymous+user+not+allowed|-1099|#
```

Notify User (extended version to carry identification data included in the client SSL certificate)

Edition note: https connections is an optional feature, available depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).
Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NUA`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- 1 *user password*, as string (can be null if no password is specified)
- 1 client principal, as string (can be null if client not authenticated)
- N *http header name-value pairs*, both names and values as string
  The last pair is added by the Server; the name is “REQUEST_ID” and the value is a unique id assigned to the client request.

```
<ID>|NUA|S|<user name>|S|<user password>|S|<client principal>|S|<header 1>|S|
<header value 1>|...|S|<header N>|S|<header value N>|S|REQUEST_ID|S|<id>
```

Expected reply data segments:

- 1 *allowed max bandwidth*, as double
  Edition note: Bandwidth Control is an optional feature, available depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).
- 1 *wants table (i.e. subscription) notifications flag*, as boolean

```
<ID>|NUA|D|<allowed max bandwidth>|B|<wants table notifications flag>
```

As you can see, this request, like its base version, also integrates the *getAllowedMaxBandwidth* and *wantsTablesNotification* methods of the Java interface.

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Access, ResourceUnavailable,* or *Credits* type; for Credits exceptions, the error code must be either 0 or negative.

Examples:

```
→      10000010c3e4d0462|NUA|S|user1|S|password|S|cn=john,cn=users,dc=acme,dc=com
|S|host|S|www.mycompany.com|...
←      10000010c3e4d0462|NUA|D|40|B|0

→      20000010c3e4d0462|NUA|S|user1|S|password|S|#|S|connection|S|Keep-Alive|...
←      20000010c3e4d0462|NUA|EC|Unauthenticated+user+not+allowed|-1098|#
```

Notify New Session

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NNS`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- 1 *session ID*, as string
- N *client context name-value pairs*, both names and values as string
  The possible pairs differ from the Java interface notifyNewSession case, because the “HTTP_HEADERS” property is not provided; a “REQUEST_ID” property is provided instead (see notes below).

```
<ID>|NNS|S|<user name>|S|<session ID>|S|<context prop name 1>|S|<context prop 1
value>|...|S|<context prop name N>|S|<context prop N value>
```

Expected reply data segments: two possibilities are available:

- none (a void is expected)

```
<ID>|NNS|V
```

- the following:
  - 1 *time to live*, as integer

```
<ID>|NNS|I|<time to live>
```

As you can see, this request also integrates the *getSessionTimeToLive* method of the Java interface.

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Notification*, *Credits* or *Conflicting Session* type. In the last case, a second invocation of the command with the same "REQUEST_ID" and a different session ID will be received. For Credits or Conflicting Session exceptions, the error code must be either 0 or negative.

Examples:

```
→      30000010c3e4d0462|NNS|S|user1|S|S8f3da29cfc463220T5454537|S|REMOTE_IP|
S|192.168.0.1|...
←      30000010c3e4d0462|NNS|V

→      30000010c3e4d0462|NNS|S|user1|S|Sc4a1769b6bb83a4aT2852044|S|REMOTE_IP|
S|192.168.0.1|...
←      30000010c3e4d0462|NNS|I|300

→      40000010c3e4d0462|NNS|S|user1|S|S9cb4758037a95c01T0439915|S|USER_AGENT|S|#|...
←      40000010c3e4d0462|NNS|EX|No+more+than+one+session+allowed|-1101|#|
S8f3da29cfc463220T5454537
```

Notify Session Close

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NSC`

Data segments:

- 1 *session ID*, as string

```
<ID>|NSC|S|<session ID>
```

Expected reply data segments: none, a void is expected

```
<ID>|NSC|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* or *Notification* type.

Examples:

```
→      20000010c3e4d0462|NSC|S|S8f3da29cfc463220T5454537
```

```
←       20000010c3e4d0462|NSC|V

→       30000010c3e4d0462|NSC|S|S9cb4758037a95c01T0439915
←       30000010c3e4d0462|NSC|EN|Session+not+open
```

Force Session Termination

Direction: from counterpart to Proxy Adapter (backward request)
Expects reply: yes
Method tag: `FST`

Data segments: two possibilities are available

- short form
  - 1 *session ID*, as string
  - a void

```
<ID>|FST|S|<session ID>|V
```

- full form
  - 1 *session ID*, as string
  - 1 *cause code*, as integer
  - 1 *cause message*, as string

```
<ID>|FST|S|<session ID>|I|<cause code>|S|<cause message>
```

Expected reply data segments: none, a void is expected

```
<ID>|FST|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* type.

Examples:

```
←       20000010c3e4d0462|FST|S|S8f3da29cfc463220T5454537|V
→       20000010c3e4d0462|FST|V

←       30000010c3e4d0462|FST|S|S9cb4758037a95c01T0439915|I|-2|S|timeout+expired
→       30000010c3e4d0462|FST|E|internal+error
```

Get Items

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `GIS`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- 1 *item group name (or item list specification)*, as string

- 1 *session ID*, as string

```
<ID>|GIS|S|<user name>|S|<item group name>|S|<session ID>
```

Expected reply data segments:

- N *item names*, as string

```
<ID>|GIS|S|<item 1>|S|<item 2>|...|S|<item N>
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* or *Items* type.

Examples:

```
→       50000010c3e4d0462|GIS|S|#|S|nasdaq100_AA_AL|S|S8f3da29cfc463220T5454537
←       50000010c3e4d0462|GIS|S|aapl|S|atvi|S|adbe|S|akam|S|altr

→       60000010c3e4d0462|GIS|S|#|S|nasdaq100_AA_AL|S|S9cb4758037a95c01T0439915
←       60000010c3e4d0462|GIS|EI|Unknown+group
```

Get Schema

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `GSC`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- 1 *item group name (or item list specification)*, as string
- 1 *field schema name (or field list specification)*, as string
- 1 *session ID*, as string

```
<ID>|GSC|S|<user name>|S|<item group name>|S|<field schema name>|S|<session ID>
```

Expected reply data segments:

- N *field names*, as string

```
<ID>|GSC|S|<field 1>|S|<field 2>|...|S|<field N>
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Items* or *Schema* type.

Examples:

```
→       70000010c3e4d0462|GSC|S|#|S|nasdaq100_AA_AL|S|short|S|S8f3da29cfc463220T5454537
←       70000010c3e4d0462|GSC|S|last_price|S|time|S|pct_change

→       80000010c3e4d0462|GSC|S|#|S|nasdaq100_AA_AL|S|short|S|S9cb4758037a95c01T0439915
←       80000010c3e4d0462|GSC|ES|Unknown+schema
```

Get Item Data

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `GIT`

Data segments:

- N *item names*, as string

```
<ID>|GIT|S|<item 1>|S|<item 2>|...|S|<item N>
```

Expected reply data segments:

- N *item data structures*, composed by:
  - 1 *distinct snapshot length*, as integer
  - 1 *min source frequency*, as double
  - 1 *allowed modes*, as Mode array

```
<ID>|GIT|I|<dist. snapsh. len. 1>|D|<min source freq. 1>|M|<all. modes 1>|...
          ...|I|<dist. snapsh. len. N>|D|<min source freq. N>|M|<all. modes N>
```

As you can see, this request replaces the *getDistinctSnapshotLength*, *getMinSourceFrequency* and *modeMayBeAllowed* methods of the Java interface.
Note that empty or null Mode arrays are accepted but pointless, as preventing all subscriptions.

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* type.

Examples:

```
→      90000010c3e4d0462|GIT|S|aapl|S|atvi
←      90000010c3e4d0462|GIT|I|10|D|0|M|RMDC|I|30|D|0.01|M|R

→      a0000010c3e4d0462|GIT|S|aapl|S|atvi
←      a0000010c3e4d0462|GIT|E|Database+connection+error
```

Get User Item Data

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `GUI`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- N *item names*, as string

```
<ID>|GUI|S|<user name>|S|<item 1>|S|<item 2>|...|S|<item N>
```

Expected reply data segments:

- N *user item data structures*, composed by:
  - 1 *allowed buffer size*, as integer
  - 1 *allowed max frequency*, as double
    Edition note: A further global frequency limit could also be imposed by the Server, depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).
  - 1 *allowed modes*, as Mode array

```
<ID>|GUI|I|<all. buf. size 1>|D|<all. max freq. 1>|M|<all. modes 1>|...
       ...|I|< all. buf. size N>|D|<all. max freq. N>|M|<all. modes N>
```

As you can see, this request replaces the *getAllowedBufferSize*, *getAllowedMaxItemFrequency* and *isModeAllowed* methods of the Java interface.
Note that empty or null Mode arrays are accepted but pointless, as preventing all subscriptions.

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* type.

Examples:

```
→      b0000010c3e4d0462|GUI|S|user1|S|aapl|S|atvi
←      b0000010c3e4d0462|GUI|I|30|D|3|M|RMDC|I|30|D|0.3|M|$

→      c0000010c3e4d0462|GUI|S|#|S|aapl|S|atvi
←      c0000010c3e4d0462|GUI|E|Database+connection+error
```

Notify User Message

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NUM`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- 1 *session ID*, as string
- 1 *user message*, as string

```
<ID>|NUM|S|<user name>|S|<session ID>|S|<user message>
```

Expected reply data segments: none, a void is expected

```
<ID>|NUM|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Notification* or *Credits* type; for Credits exceptions, the error code must be either 0 or negative.

Examples:

```
→       d0000010c3e4d0462|NUM|S|user1|S|S8f3da29cfc463220T5454537|S|stop+logging
←       d0000010c3e4d0462|NUM|V

→       e0000010c3e4d0462|NUM|S|#|S|S9cb4758037a95c01T0439915|S|start+logging
←       e0000010c3e4d0462|NUM|EC|Anonymous+user+logging+not+allowed|-1095|#
```

Notify New Tables

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NNT`

Data segments:

- 1 *user name*, as string (can be null for the anonymous user)
- 1 *session ID*, as string
- N *table (i.e. subscription) info structures*, composed by:
  - 1 *window index*, as integer
  - 1 *publishing mode*, as 1-length Mode array
  - 1 *item group name (or item list specification)*, as string
  - 1 *data adapter name*, as string
  - 1 *field schema name (or field list specification)*, as string
  - 1 *index of the first item*, as integer
  - 1 *index of the last item*, as integer
  - 1 *selector*, as string (can be null if no selector is associated)
  - 1 *number of items*, as integer
  - M *item name*, as string, where M corresponds to the *number of items* specified above

```
<ID>|NNT|S|<user name>|S|<session ID>|
     I|<win. index 1>|M|<pub. mode 1>|S|<item group 1>|S|<data adapter 1>|
     S|<field schema 1>|I|<first item idx. 1>|I|<last item idx. 1>|S|<selector 1>|
     I|<number of items 1 := M1>|S|<item name 1.1>|...S|<item name 1.M1>|
     ...
     I|<win. index N>|M|<pub. mode N>|S|<item group N>|S|<data adapter N>|
     S|<field schema N>|I|<first item idx. N>|I|<last item idx. N>|S|<selector N>|
     I|<number of items N := MN>|S|<item name N.1>|...S|<item name N.MN>
```

(BTW selector information is currently redundant, as any request with a non-null selector has already been refused, since the Proxy Metadata Adapter always returns false to *isSelectorAllowed*)

Expected reply data segments: two possibilities are available:

- none (a void is expected)

  ```
  <ID>|NNT|V
  ```

- the following:
  - 1 *enable unsubscriptions flag*, as boolean

- 1 *wants final statistics flag*, as boolean

where a positive *enable unsubscriptions flag* is needed to enable the support of a subsequent `FUS` request on this subscription (however, if more than one *table info structures* is included, the `FUS` request will not be supported anyway);
on the other hand, a positive *wants final statistics flag* is needed to receive the full version of the final `NNT` request (which includes traffic statistics) for this subscription; otherwise, the reduced version of `NNT` will be received; note that the final statistics may, in some cases, increase the size of the notification significantly.

```
<ID>|NNT|B|<enable unsubscriptions flag>|B|<wants final statistics flag>
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Notification* or *Credits* type; for Credits exceptions, the error code must be either 0 or negative.

Examples:

```
→       f0000010c3e4d0462|NNT|S|#|S|S8f3da29cfc463220T5454537|I|1|M|M|
S|nasdaq100_AA_AL|S|QUOTES|S|short|I|1|I|5|S|#|I|5|S|aapl|S|atvi|S|adbe|S|akam|S|altr
←       f0000010c3e4d0462|NNT|V

→       f0000010c3e4d0462|NNT|S|#|S|Sc4a1769b6bb83a4aT2852044|I|1|M|M|
S|nasdaq100_AA_AL|S|QUOTES|S|short|I|1|I|5|S|#|I|5|S|aapl|S|atvi|S|adbe|S|akam|S|altr
←       f0000010c3e4d0462|NNT|B|0|B|1

→       10000010c3e4d0462|NNT|S|#|S|S9cb4758037a95c01T0439915|I|1|M|M|
S|nasdaq100_AA_AL|S|QUOTES|S|short|I|1|I|5|S|#|I|5|S|aapl|S|atvi|S|adbe|S|akam|S|altr
←       10000010c3e4d0462|NNT|EN|Session+timed+out
```

Notify Tables Close

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `NTC`

Data segments: two possibilities are available, depending on the response to the corresponding `NNT` request:

- reduced version
  - 1 *session ID*, as string
  - N *reduced table (i.e. subscription) info structures*, composed by:
    - 1 *window index*, as integer
    - 1 *publishing mode*, as 1-length Mode array
    - 1 *item group name (or item list specification)*, as string
    - 1 *data adapter name*, as string
    - 1 *field schema name (or field list specification)*, as string
    - 1 *index of the first item*, as integer
    - 1 *index of the last item*, as integer

        - 1 *selector*, as string (can be null if no selector is associated)
        - 1 *number of items*, as integer
        - M *item name*, as string, where M corresponds to the *number of items* specified above
        - 1 *number of item statistics == 0, as integer*

```
<ID>|NTC|S|<session ID>|
     I|<win. index 1>|M|<pub. mode 1>|S|<item group 1>|S|<data adapter 1>|
     S|<field schema 1>|I|<first item idx. 1>|I|<last item idx. 1>|S|<selector 1>|
     I|<number of items 1 := M1>|S|<item name 1.1>|...S|<item name 1.M1>|I|0
     ...
     I|<win. index N>|M|<pub. mode N>|S|<item group N>|S|<data adapter N>|
     S|<field schema N>|I|<first item idx. N>|I|<last item idx. N>|S|<selector N>|
     I|<number of items N := MN>|S|<item name N.1>|...S|<item name N.MN>|I|0
```

- full version with final statistics
    - 1 *session ID*, as string
    - N *full table (i.e. subscription) info structures*, composed by:
        - 1 *window index*, as integer
        - 1 *publishing mode*, as 1-length Mode array
        - 1 *item group name (or item list specification)*, as string
        - 1 *field schema name (or field list specification)*, as string
        - 1 *index of the first item*, as integer
        - 1 *index of the last item*, as integer
        - 1 *selector*, as string (can be null if no selector is associated)
        - 1 *data adapter name*, as string
        - 1 *number of items*, as integer
        - M *item name*, as string, where M corresponds to the *number of items* specified above
        - 1 *number of item statistics == M, as integer*
        - M *item statistics info structures,* composed by:
            - 1 *real-time events sent*, as long
            - 1 *lost events*, as long
            - 1 *filtered out events*, as long

```
<ID>|NTC|S|<session ID>|
     I|<win. index 1>|M|<pub. mode 1>|S|<item group 1>|S|<data adapter 1>|
     S|<field schema 1>|I|<first item idx. 1>|I|<last item idx. 1>|S|<selector 1>|
     I|<number of items 1 := M1>|S|<item name 1.1>|...S|<item name 1.M1>|
     I|<M1>|L|<real-time 1.1>|L|<lost 1.1>|L|<filtered 1.1>|
     ...L|<real-time 1.M1>|L|<lost 1.M1>|L|<filtered 1.M1>
     ...
     I|<win. index N>|M|<pub. mode N>|S|<item group N>|S|<data adapter N>|
     S|<field schema N>|I|<first item idx. N>|I|<last item idx. N>|S|<selector N>|
     I|<number of items N := MN>|S|<item name N.1>|...S|<item name N.MN>|
     I|<MN>|L|<real-time N.1>|L|<lost N.1>|L|<filtered N.1>|
     ...L|<real-time N.MN>|L|<lost N.MN>|L|<filtered N.MN>
```

(BTW selector information is currently redundant, as any request with a non-null selector has already been refused, since the Proxy Metadata Adapter always returns false to *isSelectorAllowed*)

Expected reply data segments: none, a void is expected

```
<ID>|NTC|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* or *Notification* type.

Examples:

```
→       f0000010c3e4d0462|NTC|S|S8f3da29cfc463220T5454537|I|1|M|M|
S|nasdaq100_AA_AL|S|QUOTES|S|short|I|1|I|5|S|#|
I|5|S|aapl|S|atvi|S|adbe|S|akam|S|altr|I|0
←       f0000010c3e4d0462|NTC|V

→       10000010c3e4d0462|NTC|S|S9cb4758037a95c01T0439915|I|1|M|M|
S|nasdaq100_AA_AL|S|QUOTES|S|short|I|1|I|2|S|#|
I|2|S|aapl|S|atvi|I|2|
L|150|L|230|L|0|L|23|L|0|L|0
←       10000010c3e4d0462|NTC|EN|Table+not+open
```

Force Unsubscription

Direction: from counterpart to Proxy Adapter (backward request)
Expects reply: yes
Method tag: `FUS`

Data segments:

- 1 *session ID*, as string
- 1 *window index*, as integer

```
<ID>|FUS|S|<session ID>|I|<window index>
```

Expected reply data segments:

- 1 *processed flag*, as boolean

  where a false *processed flag* indicates that the request was not supported, hence not processed at all; see `NNT` for a resume of the conditions for the support of this request.

```
<ID>|FUS|B|<processed flag>
```

Note that in the Java interface the corresponding method pertains to the proper *TableInfo* object.

Reply can also be an exception of *Generic* type.

Examples:

```
←       20000010c3e4d0462|FUS|S|S8f3da29cfc463220T5454537|I|11
→       20000010c3e4d0462|FST|B|1

←       30000010c3e4d0462|FST|S|S9cb4758037a95c01T0439915|I|12
```

```
→       30000010c3e4d0462|FST|E|internal+error
```

Notify MPN Device Access

Edition note: Push Notifications is an optional feature, available depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `MDA`

Data segments:

- 1 *user name*, as string
- 1 *session ID*, as string
- 1 *mobile platform type*
- 1 *application ID* (a.k.a. *package name*), as string
- 1 *device token* (a.k.a. *registration ID*), as string

```
<ID>|MDA|S|<user name>|P|<platform type>|S|<application ID>|S|<device token>
```

Expected reply data segments: none, a void is expected

```
<ID>|MDA|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Credits* or *Notification* type; for Credits exceptions, the error code must be either 0 or negative.

Example:

```
→       b00000147c9bc4c74|MDA|S|$|S|S8f3da29cfc463220T5454537|P|A|S|com.lightstream-
er.demo.ios.stocklistdemo|S|f780e9d8ffc86a5ec9a329e7745aa8fb3a1ecce77c09e202ec24cf-
f14a9906f1
←       b00000147c9bc4c74|MDA|V

→       c00000147cac69643|MDA|S|$|S|S9cb4758037a95c01T0439915|P|A|S|com.lightstream-
er.demo.ios.stocklistdemo|S|f780e9d8ffc86a5ec9a329e7745aa8fb3a1ecce77c09e202ec24cf-
f14a9906f1
←       c00000147cac69643|MDA|EC|banned+token
```

Notify MPN Subscription Activation

Edition note: Push Notifications is an optional feature, available depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `MSA`

Data segments:

- 1 *user name*, as string
- 1 *session ID*, as string

- 1 *table (i.e. subscription) info structure*, composed by:
  - 1 *window index*, as integer
  - 1 *publishing mode*, as 1-length Mode array
  - 1 *item group name (or item list specification)*, as string
  - 1 *data adapter name*, as string
  - 1 *field schema name (or field list specification)*, as string
  - 1 *index of the first item*, as integer
  - 1 *index of the last item*, as integer
  - 1 *number of items* as integer
  - M *item name*, as string, where M corresponds to the *number of items* specified above
- 1 *MPN subscription info structure*, composed by:
  - 1 *mobile platform type*
  - 1 *application ID*, as string
  - 1 *device token*, as string
  - 1 *trigger expression*, as string
  - 1 *notification format*, as string

The “notification format” field is a descriptor of the push notifications format of this subscription. The structure of the format descriptor depends on the platform type and it is represented in json.

```
<ID>|MSA|S|<user name>|S|<session ID>|I|<win. index>|M|<pub. Mode>|
        S|<item group>|S|<data adapter>|S|<field schema>|
        I|<first item idx.>|I|<last item idx.>|
        I|<number of items := M>|S|<item name 1>|...S|<item name M>|
        P|A|S|<application ID>|S|<device token>|S|<trigger>|
        S|<notification format>
```

Expected reply data segments: none, a void is expected

```
<ID>|MSA|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Credits* or *Notification* type; for Credits exceptions, the error code must be either 0 or negative.

Examples:

```
→       c00000147c9bc4c74|MSA|S|$|S|Sc4a1769b6bb83a4aT2852044|I|1|M|M|S|item4+item19|S|
QUOTES|S|stock_name+last_price+time|I|1|I|2|I|2|S|item4|S|item9|P|A|S|com.lightstream-
er.demo.ios.stocklistdemo|S|f74d8ffc5ee7cb31749a329a8f9202867c0a9906e80ee7f9eecc-
ca1f24c5aaf1|S|Double.parseDouble%28%24%7Blast_price%7D%29+%3E+1000.0|S|%7B%22aps
%22%3A%7B%22alert%22%3A%22%24%7Bmessage%7D%22%2C%22badge%22%3A%22AUTO%22%7D%2C
%22acme2%22%3A%5B%22%24%7Btag1%7D%22%2C%22%24%7Btag2%7D%22%5D%7D
←       c00000147c9bc4c74|MSA|V

→       c00000147cac69643|MSA|S|#|S|S401e2449d3b79feT1213883|I|1|M|M|S|item4+item19|S|
QUOTES|S|stock_name+last_price+time|I|1|I|2|I|2|S|item4|S|item9|P|G|S|com.lightstream-
er.demo.android.stocklistdemo|S|2082055669|S|Double.parseDouble%28%24%7Blast_price%7D
%29+%3E+1000.0|S|%7B%22priority%22%3A%22NORMAL%22%2C%22notification%22%3A%7B%22icon
%22%3A%22my_icon%22%2C%22body%22%3A%22my_body%22%2C%22title%22%3A%22my_title%22%7D%7D
```

```
←      c00000147cac69643|MSA|EC|too+many+subscriptions
```

Notify MPN Device Token Change

Edition note: Push Notifications is an optional feature, available depending on Edition and License Type. To know what features are enabled by your license, please see the License tab of the Monitoring Dashboard (by default, available at /dashboard).

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `MDC`

Data segments:

- 1 *user name*, as string
- 1 *session ID*, as string
- 1 *mobile platform type*
- 1 *application ID* (a.k.a. *package name*), as string
- 1 *device token* (a.k.a. *registration ID*), as string
- 1 *new device token*, as string

```
<ID>|MDC|S|<user name>|P|<platform type>|S|<application ID>|S|<device token>|
   S|<new device token>
```

Expected reply data segments: none, a void is expected

```
<ID>|MDC|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of Credits or Notification type; for Credits exceptions, the error code must be either 0 or negative.

Example:

```
→      3700000147c9bc4c74|MDC|S|$|S|<session ID>|P|A|S|com.lightstreamer.demo.ios.s-
tocklistdemo|S|f780e9d8ffc86a5ec9a329e7745aa8fb3a1ecce77c09e202ec24cff14a9906f1|S|
0849781a0afe0311f58bbfee1fcde031bfc56635c89c566dda3c6708fd893549
←      3700000147c9bc4c74|MDC|V
```

Failure

Direction: from counterpart to Proxy Adapter (backward request)
Expects reply: no
Method tag: `FAL`

Data segments:

- 1 *reason*, as generic exception

```
<ID>|FAL|E|<reason>
```

Note that the ID of this request is not actually relevant.

Examples:

```
←       3700000147c9bc4c74|FAL|E|Connection+lost
```

1.6 Data Provider Protocol Packets

Data Provider protocol aims at exposing the methods of the DataProvider class, of the Java In-Process Adapter SDK interface, through the protocol. To find the best trade-off between pros of interface remoting and cons of protocol overhead, some of the original Java interface methods have not been remotized, and others have been aggregated.
In particular, the “smart” version of the data exchange is not possible here and having different flavors of *update* is of no advantage. In addition, *setListener* is obviously not needed.

On the other hand, *isSnapshotAvailable* has not been ported and it is implemented by the Proxy Data Adapter to always return true. As a consequence, upon a successful subscription, the Remote Adapter is always due to send the item snapshot; if no snapshot information is available, an empty snapshot should be specified.

Another noteworthy difference is that the Proxy Data Adapter, to simplify its own logic, does not wait for the response to *Subscribe*; rather, it considers subscriptions always immediately successful; then, if the response is an exception, it just logs the fact at ERROR level. In practice, returning an exception from *Subscribe* doesn’t invalidate the subscription at LS Server level. As a consequence, the final *Unsubscribe* will still always be issued.
Another consequence is that, if, despite the subscription failure, any updates or other notifications are sent for the subscription, they will reach the client as usual.
Likewise, the Proxy Data Adapter does not wait for the response to *Unsubscribe*, but considers the unsubscriptions always immediately successful. This means that some subscription-related notifications sent just before sending the response to *Unsubscribe* may be considered already late and discarded.

For more explanations regarding the semantics of each information, please refer to the interface API documentation of the Java In-Process Adapter SDK.

The Data Provider protocol involves requests, replies, and notifications, whereas it does not include backward requests and corresponding replies. We will refer to its channels as the **Data requests channel** and the **Data replies channel**.
In practice, the Proxy Data Adapter and the Remote Data Adapter communicate through a TCP connection carrying the Data requests channel and the Data replies channel as the two streams.

However, in order to support old Remote Data Adapters, the Proxy Data Adapter can communicate also through two separate TCP connections. In fact, for Server versions earlier than 7.4, this was the only way available for the communication.
For this reason, for convenience, we will also refer to distinct channels for replies and notifications, namely the **Data pure-replies channel** and the **Data notifications channel**, with obvious meaning, where the Data replies channel will be the union of the two.
In fact, in the two-connections communication, one connection carries the Data requests channel and the Data pure-replies channel as the two streams, and the other one carries the Data notifications channel as the only stream (that is, as a one-way connection).

Data Init

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `DPI`

Data segments:

- N *initialization parameter name-value pairs*, both names and values as string; all strings should be encoded with the Backward-Compatibility Encoding.

  The parameters are supplied by the Proxy Adapter based on its own configuration in an unspecified order.
  A parameter named “ARI.version” will always be supplied, to carry the maximum protocol version supported by the Proxy Adapter.
  A parameter named “keepalive_hint.millis”, if present, specifies the inactivity time after which a keepalive packet should be issued on the replies channel to prevent the Proxy Adapter from closing the connection for inactivity. However, with the old two-connections communication, this applies independently to the pure-replies channel and the notifications channel. Note that the value is a number, but encoded as a string.

```
<ID>|DPI|S|<param 1>|S|<param 1 value>|...|S|ARI.version|S|<version>|...|S|
<param N>|S|<param N value>
```

Expected reply data segments:

- N *initialization parameter name-value pairs*, both names and values as string; all strings will be encoded with the Backward-Compatibility Encoding.

  The supplied parameters are handled by the Proxy Adapter based on its own configuration and regardless of their order.
  A parameter named “ARI.version” is mandatory, to carry the protocol version that the Remote Data Adapter is willing to use. Based on it, the Proxy Adapter will either proceed or interrupt the connection. If it is lower than the request’s ARI.version, the Proxy Adapter may still support it and proceed. If it is higher, the Proxy Adapter will interrupt (the Proxy Adapter will recognize the response, regardless of its protocol version); in this case, returning a Generic Exception (see below) is also an option.

```
<ID>|DPI|S|<param 1>|S|<param 1 value>|...|S|ARI.version|S|<version>|...|S|
<param N>|S|<param N value>
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic* or *Data* type (based on the Backward-Compatibility Encoding for strings). This will cause the Proxy Adapter to interrupt the connection. The Proxy Adapter will recognize such response, regardless of its protocol version.

Examples:

```
→      10000010c3e4d0462|DPI|S|ARI.version|S|1.8.2|S|keepalive_hint.millis|S|8000|S|
adapters_conf.id|S|DEMO|S|data_provider.name|S|STOCKLIST|...
←      10000010c3e4d0462|DPI|S|ARI.version|S|1.8.2

→      10000010c3e4d0462|DPI|S|ARI.version|S|1.8.2|S|adapters_conf.id|S|DEMO|S|
data_provider.name|S|STOCKLIST|...
←      10000010c3e4d0462|DPI|S|ARI.version|S|1.8.2|S|user|S|remote1|S|password|S|fd-
hjkslghak

→      20000010c3e4d0462|DPI|S|ARI.version|S|1.8.2|S|custom:feed_name|S|feed1|S|cus-
tom:feed_port|S|8080|...
←      20000010c3e4d0462|DPI|ED|Data+Feed+unavailable
```

Subscribe

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `SUB`

Data segments:

- 1 *item name*, as string

```
<ID>|SUB|S|<item name>
```

Expected reply data segments: none, a void is expected

```
<ID>|SUB|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Subscription* or *Failure* type.

Examples:

```
→      10000010c3e4d0462|SUB|S|aapl
←      10000010c3e4d0462|SUB|V

→      20000010c3e4d0462|SUB|S|xyzy
←      20000010c3e4d0462|SUB|EU|Unknown+item
```

Unsubscribe

Direction: from Proxy Adapter to counterpart
Expects reply: yes
Method tag: `USB`

Data segments:

- 1 *item name*, as string

```
<ID>|USB|S|<item name>
```

Expected reply data segments: none, a void is expected

```
<ID>|USB|V
```

Alternatively, the reply can contain a single data segment consisting of an exception of *Generic*, *Subscription* or *Failure* type.

Examples:

```
→       30000010c3e4d0462|USB|S|aapl
←       30000010c3e4d0462|USB|V

→       40000010c3e4d0462|USB|S|xyzy
←       40000010c3e4d0462|USB|EU|Item+not+subscribed
```

End Of Snapshot

Direction: from counterpart to Proxy Adapter (notification)
Expects reply: no
Method tag: `EOS`

Data segments:

- 1 *item name*, as string
- 1 *unique ID of the originating subscription request*, as string

```
<timestamp>|EOS|S|<item name>|S|<ID>
```

Examples:

```
←       1152096504423|EOS|S|aapl|S|10000010c3e4d0462
```

Update By Map

Direction: from counterpart to Proxy Adapter (notification)
Expects reply: no
Method tag: `UD3`

Data segments:

- 1 *item name*, as string
- 1 *unique ID of the originating subscription request*, as string
- 1 *is snapshot flag*, as boolean
- N *field-value pairs*, composed by:
  - 1 *field name*, as string
  - 1 *field value*, as string or byte array

```
<timestamp>|UD3|S|<item name>|S|<ID>|B|<is snapshot>|S|<field 1>|S|<value
1>|  ...  |S|<field N>|S|<value N>

<timestamp>|UD3|S|<item name>|S|<ID>|B|<is snapshot>|S|<field 1>|Y|<value
1>|  ...  |S|<field N>|Y|<value N>
```

Note: the special mandatory fields for items to be requested in COMMAND mode, named “key” and “command”, must be encoded as string.

Examples:

```
←      1152096504423|UD3|S|aapl|S|10000010c3e4d0462|B|1|S|pct_change|S|0.44|
S|last_price|S|6.82|S|time|S|12%3a48%3a24

←      1152096504423|UD3|S|aapl|S|10000010c3e4d0462|B|1|S|pct_change|Y|MC40NA==|
S|last_price|Y|Ni44Mg==|S|time|Y|MTI6NDg6MjQ=
```

Declare Field Diff Order

Direction: from counterpart to Proxy Adapter (notification)
Expects reply: no
Method tag: `DFD`

Data segments:

- 1 *item name*, as string
- 1 *unique ID of the originating subscription request*, as string
- N *field-value pairs*, composed by:
  - 1 *field name*, as string
  - 1 *algorithm list*, as Diff Algorithm array

```
<timestamp>|DFD|S|<item name>|S|<ID>|S|<field 1>|F|<algorithm list 1>|  ...  |
S|<field N>|F|<algorithm list N>
```

Note that if a null value is passed as a Diff Algorithm array, the entry will be just ignored. On the other hand, an empty array declares that the Field supports no algorithm and the Server default behavior doesn’t apply.

Examples:

```
←      1152096504423|DFD|S|aapl|S|10000010c3e4d0462|S|message|F|MJ|
S|timestamp|A|$
```

Clear Snapshot

Direction: from counterpart to Proxy Adapter (notification)
Expects reply: no
Method tag: `CLS`

Data segments:

- 1 *item name*, as string
- 1 *unique ID of the originating subscription request*, as string

```
<timestamp>|CLS|S|<item name>|S|<ID>
```

Examples:

```
←      1152096504423|CLS|S|aapl|S|10000010c3e4d0462
```

Failure

Direction: from counterpart to Proxy Adapter (notification)
Expects reply: no
Method tag: `FAL`

Data segments:

- 1 *reason*, as generic exception

```
<timestamp>|FAL|E|<reason>
```

Examples:

```
←       1152096504423|FAL|E|Connection+lost
```

1.7 Common Packets

The following packets are not part of the communication with the Adapters, but rather connection-related and may pertain to multiple channels.

Remote Adapter Credentials

Direction: from counterpart to Proxy Adapter
Channels:

- Metadata Adapter replies channel
- Data Adapter replies channel

Method tag: `RAC`

This special packet is sent on replies channels, but it does not take the form of a backward requests, but rather of an unsolicited reply, where the implicit request ID must be set as 1.

For Server version 7.1 and above, the Proxy Adapter will always recognize the packet. However, the packet contents will be understood and considered depending on the Server version.

If Remote Adapter authentication is configured on the Proxy Adapter for a Remote Adapter, the packet is mandatory and should be the first packet sent on all pertaining channels. If the credentials are refused, the Proxy Adapter will ignore any other packet and will interrupt the connection.
Otherwise, the packet is optional and can be sent to provide the Proxy Adapter with other connection-related information.

With the old two-connections communication with Remote Data Adapters, this packet is handled differently, and pertains to both the Data pure-replies channel and the Data notifications channel. Moreover, when sent on a notifications channel, it takes the form of a normal notification.

Otherwise, the packet should not be sent on the Data notifications channel. However, sending it is tolerated and the packet is ignored. This allows for a “transitional” Remote Data Adapter implementation, which keeps handling the pure-replies and notifications channels separately, as needed for the old two-connections communication, then just merges the packet flows in a single connection.

Data segments:

- N *initialization parameter name-value pairs*, both names and values as string; all strings will be encoded with the Backward-Compatibility Encoding.

  The supplied parameters are handled by the Proxy Adapter regardless of their order.
  - For Server version 7.1 and above, parameters with names “user” and “password” must be provided to specify the credentials, when needed.
  - For Server version 7.2 and above, a parameter with name “enableClosePacket” and value “true” on a replies channel will request the Proxy Adapter to (try to) issue a `CLOSE` packet on the corresponding requests channel in case of connection interruption for any reason. Then, after a successful response to the `MPI/DPI` packet, the availability of the `CLOSE` packet will depend only on the chosen protocol version.
  - For Server version 7.4 and above, a parameter with name “SDK” is detected and logged by the Server at INFO level. If leveraged, the recommended value is “Generic Adapter SDK”, which may help log-based diagnostics.

```
As a reply:
1|RAC|S|<param 1>|S|<param 1 value>|...|S|<param N>|S|<param N value>

As a notification:
<timestamp>|RAC|S|<param 1>|S|<param 1 value>|...|S|<param N>|S|<param N value>
```

Examples:

```
        As a reply:
←       1|RAC|S|user|S|remote1|S|password|S|fdhjkslghak|S|enableClosePacket|S|true

        As a notification:
←       <timestamp>|RAC|S|user|S|remote1|S|password|S|fdhjkslghak
```

Keepalive

Direction: from counterpart to Proxy Adapter
Channels:

- Metadata Adapter replies channel
- Data Adapter replies channel

Method tag: `KEEPALIVE`

The packet has a special syntax, independent from the channel, as specified in the introductory notes.

With the old two-connections communication with Remote Data Adapters, this packet is handled differently, and pertains to both the Data pure-replies channel and the Data notifications channel, where it should be sent based on the own activity of each channel.

Otherwise, obviously, the keepalives should be sent based on the overall activity of the replies channel. However, if a “transitional” Remote Data Adapter implementation keeps handling the pure-replies and notifications channels separately, as needed for the old two-connections communication, and just merges the packet flows in a single connection, there is no issue: it will just send more keepalive packets than needed, but outages will be detected anyway.

Examples:

```
←       KEEPALIVE
```

Close Notification

Direction: from Proxy Adapter to counterpart
Channels:

- Metadata Adapter requests channel
- Data Adapter requests channel

Method tag: `CLOSE`

The packet takes the form of a request with no possible reply, where the implicit request ID is always set as 0.

Data segments:

- N *closure parameter name-value pairs*, both names and values as string; all strings should be encoded with the Backward-Compatibility Encoding.

  The parameters are supplied by the Proxy Adapter in an unspecified order.
  A parameter named “reason” will always be supplied, to carry a text message describing the reason for an ongoing connection interruption.
  The syntax is predisposed for the introduction of other parameters in future versions of the protocol.

```
0|CLOSE|S|<param 1>|S|<param 1 value>|...|S|reason|S|<close reason>|...|S|
<param N>|S|<param N value>
```

No reply is expected.

Note: this packet is sent by the Proxy Adapter upon an ordered closure of the connection. Obviously, an abrupt closure of the connection, without this packet, is always possible.

Examples:

```
→       0|CLOSE|S|reason|S|keepalive+timeout
```

1.8 Notes on Protocol Method Sequence

Metadata Init, Data Init

The proper *Init* method is the very first request sent from the Proxy Adapter to the counterpart. In this way, the Proxy Adapter can send initialization parameters to the Remote Adapter

(based on its own configuration) before starting to issue requests. The Proxy Adapter won't send any request until the *Init* method sends back a successful response.

Remote Adapter Credentials

The *Remote Adapter Credentials* packet is optional. However, the Proxy Adapter, depending on its current configuration, may request this packet, to receive suitable identification information, before accepting any other packet on the channel and before sending the proper *Init* method on requests channels.
In any case, the packet is valid only if it is the first one sent on the channel.

Notify User

The *Notify User* method is invoked before any other requests for the same user (which version is used depends on the configuration of <use_client_auth>; by default, the base version is used). This means that the Metadata Adapter has always a chance to authenticate users before any detail about their profile is requested.

Notify User, **Notify New Session**

All the authorization request management is expected to depend on the user name only. Anyway, some information on the specific client request instance are supplied to the *Notify New Session* method, as the *client context*. The *REQUEST_ID* property is the same id that has just been supplied to *Notify User* for the same client request instance; this allows for using local authentication-related details for the authorization task.
Note: the Remote Metadata Adapter is responsible for disposing any cached information in case Notify New Session is not issued because of any early error during request management.

Notify New Tables, **Notify Tables Close**

These methods are requested by the Proxy Metadata Adapter only when the Remote Metadata Adapter asks for them through the *wants table (i.e. subscription) notifications flag* returned with the *Notify User* method, on a user basis. If this flag is always returned as false, then these calls are never received.

Notify New Tables, **Notify New Session**, **Get Items, Get Schema, Get Item Data**, **Get User Item Data**

These methods are requested by the Proxy Metadata Adapter synchronously (i.e.: the requesting thread waits for the reply; however, the Proxy Metadata Adapter may still issue multiple requests in parallel, as stated in the general notes). Moreover, the requesting threads are taken from a limited pool.
This means that these requests should be processed as fast as possible; but, anyway, any roundtrip delay related to the remote call will keep the Server waiting.
In order to avoid that delays on one session propagate to other sessions, the size of the thread pool devoted to the management of the client requests should be properly set, through the "server_pool_max_size" flag, in the Server configuration file.

Alternatively, a dedicated pool, properly sized, can be defined for the involved Adapter Set in “adapters.xml”. An even more restricted dedicated pools can be defined for each Data Adapter in the Adapter Set. The latter pool would also run any Metadata Adapter method related to the items supplied by the specified Data Adapter.

Notify User, Notify User Message

These methods are requested by the Proxy Metadata Adapter asynchronously, hence there are no threads waiting for the reply; however, there are still resources blocked within the Server, hence the number of pending requests can be subject to a limit.
This means that these requests should still be processed as fast as possible, although the delays on one session should not directly propagate to other sessions.
The set of pending invocations is handled by a special type of “thread pool”, named in this way to keep a uniform view across all methods. Precisely, there are two dedicated pools, the authentication pool and the messages pool, for each Adapter Set, which can be configured in “adapters.xml”.

Notify New Tables, Notify Tables Close

If desired, invocations of these methods, when related to the same session, can be performed by the Server sequentially. This can be achieved by leveraging the <sequentialize_table_notifications> flag available in the Proxy Metadata Adapter configuration. Obviously, this would mean that any delay or roundtrip time involved may keep further subscription requests for the same session blocked. Obviously, when the two methods refer to the same subscription, they are always invoked sequentially.

Notify Tables Close, **Notify Session Close**, **Subscribe**, **Unsubscribe**

These methods are requested by the Proxy Metadata Adapter side asynchronously (i.e.: without waiting for the reply). This does not mean that the reply should not be sent: the reply is mandatory (or a timeout exception will be raised on the Proxy Metadata Adapter side), but any exception that it should carry would simply be logged. Anyway, this means that these methods don't need to be non-blocking, as long as they don't last more than the configured timeout limit.

Notify New Session, **Notify Session Close**

These methods are invoked consistently. *Notify New Session* always precedes other methods related with the same session and *Notify Session Close* always follows other methods related with the same session.
To be more precise, after *Notify Session Close*, no more calls to *Notify New Tables* and *Notify Tables Close* for this session ID are possible. On the other hand, trailing invocations of methods related with the validation of client requests, like *Get Items*, are still possible and accepting them would have no effect. However, if the method may have side-effects on the Adapter, like *Notify User Message*, the Adapter is responsible for checking if the session is still valid.

Subscribe, **Unsubscribe**

These methods are invoked consistently. *Subscribe* always precedes *Unsubscribe* for the same item; after *Unsubscribe*, a further *Subscribe* is possible, and so on. However, since the Proxy Data Adapter doesn't wait for the answer to Subscribe or Unsubscribe before issuing the next invocation, if needed, then it is possible that multiple *Subscribe*-*Unsubscribe* requests are pending at the same time.

Subscribe, **End Of Snapshot, Update By Map**

Upon a successful *Subscribe*, the Remote Data Adapter is always due to provide the Proxy Data Adapter with snapshot information. This is done by sending *Update By Map* notifies with the "is snapshot" flag set to true and a terminating *End Of Snapshot* notification. This should be done as soon as possible and must precede any real-time updates carried by *Update By Map*. In case no snapshot information is available, an empty snapshot should be sent.

Subscribe, **End Of Snapshot, Update By Map**

Since the Proxy Data Adapter doesn't wait for the answer to *Subscribe*, it is not mandatory that the *End Of Snapshot* notification and even *Update By Map* and other notifications related with a successful subscription are sent after the reply to *Subscribe*. They may even precede an unsuccessful reply to *Subscribe*, as the subscription issue will just be logged by the Proxy Adapter but will not impact on the data flow.
By the way, this lack of an ordering constraint allows for a "transitional" Remote Data Adapter implementation, which keeps handling the pure-replies and notifications channels separately, as needed for the old two-connections communication, then just merges the packet flows in a single connection.

Declare Field Diff Order, Update By Map

The *Declare Field Diff Order* messages can be sent at any point in which *Update By Map* can be sent and can be sent multiple times, although usually it should be sent (if at all needed) only once, before the first use of *Update By Map*.
Hence the "diff" order of a field can be specified multiple times, but it cannot be changed during the life of the subscription. Note that if a field is used in *Update By Map* and it was never included before in *Declare Field Diff Order,* this implicitly accepts the Server default behavior, which also cannot be changed for the remaining life of the subscription.

End Of Snapshot, Update By Map, Clear Snapshot

As said above, the optional initial snapshot and the terminating *End Of Snapshot* notification should be sent first thing. Actually, the *End Of Snapshot* notification can be omitted and in that case the subsequent *Update By Map* with the "is snapshot" flag set to false will imply snapshot termination. But note that this would be equivalent only if the *Update By Map* came immediately; otherwise, the Server would, pointlessly, keep waiting for more snapshot to come.
Similarly, a *Clear Snapshot* notification should only occur after snapshot termination; otherwise, it would imply snapshot termination as well.

Keepalive

Keepalive packets can be sent on any channel at any moment. The Proxy Adapter may be configured to close the connections after an inactivity period. Sending a *Keepalive* packet when no other activity is in place will prevent this.

Close Notification

A *Close Notification* packet can be sent by the Proxy Adapter as the last one before connection interruption. The presence of this packet adds no guarantee that the latest responses sent by the Remote Adapter have been processed by the Proxy Adapter.
The *Close Notification* packet is enabled on the Proxy Adapter after receiving the response to the *Init* packet, which establishes the protocol version. Before that, it can be enabled through the *Remote Adapter Credentials* packet.