



## **Mobile Content Delivery Protocol**

Adobe Systems Incorporated  
601 Townsend Street  
San Francisco, CA  
94103  
415-252-2000

Last Updated: 9/9/2008

Copyright © 2008 Adobe Systems Incorporated. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without written approval from Adobe Systems Incorporated. Notwithstanding the foregoing, a person obtaining an electronic version of this manual from Adobe may print out one copy of this manual provided that no part of this manual may be printed out, reproduced, distributed, resold, or transmitted for any other purposes, including, without limitation, commercial purposes, such as selling copies of this documentation or providing paid-for support services.

### **Trademarks**

Adobe, ActionScript, Flash and Flash Cast are either registered trademarks or trademarks of Adobe Systems Incorporated and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words, or phrases mentioned within this publication may be trademarks, service marks, or trade names of Adobe Systems Incorporated or other entities and may be registered in certain jurisdictions including internationally.

### **Third-Party Information**

This guide contains links to third-party websites that are not under the control of Adobe Systems Incorporated, and Adobe Systems Incorporated is not responsible for the content on any linked site. If you access a third-party website mentioned in this guide, then you do so at your own risk. Adobe Systems Incorporated provides these links only as a convenience, and the inclusion of the link does not imply that Adobe Systems Incorporated endorses or accepts any responsibility for the content on those third-party sites. No right, license or interest is granted in any third party technology referenced in this guide.

**NOTICE: THIS PUBLICATION AND THE INFORMATION HEREIN IS FURNISHED “AS IS”, IS SUBJECT TO CHANGE WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY ADOBE SYSTEMS INCORPORATED. ADOBE SYSTEMS INCORPORATED ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES, MAKES NO WARRANTY OF ANY KIND (EXPRESS, IMPLIED, OR STATUTORY) WITH RESPECT TO THIS PUBLICATION, AND EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR PARTICULAR PURPOSES, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS.**

Adobe Systems Incorporated

Published September 2008.

# Contents

1	Scope.....	7
2	References.....	7
3	Definitions.....	7
4	Terminology.....	8
5	Introduction.....	9
6	Mobile Content Deliver Protocol overview.....	9
6.1	Protocol HTTP commands.....	9
6.1.1	HTTP version.....	10
6.1.2	Message payloads.....	10
6.1.3	HTTP headers.....	10
6.1.4	HTTP cookies.....	10
6.1.5	Content Types.....	10
6.2	Feed item collections.....	11
6.2.1	Feed item encoding.....	12
6.3	Full and delta channel updates.....	12
6.3.1	Full updates.....	12
6.3.2	Delta updates.....	13
6.3.3	Sync levels.....	13
6.3.4	Filters.....	13
6.3.5	Feed update process.....	14
6.4	About channels.....	14
6.4.1	Standard channels.....	15
6.4.2	System channels.....	16

6.5	Channel subscription model.....	17
6.5.1	Subscription states.....	17
6.5.2	Subscription modifications.....	18
6.5.3	Channel download and confirmation.....	18
6.5.4	Subscription targeting.....	19
6.6	About channel namespaces and content types .....	19
6.7	Device capabilities and capability negotiation .....	20
6.8	Server initiated communications .....	20
6.9	Channel permissions.....	20
6.10	Error handling.....	22
6.10.1	Transport layer .....	23
6.10.2	Request layer errors.....	23
6.10.3	Feed layer errors .....	24
6.10.4	Client errors .....	25
6.10.5	Error message localization .....	25
7	Protocol commands overview .....	25
7.1	Login command .....	26
7.1.1	Login request message .....	26
7.1.2	Login response message .....	27
7.1.3	Session ID options.....	28
7.1.4	Device capabilities string.....	28
7.1.5	Authentication options .....	29
7.2	Logout.....	29
7.2.1	Logout request message .....	29

7.2.2	Logout response message .....	29
7.3	GetData command .....	30
7.3.1	Request message.....	30
7.3.2	Response message.....	31
7.4	SetData.....	32
7.4.1	Request message.....	32
7.4.2	Response message.....	33
7.4.3	Operations performed with SetData .....	34
7.4.4	Subscription changes .....	34
7.4.5	Filter changes.....	38
7.4.6	Settings data changes .....	38
7.5	Server-initiated communications.....	38
8	Binary transport format .....	39
8.1	Data types .....	40
8.2	Record header format.....	41
8.3	BeginFeed record format .....	41
8.4	ItemData record format .....	41
8.5	PropData record format.....	42
8.6	Sample update message .....	42
9	Use cases and scenario diagrams.....	44
9.1	Initial login .....	44
9.2	Login successful, no session.....	46
9.3	Login successful, capabilities change .....	48
9.4	Login failure, fail (generic).....	50

9.5	Login failure, fail-auth.....	51
9.6	Login failure, fail-retry .....	52
9.7	Server-initiated subscription add.....	53
9.8	Client-initiated new subscription.....	54
9.9	Server-initiated subscription add, client declines download.....	55
10	Appendix A: Feed item schemas.....	56
10.1	Subscription feed schema.....	57
10.1.1	Channel information (feedapp-info) items .....	57
10.1.2	Channel subscription (feedapp-sub) items.....	59
10.1.3	Subscription download (feedapp-download) items .....	60
10.1.4	Channel preview (feedapp-preview) items.....	62
10.2	Catalog feed schema .....	62
10.2.1	Category item (category) schema .....	63
10.2.2	Channel item schema .....	63
10.3	Filter items (feedapp-pref-select).....	63
10.4	Settings items (feedapp-pref-setting) .....	64
10.5	Root content items (feedapp-bininfo).....	65
10.6	Localized resource items (feedapp-localeinfo) .....	65
10.7	Feed error items (fc-result-info) .....	66
11	Appendix B: Client error codes .....	66
12	Appendix C: Channel permission bits .....	71
13	Appendix D: Device profile classes and capabilities .....	73

# 1 Scope

This document describes the Mobile Content Deliver Protocol, which provides a mechanism for a server to deliver applications and data to mobile clients. The document covers communication and semantics, session and transaction handling, encoding formats, status and error codes, and protocol scenarios and use cases.

## 2 References

The following reference material may be helpful in understanding this document.

- “Hypertext Transfer Protocol--HTTP/1.1”(<http://www.ietf.org/rfc/rfc2616.txt>)
- “Form content types” (Section 17.13.4.2) of the [XHTML 4.0.1 specification](#).
- “Codes for the Representation of Names of Languages” ([http://www.loc.gov/standards/iso639-2/php/code\\_list.php](http://www.loc.gov/standards/iso639-2/php/code_list.php))
- “English country names and code elements” ([http://www.iso.org/iso/country\\_codes/iso\\_3166\\_code\\_lists/english\\_country\\_names\\_and\\_code\\_elements.htm](http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm))

## 3 Definitions

**Client:** An application hosted by the mobile handset OS that implements the client-facing parts of the Mobile Content Delivery Protocol.

**Server:** A server application that implements the server-side part of the Mobile Content Delivery Protocol.

**Feed item collection:** The primary data structure used within the protocol, consisting of typed data objects called *feed items*. Feed item collections are used to describe and transport each channel’s data set, as well as to express and transport channel-specific commands between a client and server.

**Channel:** A user-facing application and its associated data set, hosted by the client. This is a mini application that renders the data set and interacts with the user’s input. Typically the data set and the code to render the data are persistently cached on the client, providing an instantaneous startup experience of each channel.

**Subscription:** The relationship between a user and a channel. Each subscription is managed on the server and contains subscription related meta-data such as duration, price and other terms.

**System channel:** A channel that provides users with a means to subscribe to a new channel, or discover new channel content. System and non-system channels are different only in the types of data they consume and the permissions they have within the host client. System channels may not have a user interface, in which case they are controlled by a set of APIs accessible by other channels.

**Filter:** Models a restrictive query on a channel’s feed item collection by stating the item subset of interest; it indicates a user’s preference for specific data points (for example, weather data for a specific region).

**Settings:** Persistent name/value pairs that can be stored within a channel and then later retrieved. Settings are stored on both the client and the server, to guarantee they can be retrieved even in the event of a hard reset of the phone.

**Preference:** The filters and settings for a given channel make up the *preference* for that channel.

**Channel data state:** A channel's feed item collection, which contains the channel's data, filters, settings and any application binaries and resources.

**Channel feeds:** Feed item collections used by non-system channels.

**System feeds:** Feed item collections that are used by system channels and contain system or user-specific information, such as the list of available content categories, or a user's current list of subscribed applications.

**Root content:** Representation within the protocol of a channel's primary user-facing application (typically a SWF file). This content is the main entry point for rendering the channel's data set.

**Sync level:** A monotonically increasing value associated with feed items and feed item collections that allow a server to deliver differential, or *delta*, data updates to clients.

**Binary transport format:** A binary encoding format used to serialize feed item collections prior to transport over the network.

**Device profile classes:** An aggregation of device capabilities, such as display characteristics, or supported audio formats.

**Device profile:** An aggregation of device profile classes managed by the server.

**Device capabilities string:** A specially formatted string the client sends to the server that contains the device's capabilities, organized into device profile classes.

## 4 Terminology

This specification uses key words based on [RFC2119](#) to indicate requirement level. In particular, the following words are used to describe the actions of an application or library that implements the client protocol.

**May:** The word *may*, or the adjective *optional*, mean that conforming protocol implementations are permitted, but need not behave as described.

**Should:** The word *should*, or the adjective *recommended*, mean that there could be reasons for a protocol implementation to deviate from the behavior described, but that such deviation could hurt interoperability and should therefore be advertised with appropriate notice.

**Must:** The word *must*, or the term *required* or *shall*, mean that the behavior described is an absolute requirement of the specification.

## 5 Introduction

The Mobile Content Delivery Protocol provides a means for delivering applications to mobile devices, called *channels*, and the synchronized, channel-specific data sets on which they operate, called *feeds*. The protocol includes a control-flow mechanism for managing channel subscriptions, and application preferences.

The protocol could be used by anyone wishing to serve applications and content to the Adobe Mobile Client, or someone who wishes to develop a custom client that can interact with the Adobe Mobile Server.

The protocol is bi-directional, with a client-initiated “pull” component operating over HTTP or HTTPS, and a server-initiated “push” component operating over SMS. The protocol is designed for lightweight content updates personalized according to end user preferences. It is not designed to handle applications that require low network latency, such as instance messaging, or real-time data updates, such as a multiplayer game.

## 6 Mobile Content Deliver Protocol overview

The basic features of the Mobile Content Delivery Protocol are as follows:

- **HTTP-based.** All messages transmitted from the client to the server are sent using HTTP POST requests; all server responses are contained in the associated HTTP responses. (The one exception to this rule is a feature that allows a server to send an SMS message to a client requesting it to initiate a new HTTP request.)
- **Differential data updates.** The protocol provides a means for a server to only deliver channel data that has been updated since the last client update.
- **Personalized content.** The protocol provides a means for a client to specify, on a per-user and per-channel basis, the data subset that it would like the server to deliver.
- **Subscription control mechanism.** The protocol provides a control flow mechanism for a client and server to manage a user’s subscriptions.
- **Content discovery.** The protocol provides a way for a server to regularly deliver an updated content catalog to clients.
- **Session-based.** Upon each login a client establishes a session ID with the server that it sends in every HTTP request.

### 6.1 Protocol HTTP commands

The protocol consists of four client commands, namely:

- **Login**—Establishes a new client session with a server.
- **Logout**—Terminates a client session with a server.
- **GetData**—Instructs the server to synchronize one or more feeds with the client.
- **SetData**—Transmits feed item collection updates to the server.

Each command is associated with an endpoint URL on an HTTP server—for example ) and defines a set of message parameters. Depending on the specific command, a message’s parameters may either be URL-encoded

name/value pairs, or a collection of feed items serialized in the binary transport format (see Binary transport format).

### 6.1.1 HTTP version

The client-server protocol MUST operate using the standard features of HTTP 1.0, though HTTP 1.1 SHOULD be supported and is preferred, provided the mobile client can support it. However, the more advanced features of HTTP 1.1 (such as keep-alive or chunked-encoding) are not required by the protocol.

### 6.1.2 Message payloads

All messages transmitted from the client to the server are sent by HTTP POST requests. All messages transmitted from the server to the client are contained in the related HTTP responses. In both cases, feed item collections (encoded in a binary format) are used to transfer content and express commands in both directions. Expressing commands as state –items in a feed item collection–allows for a simple extension mechanism between the client and the server. However, in order to guarantee maximum interoperability an implementation SHOULD only rely on the commands described in this specification.

### 6.1.3 HTTP headers

Server and client must support the standard `Content-Length`, `Content-Type`, and `Pragma` headers.

The server MUST use a custom `Pragma` directive named `flashcast-fetch-time`, which specifies the number of seconds the client SHOULD wait before re-synchronizing. A value of 0 signifies that the client should resynchronize immediately. If the server wished the client to synchronize its content again in one hour, the header would look like the following:

```
Pragma: flashcast-fetch-time=3600
```

In addition to these standard HTTP headers, the protocol uses a custom header called `flashcast-capabilities`. This header is used only in the client's Login request message. It consists of a URL-encoded string that describes the features and capabilities of the device. The server SHOULD use this information to deliver content appropriate for the device. For details about device capabilities, see Device capabilities and capability negotiation.

### 6.1.4 HTTP cookies

The use of HTTP cookies is required for session management, as described below. The client must be able to detect `Set-Cookie` headers in the HTTP response messages including multiple cookies per header, and return the same values in subsequent request messages during that session in the corresponding `Cookie` HTTP header. Refer to RFC 2616 "[Hypertext Transfer Protocol -- HTTP/1.1](#)" for details on cookie handling.

### 6.1.5 Content Types

The protocol uses two content types for transporting messages.

**URL-encoded**—A standard encoding format used to transport messages containing simple name-value pairs, such as the device’s capabilities or authentication credentials. This format is specified in the `Content-Type` HTTP header as `application/x-www-form-urlencoded`. For information about this format, URL encoding, see [Section 17.13.4.2](#) (“Form content types”) of the [XHTML 4.0.1 specification](#). The document titled “Uniform Resource Identifier (URI): Generic Syntax” located at <http://www.ietf.org/rfc/rfc2616.txt>.

**Binary transport format**—A format used to serialize feed item collections for transport over the network. This format is specified by the `Content-Type` `application/x-flashcast-data-update`. For details on this format, see [Binary transport format](#).

## 6.2 Feed item collections

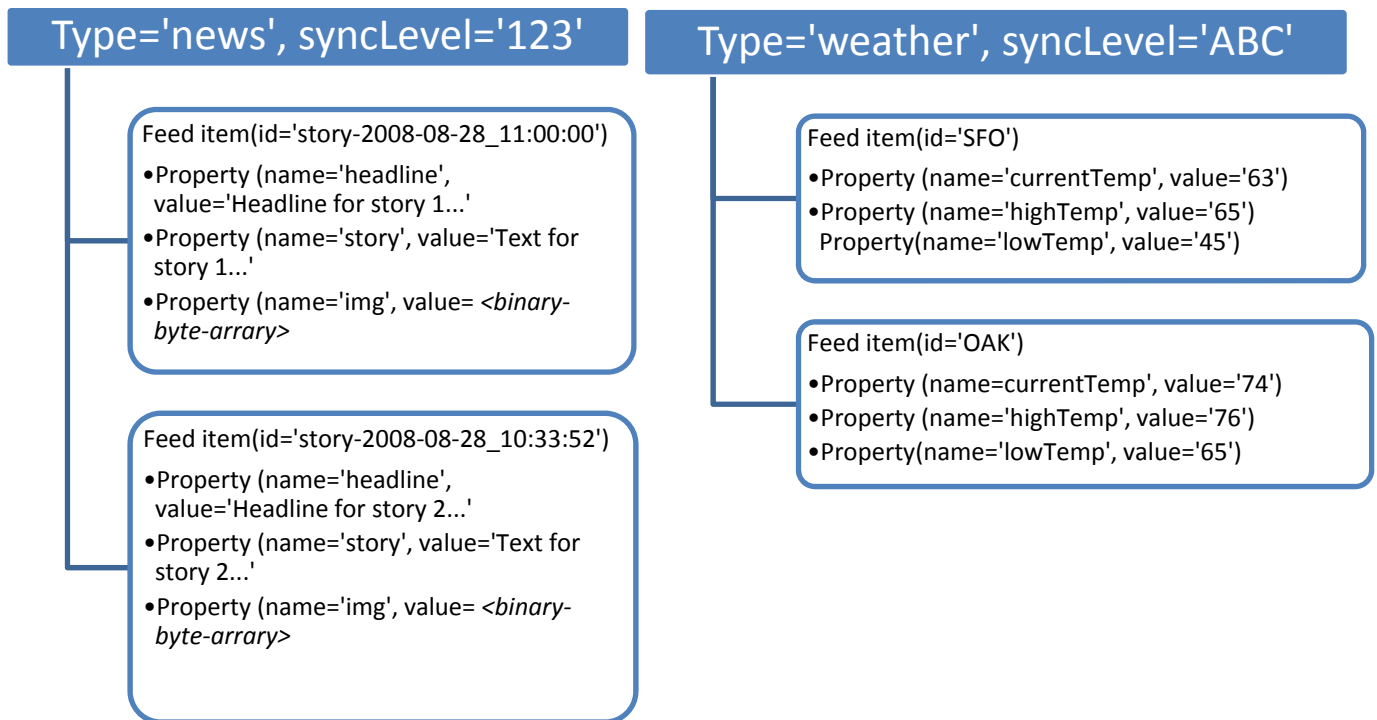
The bulk of the interaction between a client and server deals with the transfer and synchronization of *feed item collections*, one per each channel that a client is subscribed to. Everything needed for channel operation—application, resources and data—is modeled as part of each channel’s feed item collection.

A feed item is a collection of named properties; a property’s value may be a simple string (“Hello world”, for example), or a binary byte array accompanied by a MIME type indicating its format. Each feed item is associated with a type (a short string) and an ID (also a string), which is unique among other feed items of the same type.

A feed item collection is analogous to a table in a relational database. Feed items of the same type are like rows in a database table, each uniquely identified by its feed item ID. Feed item property names are like column names in the table.

In addition, each feed item collection has an associated synchronization level, or *sync level*, that the server uses to calculate differential updates (see [Full and delta updates](#)).

The following diagram illustrates a feed item collection for a channel that displays news and weather information. The channel’s feed item collection contains two feed item types: `news` and `weather`. Feed items of type `news` contain two text properties named `headline` and `title`, and a binary-asset property named `img`. The feed items of type `weather` are all string properties named `currentTemp`, `highTemp`, and `lowTemp`.



### 6.2.1 Feed item encoding

Prior to transport over the network, a client or server MUST serialize feed items in a binary message format known as the binary transport format. On the receiving end of the transmission, the client or server MUST de-serialize the contents of the binary message into local data structures.

For more information about this format, see Binary transport format.

## 6.3 Full and delta channel updates

The protocol allows for clients to request, or servers to deliver, a *full update* or a *delta update* to the feed item collection for each of the client's subscribed channels. Whether a feed item collection is a full or delta update is indicated by the `delta` field in the feed item collection delivered to the client.

### 6.3.1 Full updates

A full update for given channel contains all the feed items maintained by the server that fall within the subscriber's filters for that channel—regardless of whether any of those items have been updated since the last client request.

A client MUST interpret a full update as a complete and up to date snapshot of the given channel and use it to completely replace any existing feed item collection for the given channel. A full update SHOULD not contain any delete items, as they are not meaningful when doing a full replacement of the channel's feed item collection.

### 6.3.2 Delta updates

A delta update for a channel contains only those feed items that are new, have been updated, or were deleted, since the client last received an update from the server. When a client requests an update from the server, it can send the sync level that it currently has for that channel. The server compares the client's sync level to its own sync level for that channel's feed item collection. The server then returns to the client only those feed items whose sync level is greater than the one specified by the client.

The client **MUST** treat a delta update as an incremental change to its existing feed item collection, and only apply the necessary changes without affecting any unchanged feed items.

Each feed item has an intrinsic property named `delete-after` that indicates whether the item has been deleted from the collection. A value of `1` assigned to this property indicates that the item has been deleted; a value of `0` indicates that the item should be updated or added to the client's feed item collection. For more information about how this property is encoded, see "ItemData record format".

### 6.3.3 Sync levels

Each channel's feed item collection has an associated synchronization level, or *sync level*. A sync level is a monotonically increasing value associated with feed items and feed item collections that allow a server to deliver differential, or *delta*, data updates to clients.

The server **MAY** use a UTC timestamp or any other monotonically increasing value to set sync levels. The server uses the sync level to calculate what items have changed since the client's last `GetData` request. Only those feed items whose sync level is greater than the value requested by the client—and that fall within the user's filters for that application—should be returned in the server's HTTP response.

The feed item collection that is returned as a result of a channel update has a sync level, too. This sync level is used for the next `GetData` request and indicates up to which point the client has received updates to the entire collection. The client **MUST** never generate sync levels on own; rather, it **MUST** always include the sync level previously delivered to it by the server. It can specify a sync level of `0` when it has no prior sync level stored. This prevents any side effects due to changing system clocks on the clients.

### 6.3.4 Filters

A filter represents a restrictive query on a channel's feed item collection. In other words, a filter is a collection of feed item "selectors" that specify the following:

- A feed item type
- Zero or more item IDs (optional)
- Zero or more property names (optional)

An implementation of the server side of the protocol **MUST** support filters. When the server builds the feed item collection for a particular subscription (in response to a `GetData` call, for example), it only considers those feed items of the type specified by the filter. If the channel's feed item collection maintained on the server contains no items of the specified type, then the server must not return any feed items of that type in its response. Clients

communicate changes to their filters to the server by using a SetData command that contains the updated filters encoded within a feed item collection.

In addition:

- If the filter specifies one or more item IDs, then the server must only include those feed items whose IDs are stated in the filter; if the filter doesn't specify any IDs, then every feed item of the given type is included.

In addition:

- If the filter specifies one or more property names, the server must only include in the update the feed item properties whose names appear in the filter; if the filter contains no property names then the server must include all properties of the previously matched items.

Since a channel's feed item collection often contains feed items of several different types, it's often necessary to use several filters to express a complete set of user preferences. If multiple filters match the same item, the server must return the feed item property list specified by the last selector that it processed.

### 6.3.5 Feed update process

For each active user, the server maintains the filters for each of the user's subscribed channels. When a client makes a request for new data, the server performs the following tests against each of the user's subscribed channels:

- Is channel subscription currently in active state. The server must only deliver data updates for active subscriptions (see [Subscription states](#)). If the subscription is active, continue to the next steps:
- If the client requested a *full* update (i.e. sync level = 0) for the channel, the server **MUST** return an update containing the all the feed items in the channel's data state that also fall within the user's filters for that channel (see [Filters](#)). The server **MUST** also include any root content items, filters and resources in the response so that the client can completely refresh its cache for the channel.
- If the client requested a *delta* update for the channel, the update that the server returns must only those contain those feed items whose sync level is greater than the channel's sync level currently maintained by the client, and that also fall within the user's filters for the channel. In addition, the server's response **MAY** also contain any updates to filters, root content items or other channel components that have changed on the server side since the last client request.

Also if a channel subscription is currently in a "targeted" state the server must not return any data for that channel. For more information about subscription targeting, see [Subscription targeting](#).

## 6.4 About channels

A channel is a user-facing application and its associated data set, cached by the client. The client makes requests to the server on behalf of each channel that it's subscribed to. Channel applications do not directly communicate with the server; rather they execute commands and queries that result in protocol communications. The server commands that can be used are described in detail in section "Protocol commands overview".

In addition, channel applications may communicate with other servers (such as a web server or Adobe Media Server) using client-side APIs, subject to the permissions set for the application. For more information about permissions, see [Channel permissions](#).

The protocol supports “standard” channels (such as a “News” or “Sports” channel), as well as two *system* channels, called the Subscription and Catalog channels. The following sections briefly describe the feed item types used by standard and system channels.

### 6.4.1 Standard channels

Standard channels define their own feed item types (as well as item IDs, property names, and so forth) to represent their data. For instance, a channel that displays weather data to users might define feed items of type `location`, each of which is assigned an item ID equal to the location’s zip code (94103, for example), and that defines two feed item properties named `highTemp` and `lowTemp`.

*Note:* Channel-defined feed item types MUST not start with the prefix `feedapp-`. This prefix is reserved for item types that are intrinsic to the protocol, as seen below.

In addition to the channel-specific feed items, each standard channel’s feed item collection contains several “well-known” feed item types that represent various parts of the channel’s functionality, such as the channel’s main binary application, the user’s preferences for the channel, and other channel elements.

The following table lists the well-known feed item types within a standard channel’s feed item collection. Complete schemas for all feed item types are provided in [Appendix A: Feed item schemas](#).

Feed item type	Contains	Description
<code>feedapp-bininfo</code>	Root content items	Contains the channel’s application binary (such as a SWF application) and a channel icon, if present. For detailed schema information, see <a href="#">Root content items (feedapp-bininfo)</a> .
<code>feedapp-pref-select</code>	Filters	Collectively, items of this type comprise the filters for the channel. For detailed schema information, see <a href="#">Filters (feedapp-pref-select)</a> .
<code>feedapp-pref-settings</code>	Settings	Settings are name/value pairs the channel developer can use to store application state and configuration settings. For detailed schema information, see <a href="#">Settings (feedapp-pref-setting)</a> .
<code>feedapp-localeinfo</code>	Localized channel resources	Localized resource protocol items can be used to deliver and support preloaded language pack SWF files for a given channel. Instead of having to select items specific to a locale in user defined filters, the system allows to add these resources to the feed where they are selected automatically based on the device’s locale information. For detailed schema information, see <a href="#">Localized</a> .

## 6.4.2 System channels

The protocol defines two system channels called the “Subscription” and “Catalog” channels.

The purpose of the Subscription channel is to describe a user’s subscription state data—what channels a user is subscribed to, and the terms of those subscriptions. Each user **MUST** be automatically subscribed to the Subscription channel upon a successful login. In addition, the Subscription channel’s ID must always be hexadecimal `ffff`.

The purpose of a Catalog channel is to expose clients to the channel content available on the server. A client may provide a user interface for users to browse the catalog and make new channel subscription requests. A client may be subscribed to zero, one, or more Catalog channels.

The feed item types and properties that comprise the Subscription and Catalog channels are well-known. For example, the Subscription channel’s feed item collection contains items of type `feedapp-sub`, which represent a user’s list of subscribed channels; a Catalog channel’s feed item collection contains feed items of type `category` to represent the channel categories available on the server. By inspecting the items within these types the client can, respectively, determine what channels a user is subscribed to, and what content categories are available on the server.

This section discusses the feed item types in the Subscription and Catalog feeds, their basic structure and meaning. Full schemas for all system feeds are provided in [Appendix A: Feed item schemas](#).

### 6.4.2.1 Subscription channel

The Subscription channel’s feed item collection is composed of the feed item types described below. For a complete schema for each of these feed items, see [Subscription feed schema](#) in Appendix A.

Feed item type	Description
<code>feedapp-sub</code>	<p>Describes the state of a user’s channel subscription, such as its time period, whether it’s a trial subscription or not, and so forth.</p> <p>There is always one <code>feedapp-sub</code> item for every subscribed channel.</p>
<code>feedapp-info</code>	<p>Contains information pertaining to each subscribed channel as a whole, rather than to the individual subscription held by the user.</p> <p>There is always one <code>feedapp-info</code> item for every subscribed channel.</p>
<code>feedapp-download</code>	<p>Delivered when the client has been “targeted” by the server to receive a new channel subscription, or an update to a subscribed channel (see <a href="#">Subscription targeting</a>).</p> <p><code>feedapp-download</code> items combine most of the properties of <code>feedapp-sub</code> and <code>feedapp-info</code>.</p>
<code>feedapp-preview</code>	<p>Provides the URL to a channel preview image.</p>

### 6.4.2.2 Catalog channel

A Catalog channel's feed item collection defines a set of categories, each with a unique descriptive name (such as "Games" or "News") along with a small number of descriptive properties for each. Each category describes a set of channels that belong to it. Each channel may belong to more than one category. Thus a many-to-many relationship exists between channels and categories.

Each Catalog channel may be assigned a content type of `discovery`, which allows a client to filter catalog channels from other channel types (see [About channel namespaces and content types](#)).

Its feed item collection is composed of the following feed item types:

Feed item type	Description
<code>category</code>	Represent the categories defined on the server, including the category display name and order. The ID of each <code>category</code> item is the category's name. For example, the ID of a category named "News and Headlines" might be "news".
"Channel" feed items	For each <code>category</code> feed item type, there exists an additional set of feed items whose type matches the category ID. Each of these "channel" feed items represent the channels available for subscription in that category.

For a complete schema description of these types, see [Catalog feed schema](#).

## 6.5 Channel subscription model

The protocol models a subscription as the association of a client with a channel. Subscriptions have various properties, such as cost, lifetime, and progress through a simple series of states over time, such as "subscribed", "expired", and so forth.

A client can modify a user's subscribed channels by sending the server an update to the client's Subscription channel's feed item collection in a `SetData` call. The server will echo the results of the command by returning a feed item collection for the subscription feed back to the client. Since all `SetData` commands are asynchronous from the client's perspective the client can observe the results by reading the current state of the feed item collection of the Subscription feed.

For more information about subscription operations a client can perform, see [Subscription changes](#).

### 6.5.1 Subscription states

A channel subscription can be in one of the following states. Note that there are transitional states that occur when either the client side or the server side made a change and the change was not yet communicated and acknowledged by the other side:

- `unsubscribed`—Client is not subscribed to the application.
- `deliver`—The client has requested that the server deliver a channel's binaries and data. (Transitional)
- `delivered`—The server has delivered a channel's binaries and data to the client. (Transitional)
- `installed`—The client has successfully installed a channel's binaries. (Transitional)

- `subscribed`—Client is subscribed to the channel, but is not receiving data updates. This state is sometimes referred to as deactivated.
- `activated`—Client is subscribed to the channel and is receiving data updates.
- `expired`—Subscription time period has expired. (Transitional)
- `cancelled`—Subscription was either cancelled by a request from an external site via a server API, or server administrator action.

The `expired` and `cancelled` subscription states assume a special role and used to implement a two-step transition. The first step is an event that occurs on the server side, such as a time-based or API event which leads to the subscription's expiration or cancelation, or an administrator's action. As a result of the expiration or cancellation, the server updates the user's Subscription feed with the particular subscription's state marked as either 'canceled' or 'expired'. The second step is a confirmation by the user who removes the subscription and with that the transition is completed.

Similarly, the `deliver`, `delivered` and `installed` subscription states have a special role that allows the server to track a channel's download and installation progress from the time a client makes a new subscription request, to when it actually installs the channel binaries. This process is described in detail in the use case [Client-initiated new subscription](#).

## 6.5.2 Subscription modifications

Modification to the Subscription channel's feed item collection triggers changes in the client's subscription lineup. For example, if a client wishes to unsubscribe from a channel, it will delete the representative item from the subscription feed item collection. If the client wants to subscribe to a new channel it will add a new feed item into the subscription feed item collection. For details on this process, see [Subscription changes](#).

A client SHOULD offer APIs for each of the commands that start or modify a subscription, preventing direct access to the subscription feed item collection. One reason for that is that the client MUST ensure that the aforementioned sequence of transitional states is executed and that the error cases are handled.

## 6.5.3 Channel download and confirmation

The subscription protocol follows a two-step "download and confirmation" process that tracks the progress of a new channel download.

- 1) Download phase
  - a. The client checks the available space and validates the items for correctness. For example the main channel SWF's version must be compatible with the version of the Flash Player on the client. To determine space requirements, the client must check the channel's feed size properties reported in the Catalog feed (`feedHeaderSize`, `feedSize`, `scratchpadHeaderSize` and `scratchpadSize`). Or, if the client was targeted with a new subscription, the feed size information will be reported in the `feedapp-download` item sent by the server.
  - b. The client then makes a request to the server for a new channel subscription. With this request the client sends the offered terms of the subscription (price, periodicity) back to the server.

- c. The server checks the access rights and compares the terms to its own terms for the channel. If the terms are compatible and the client is allowed to subscribe to that channel the server enters the “contract” by creating a subscription item in its persistent store.
  - d. The server delivers the channel’s feed item collection, including channel binaries, regular channel data, and other standard feed items.
- 2) Confirmation phase
- a. After installing the binaries, the client sends a confirmation back to the server, acknowledging a successful receipt and installation of the channel.
  - b. The server updates its subscription item and enters the non-transitional state of `subscribed` or `activated`.
  - c. The server delivers the item that represents the subscription as part of the updated Subscription feed item collection.
  - d. The client integrates this item into the local copy of the feed and with that enters the non-transitional state of `subscribed` or `activated`.

This process is illustrated by the sequence diagram shown in [Client-initiated new subscription](#).

#### 6.5.4 Subscription targeting

*Subscription targeting* is a protocol feature that enables the server to request that a given client make a new subscription request. To do this, the server sends the client a special feed item type named `feedapp-downloading` the Subscription feed. This special feed item indicates to the client that it SHOULD initiate a subscription download request for the specified channel. This starts the two-step download process describe above, as described in [Channel download and confirmation](#).

This process is illustrated in the sequence diagram shown in [Server-initiated subscription add](#).

A client may decide to decline a targeted subscription by returning the `feedapp-download` item to the server marked as a delete item. The server then updates the subscription state and echoes the `feedapp-download` delete in Subscription feed by returning a deleted `feedapp-download` item to the client.

### 6.6 About channel namespaces and content types

Each channel can be assigned a *content type* and a *namespace*. Content types allow a client or server to dynamically filter or query channels of a certain type. Namespaces provide a way to disambiguate multiple feeds of the same content type.

Content types and namespaces are predominantly used in catalog channel applications to implement client-side Catalog browsing logic. A client may be subscribed to zero, one, or more Catalog channels. The protocol defines a special channel content type called `discovery` that identifies Catalog channels. This provides the client with a way to query all its subscribed channels for those channels that provide catalog information. To distinguish among multiple catalog channels, the client can use the channel’s namespace (`ns` property).

A channel’s content type and namespace are described by the `contentType` and `ns` properties of the `feedapp-info` item representing the channel. See [Channel information item schema \(feedapp-info\)](#).

## 6.7 Device capabilities and capability negotiation

The protocol provides a way for a client to communicate the capabilities of its host device to the server, so that the server may deliver content that is appropriate for the target device. This section describes generally how this process is negotiated between a client and server.

When client first attempts to login to a server (and in certain other scenarios outlined below), it must send the server the capabilities of its device in a specially-formatted string, called the *device capabilities string*. These capabilities are organized into *device profile class types*. A device profile class type is, essentially, an aggregation of device capabilities. For example, the profile class type `display` defines two capabilities named `height` and `width`. For details on the formatting of the capabilities string, see [Device capabilities string](#).

Internally, the server maintains a list of *device profiles*, which are aggregations of profile classes. The device profile only has one profile per profile class type. For instance a device profile may consist of the profile classes `QVGA_Rotate`, `BREW3_0_1_AUDIO` and so on. The first profile class is of the profile class type `display` and the second is of the profile class type `audio`. Each device profile has a unique ID. When the server receives a device capabilities string from a client, it attempts to match the specified capabilities to one of its device profiles. The server then returns the ID of the selected device profile to the client. On subsequent logins, the client need only send its stored device profile ID to establish a session.

Note the following about device profile classes:

- The naming scheme for the profile classes is not visible to the client nor is it part of the client server protocol.
- The process by which the server maps the capabilities to the profile classes, and ultimately to the device profile, is not visible to the client nor is it part of the client server protocol.

The reason why profile classes are mentioned here as part of the components of a device profile is to explain the grouping of capabilities into sections where each section corresponds to a profile class type such as `display`, `audio` etc. The profile class types provide an organization scheme and namespace for capabilities.

For a full list profile class types and capabilities recognized by the Adobe Mobile Server, see [Appendix D: List of device profile classes and capabilities](#).

## 6.8 Server initiated communications

The majority of protocol communication is directed by a client's HTTP requests. If the server wants the client to initiate a request, it can send the client an application-directed SMS message. The SMS message payload indicates the required action of the client, such as logging in again, or calling `GetData` to re-synchronize its feed item collections with the server.

For more detail about the SMS payload, see [SMS signaling](#).

## 6.9 Channel permissions

Each channel maintained in the server's content catalog can have various permissions associated with it. The client runtime that hosts the channel applications MUST control what operations and features that channel can use. For example, a client may wish to control whether a channel has HTTP access, or if it can read or write settings data to persistent storage. A server implementation only needs to store and retrieve the permission settings with the channel's meta-data.

A channel's permissions are encoded in the `access` property of the `feedapp-info` item that represents the channel's public properties (see [Channel information properties \(feedapp-info\)](#)). Conceptually, this property is a bit list encoded as a hexadecimal string. When decoded from its hexadecimal form, this string is a binary sequence (for example, 0100001) where each bit represents a permission flag. The order of the bit list is big-endian, with the first permission bit appearing last in the sequence. Each permission bit has a predefined position that is fixed forever. If a new permission is added to the permission model, the bit with the next available position is reserved.

To allow custom permissions to be added without having to resolve conflicts between different client implementations, the `feedapp-info` item's `accxx` property must be used instead of the `access` property. Proprietary permissions added by Adobe will use the `accx` property.

The full list of permissions bits supported by the Adobe Mobile Server is listed in [Appendix C: Channel permission bits](#)

Note that sometimes more than one bit can cause certain functionality to become accessible, and sometimes a combination of access permissions are needed to perform a high level operation. For example, either of the permissions "Read access to every channel's data", "Read access to my data" and "System level" will lead to read access for other channel's data. Therefore revoking "Read access to my data" will effectively not revoke the permission. On the other hand, in order to play an interactive Flash Media Server video stream, the permissions "Network Video" and "Socket" must both be granted. For a detailed description of the permissions bits and their interactions, see Appendix C: Channel permission bits.

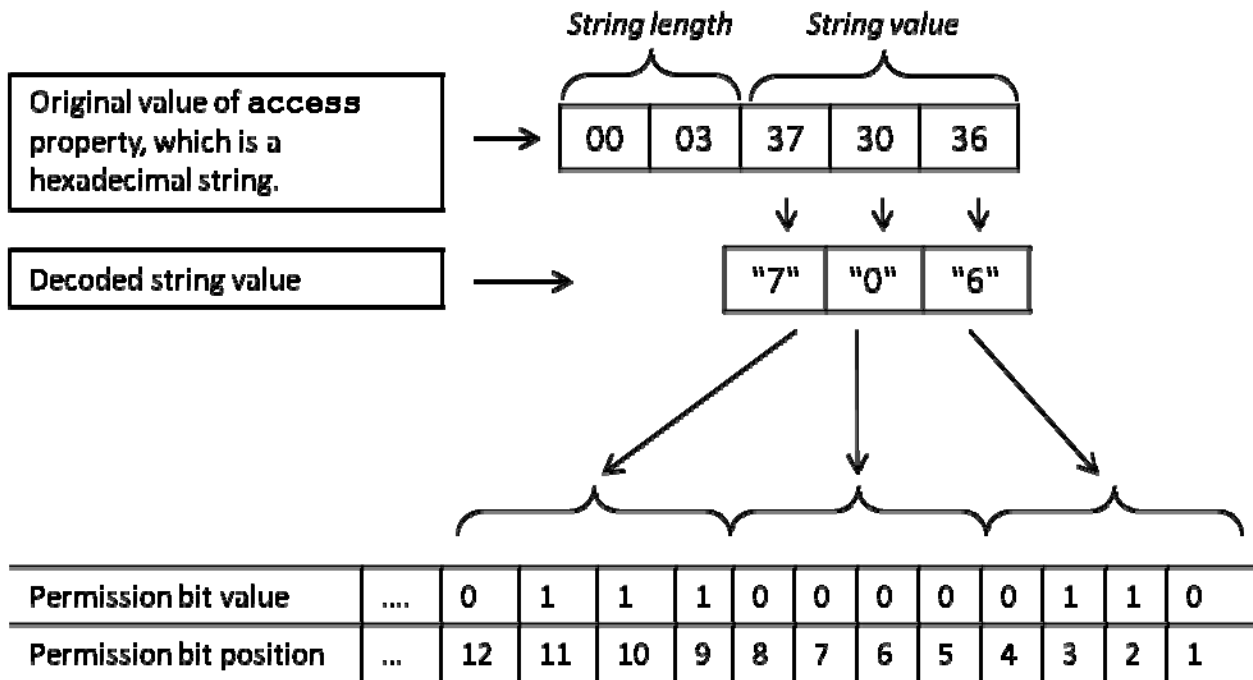
For example, suppose that a client is hosting a channel that has been assigned the following permissions:

- System Level Access (no restrictions): **Off**
- Access to own data: **On**
- Access to all data: **On**
- Access to system commands: **Off**
- Use FSCommand: **Off**
- Read settings data: **Off**
- Write settings data: **Off**
- Access to scratchpad: **Off**
- Access shared data: **Off**
- Channel runs in stand-alone mode: **On**
- Channel can launch other apps: **On**
- Access to Dialog: **On**

The first permission determines if the channel has unrestricted access to every feature, the second specifies whether the channel can read its own data (that is, its feed item collection), and so forth. Furthermore, the client MUST assume that all other permissions bits that follow after "Access to Dialog" are off.

In this scenario, the `access` property of the `feedapp-info` item representing the channel would be assigned the hexadecimal value of `706`. The `access` property is encoded in the binary transport format as a string. Strings are represented as a byte sequence, with a leading two bytes indicating the length of the string, followed by the character encoding of the digits. In this example, assuming a UTF8 encoding for strings, the `access` property's value would be encoded for transport as the byte sequence `00 03 37 30 36`. Decoded into its UTF-8 representation provides the character sequence `706`. Lastly, the client must expand these string values into their binary representation, which provides the list of permission bits.

The following diagram illustrates this concept.



## 6.10 Error handling

Errors in the protocol can occur in any of the following layers (from the bottom layer to the top layer):

- **Transport layer errors** refer to problems that might occur in the underlying HTTP protocol that a client and server use to communicate.
- **Request layer errors** refer to high-level errors such as invalid authentication credentials or an expired session ID in a client request.
- **Channel errors** are errors that a server encounters while performing a channel-specific operation requested by the client.

Below is summarized the general error reporting and handling mechanism for each layer, including protocol errors at the transport and application level. Additional command-specific and channel-specific error codes reported through this mechanism will be described together with the command itself in a later section. Errors in a higher

layer will be returned in a response without error of the underlying layer. For example a login command fails with an error that is encoded in a HTTP POST response with error code 200 (OK).

## 6.10.1 Transport layer

All messages must be contained within valid HTTP POST requests (for client-initiated messages) or HTTP Responses with a result code of 200 (for server-responses); anything else is considered an illegal message.

In addition, any request whose HTTP payload is not either `application/x-www-form-urlencoded` or `application/x-flashcast-data-update` is also illegal. If a server receives a HTTP request that does not contain one of these formats it should ignore the request entirely. If a client receives an HTTP response from a server that contains an invalid content type, the client should retry the associated request at a later time.

Client errors encountered at the TCP/IP layer or below (such as not being able to establish a TCP connection to the server, or not being able to resolve the server's DNS hostname) should be treated the same as described above.

## 6.10.2 Request layer errors

If no errors are encountered in the transport layer, the HTTP payload can be interpreted as a protocol message. In the event of success, a normal response message will be returned, as defined by that particular command. In the event of some sort of high-level error in the processing of the command, the server will return a URL-encoded message (that is, a message with mime type `application/x-www-form-urlencoded`) containing the following properties:

- `result`—Describes the sort of failure that has occurred, if any. If the command issued by the client was successful, and that command normally uses the URL-encoded format for its successful responses, this property will be set to **ok**.
- `desc:`—An optional property that, if present, contains a string that describes the error condition. This property will not be present if `result` property is set to **ok**. These descriptive messages may be localized (see [Error message localization](#)).
- `retry-interval`—An optional property that is only present when `result` is set to `fail-retry`; specifies the number of seconds the client should wait before retrying the previous operation.

The table below lists all possible values for `result`, their meanings, and the action required by a client upon receiving an error.

Value of result property	Description	Action required by client
<code>ok</code>	The last operation was successful.  This return value is only used in commands that use URL-encoded messages for successful responses, specifically, login and logout.	No action required.
<code>fail-session</code>	The last operation was attempted on an expired session.	Establish a new, valid session by executing a login operation, then immediately retry the failed request.

<code>fail-retry</code>	The last operation failed, but should be retried at a later time. An additional URL-encoded property returned in the response, <code>retry-interval</code> , specifies the number of seconds the client should wait before retrying the previous request.	Retry the request after the number of seconds specified by the <code>retry-interval</code> property has passed.
<code>fail-retry-capabilities</code>	Indicates one of the following: <ul style="list-style-type: none"> <li>The device profile associated with the client has been updated on the server and might not match the device's actual capabilities.</li> <li>The client has attempted to login without providing its device capabilities.</li> </ul>	Retry the login request with the device capabilities header.
<code>fail-auth</code>	The client failed to authenticate with the server.	The client should not retry the request unless it modifies the authentication credentials that it transmits.
<code>fail-protoversion</code>	The client has attempted to communicate with the server using a version of the protocol that the server does not support.  In this case, the server returns an additional property in the response called <code>protoversion-supported</code> that specifies the protocol version the server is expecting.	The client should not retry the request unless it first modifies the value of the <code>protoversion</code> property that it transmits in the request.
<code>fail</code>	The last operation failed and should not be repeated.	The client should not repeat the last request, but is free, however, to continue to make other requests.
<code>fail-access</code>	Similar to a <code>fail</code> message (the last operation failed and should not be repeated, but the client can continue to do other things) but also indicates that the failure is due to the lack of access rights.	The client should not repeat the last request, but is free, however, to continue to make other requests.

Note the following about possible return values:

- The `fail-auth`, `fail-protoversion`, `fail-access`, and `fail` messages are final failures and the request should not be retransmitted.
- The `fail-auth` and `fail-protoversion` prohibit the service from working at all.
- The `fail` and `fail-access` signals should be noted by the client, but the client can continue interacting with the server.
- The `fail-retry` means the same request should be attempted at a later date specified by additional the `retry-interval` property.

### 6.10.3 Feed layer errors

As explained previously, if the server encounters a high-level error while processing a client's command, it returns the error state in a URL-encoded response message. However, for errors that occur during channel-specific operations—such as attempting to subscribe a user to a new channel—the server encodes the error states within a special feed item of type `fc-result-info`. This item contains the same properties that are returned for high-level errors, namely, `result`, `desc`, and `retry-interval`, as well an additional property that indicates the operation that generated the error.

Feed layer errors are required to communicate the results of a command that operates on multiple channels. In such a scenario, the outcome cannot be encoded in a simple result property. For example a `GetData` command can request an update for multiple channels at once. Some of those updates may succeed, resulting in the server returning feed item collection updates for those channels; others may fail because there was no valid subscription for the channel.

Thus, when parsing a message response from the server, the client must inspect the contents of the message for `fc-result-info` items to determine if the operation it was attempting to perform was successful. The lack of any `fc-result-info` items in the response indicates that the last operation was an unqualified success.

For a complete schema reference for `fc-result-info` items, see [Error feed items \(fc-result-info\)](#).

#### 6.10.4 Client errors

If a client encounters an error—such as during its normal processing, The client reports errors to the server by sending the server an update to Subscription channel's feed item collection. The update must contain a `feedapp-sub` item whose `clientError` property is set to one of the established error codes (see Appendix B: Client error codes).

#### 6.10.5 Error message localization

All error messages returned within the `desc` property, as described above, **MUST** be localized according to each client's language. The server **SHOULD** load the descriptive string from a resource file based on one of the following:

- The `currentLocale` device capability set by the client.
- The `locale` capability provided in the device profile.
- The system default locale, if neither of the previous settings were available.

Locale capability consists of language code (i.e. `en`) and a country code (i.e. `US`) separated by a dash (i.e. `en-US`).The following reference document provide language and country codes:

- "Codes for the Representation of Names of Languages" ([http://www.loc.gov/standards/iso639-2/php/code\\_list.php](http://www.loc.gov/standards/iso639-2/php/code_list.php))
- "English country names and code elements" ([http://www.iso.org/iso/country\\_codes/iso\\_3166\\_code\\_lists/english\\_country\\_names\\_and\\_code\\_elements.htm](http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm))

## 7 Protocol commands overview

This section describes each of the protocol commands—Login, Logout, GetData, and SetData—including the basic request/response model for each, and the types of channel operations that each command can be used to perform.

## 7.1 Login command

The Login command creates a new client session with a server. A client must send a Login command under any of the following conditions:

- The first time the client is ever launched on the device, or after a client reset.
- Anytime the client receives a `fail-session` error from the server (see [Error handling](#)).
- If the device's capabilities have changed since the last time it has logged in. The client typically performs this check once upon start-up. For more information about device capabilities, see [Device capabilities and capability negotiation](#).

The client must be pre-configured with the URL endpoint for the Login command. Upon a successful login, the server returns the remaining command URLs (for Logout, GetData, and SetData) in its response. The client must use these URLs for the remainder of its session.

The request URL for the Login command consists of the protocol base URL concatenated with `/login`, or:

```
protocol-base-URL + "/login"
```

### 7.1.1 Login request message

The client includes in its Login request a URL-encoded message that specifies its authentication credentials, device profile, and other properties.

The table below lists and describes the properties common to all login request messages. Not shown in this table are any authentication credentials required by the specific implementation, as authentication schemes can vary from one implementation to the next. For more information about authentication options, see [Authentication](#).

Property	Data type	Description
<code>protoversion</code>	String	The version of the protocol that the client is attempting to negotiate with the server, consisting of major and minor version numbers separated by a dot (".").  This value must be set to "2.0".
<code>profileid</code>	String	A string that specifies the profile of the client device; profiles are defined on the server and uniquely identify the set of capabilities supported by the device.
<code>ts</code>	Timestamp	Indicates when the device profile specified by <code>profileid</code> was last changed on the server. The client uses this value to determine if a profile re-evaluation is needed since profile configuration is changed on the

		delivery server side.  A client can also set this property to the special value of <code>-2</code> , which instructs the server to return a device-capabilities string for the device profile specified in the request. For more information, see Login response message.
<code>clientstate</code>	Integer	Specifies the current state of the client.  Valid values are 3 (active), 2 (background), 1 (hibernate).
<code>currentlocale</code>	String	The locale currently used in the client consisting of a language code and a country code separated by a dash (en-US, for example) The client should send this parameter on every login request.

### 7.1.2 Login response message

If the server successfully receives and processes the Login command, it must return a URL-encoded message that contains a property named `result` assigned a value of `ok`. In addition, the server's response header must include a `flashcast-fetch-time` Pragma directive set to 0. This instructs the client to immediately call `GetData` for all its subscribed channels.

If the server encounters an error handling the command, it must return a URL-encoded message with a valid error string assigned to the `result` property. For a list of valid error values for `result` and their meanings, see Error handling.

Assuming a successful login, the server must return a URL-encoded response message that contains the following properties:

<b>Property</b>	<b>Data type</b>	<b>Description</b>
<code>result</code>	String	Indicates if the operation succeeded or not. If the command was successful this field contains "ok"; otherwise it contains one of the error strings described in Error handling
<code>session</code>	String	A string that uniquely identifies this client session.
<code>logout</code>	String	The URL the client must use when sending the <b>logout</b> command in this session.
<code>getData</code>	String	The URL the client must use when sending the <b>getData</b> command in this session.
<code>setData</code>	String	The URL the client must use when sending the <b>setData</b> command in this session.
<code>profileId</code>	String	A string that identifies of the set of capabilities on the device.
<code>ts</code>	String	A timestamp that specifies the last modified time of the device profile configuration on the server.

If the client's request message contains the special `ts` value of `-2`, the server must return the device capabilities string for the device specified by the request message's `profileId` field. The server must return a URL-encoded device capabilities string in the `flashcast-capabilities` HTTP response header, along with the error message `fail-retry-capabilities` (as described in [Error Handling](#)). Upon receiving this error message, the client should retry the login request and send the full device capabilities string.

The `logout`, `getData`, and `setData` properties in the server's response specify the URLs the client must use to invoke the remaining protocol commands. These URLs may be absolute or relative; if relative, they must be concatenated with the protocol base-URL; otherwise they may be used as-is.

### 7.1.3 Session ID options

To avoid sending authentication credentials with every request, the protocol uses session IDs.

A client session may be ended in one of several ways.

The session may expire due to a server-maintained inactivity timer. Every client-initiated command in the protocol (except for Login) can result in the server returning a `fail-session` error-code (see [Error Handling](#)). This means that the command was attempted on an expired or invalid session; in this case, the client must send a new Login request, and then immediately reissue the original command again.

In response to a successful Login request, the server must return a session ID to the client. The server **MUST** provide the session ID in both of the following ways:

- Embedded in the `session` variable in its URL-encoded response.
- In a `Set-Cookie` header. The client must save this session ID and send it with all subsequent requests in the corresponding `Cookie` HTTP header.

### 7.1.4 Device capabilities string

In some circumstances, the client must send the server a specially-formatted string in its Login request that describes the capabilities of the device, such as its screen dimensions, supported networks, and other features. The client must pass this string to the server in a HTTP header named `flashcast-capabilities`. A client **MUST** include the device capabilities string in its Login request under either of the following circumstances:

- The capabilities of the device have changed.
- The server returned a `fail-retry-capabilities` error message as the result of a Login or GetData request.

The device capabilities string is a set of URL-encoded set of name-value pairs, organized into groups called *profile class types*. Each profile class type defines a subset of device characteristics, such as supported networks, display characteristics, and so forth. The capabilities string takes the following form:

```
[profileclasstype1]name1=value1&...nameN=valueN...[profileclasstypeN]name1=value1&...nameN=valueN
```

When sending its device capabilities string, the client should set the `profileId` and `ts` message properties to `-1`; the server must ignore any other values of `profileId` and `ts` that the client sends with the request.

In its response, the server returns the ID of a device profile whose capabilities match those provided in the device capabilities string. Device profiles are maintained by the server, and encapsulate the capabilities of a given device. Subsequently, the client need only send its device profile ID in the Login request message's `profileId` property.

For general information about device capabilities, see Device capabilities and capability negotiation.

### 7.1.5 Authentication options

The protocol does not prescribe a specific means for a client to authenticate itself with a server. Each implementation is free to establish their own authentication scheme.

For example, in a trusted network the Login request might contain one additional parameter named `deviceId` whose value uniquely identifies the device on the network (such as its assigned phone number or similar identifier). Another implementation might require a client to provide both a device identifier and a password to authenticate, in which case the login request might contain two additional, such as `deviceId` and `password`.

## 7.2 Logout

The Logout command explicitly terminates the client's session with the server. Following the successful processing of the logout command by the server, the protocol URLs provided by the server at login become invalid, and should no longer be used.

This command is optional since most clients will never send explicitly send a logout command.

### 7.2.1 Logout request message

The request URL for the Logout command is provided by the server in its response to a successful **login** command. This might be a fully qualified URL, or relative to a protocol's base URL.

The client should not include a request message body with the logout command. The server must ignore any payload in the body of the HTTP POST.

The client **MUST** include the saved cookie header that was received in the login response. The server **MUST** use the cookie header to lookup the session and logout the client.

### 7.2.2 Logout response message

If the server receives and successfully processes the Logout command it must return a URL-encoded message with the `result` property set to `ok`.

If the server encounters an error it must return a URL-encoded message containing a `result` property set to a valid error string. For a list of valid error strings, see Error handling.

## 7.3 GetData command

The GetData command instructs the server to send updates for one or more of its active subscriptions. The server's response, assuming a successful operation, is a binary-encoded feed item collection `update` containing the new feed item collections for each of the client's subscribed channel. An error resulting from a GetData call may either be returned as a URL-encoded message (for high-level errors, such as an invalid session ID)

A client can request a delta or full update from the server for each channel (see Full and delta updates).

If a client requests delta update for a channel, the server MAY choose to deliver a full update, if it determines it would be more efficient. For instance, suppose that a client had not requested an update from the server for a long time (say, several weeks or longer) and then requests a delta update. The server may decide that the only way to bring the client up to the current sync level is to send the channel's entire feed item collection. The server MUST not return an error just because the client did not get an update for a long time.

Although the protocol permits a client to request an update for a single channel, each GetData command should request updates for **all** the user's active subscriptions.

The collection's `syncLevel` attribute indicates the `syncLevel` of the collection maintained by the client. If the client has no current `syncLevel` (because it has never performed a `getData` for that channel, or perhaps because it has lost the information), it should use the `syncLevel` of 0.

### 7.3.1 Request message

The GetData request must contain a URL-encoded message that specifies the channel, or channels, that it wishes to update. For each channel, the client can also specify whether it would like a full or delta update.

If the locale has changed since the last time the login and device capability negotiation.

The GetData request message may contain the following properties:

Property	Data type	Description and formatting
<code>id</code>	String	<p>Specifies the ID(s) of the channel(s) that the client wishes to synchronize with. If the client doesn't specify any channel IDs in the request message, then the server MUST return updates for ALL of the user's active subscriptions.</p> <p>To request an update for a single channel, a client must specify the channel's ID with the following form:</p> <pre><code>id = channelID</code></pre> <p>To request an update for multiple channels, a client must specify multiple IDs with the following form, where <i>n</i> is an integer that starts with 1 and is incremented for each channel being synchronized.</p>

		<p><code>idn = channelID</code></p> <p>For example, to synchronize two channels with IDs of '123' and '987', respectively, the client request should contain the following properties:</p> <p><code>id1=123</code> <code>id2=987</code></p>
<code>delta</code>	String	<p>Specifies how the synchronization will take place for the associated channel ID.</p> <p>To request a delta update for the specified channel, a client should set <code>delta</code> to the sync level last returned from the server for that channel's feed item collection. The server should return only those feed items whose sync level is greater than the one provided in the request.</p> <p>To request a full update for a given channel, the client should request set <code>delta</code> to 0. This might be necessary, for example, if the client lost its cached channel data, or after a client reset.</p> <p>When requesting an update for a single channel (that is, with a single <code>id</code> parameter), use the following form:</p> <p><code>delta = syncLevel</code></p> <p>When requesting an update for multiple channels (that is, with an enumeration of <code>id</code> values), use the following form:</p> <p><code>deltaN= synclevelN</code></p> <p>If you use this form, there must be a matching <code>idN</code> property for every <code>deltaN</code> property.</p>
<code>clientstate</code>	String	<p>(Optional) Specifies the current state of the client. This property does not affect content synchronization; rather, it provides a way for the client to send information about a state change to the server.</p>
<code>currentlocale</code>	String	<p>(Optional) Specifies the current locale the client has switched to after the last connection to the client. The client should send this parameter only if the current locale setting has changed on the device</p> <p>The value of this property consists of a language code and a country code separated by a dash (for example, <code>en-US</code>).</p>

### 7.3.2 Response message

If the server successfully receives and processes the `GetData` request, it should return a binary-encoded feed item collection update for all requested channels. If a requested channel's feed item collection has not been updated on the server since the previous synchronization, the server may not return a feed item collection for that channel.

If *none* of the requested channels has new data available, the server may respond to the GetData request with a 0-byte payload. The client can inspect the Content-Length HTTP header in the response message to determine if the response contains any data.

The data updates returned for each channel will either be delta or full updates. If the request message specified a full update (`delta=0`), then it is guaranteed that the collection returned will in fact be a full update. If a delta update is requested (`delta=<any sync level greater than 0>`), the server should normally return a delta update; however, if the server determines that it would be more efficient to return a full collection for that channel, it may do so.

The client can optionally pass the client's locale information (country and language codes) in the GetData request. The locale setting value may be different than the one specified by the device capabilities string the client provided in its previous login. If the server encounters an error while processing the GetData command, the server must return one of the following, depending on whether the error was a high-level request error, or a channel-specific error:

- For high-level request errors, such as an invalid session ID, the server must a URL-encoded message with the `result` property set to a non-OK value. See Error handling for details about valid error string values
- For channel-specific errors, the server must return an `fc-result-info` feed item within the collection of the channel where the error occurred.

## 7.4 SetData

The SetData command allows a client to send the server feed item updates for one or more channels. The server interprets the update as an operation for it to perform. For example, to request a new channel subscription a clients sends the server an update to the Subscription channel's feed item collection; in turn, the server interprets this change as a subscription request.

The request message the client sends with SetData must be a -binary-encoded feed item collection containing updates for one or more channels.

In response to a successful SetData command, the server returns a binary update to the client. that contains feed item updates for the requested channels.

The URL for the SetData command is returned in the server's HTTP response to the Login command. This URL may be relative or absolute. If relative, it should be concatenated with the client's protocol base URL.

### 7.4.1 Request message

The SetData request message must be a message in the binary transport format (see Binary transport format) containing one or more feed item collections. Each collection contains an update for a particular channel, specified by the collection's ID. The server interprets a feed item collection update from the client as an operation for it to perform on that channel. The specific operation—for example, a new subscription request, or the addition of a new filter—depends on the channel whose feed item collection is being updated in the request, and the specific feed item types that present in the message.

For a list of the operations that a client may perform with SetData, see [Operations performed with SetData](#).

A single SetData call may contain operations for multiple channels. For example, the same SetData call can contain an update for the Subscription feed, a change to another's channel's filters, and a settings data update for another. However, a client should not send multiple commands for the same channel in the same SetData, such as a filter and settings update for the same channel. Instead, the client must separate the operations (filter and settings changes) into individual SetData calls.

There are two important differences between the semantics of a feed item collection returned in response to a GetData message, and one transmitted in a SetData message:

- If the feed item collection's `delta` attribute is 0 (false), the server must interpret the collection as a "type-by-type" replacement of its current feed item collection, rather than a full replacement of the entire collection. For example, if a channel's feed item collection on the server contains items of type A, B, C, and the incoming collection (from a SetData call) contains only items of type A and B, the items of type C in the server's collection are left alone, rather than being deleted. In contrast, when a client receives a non-delta update from the server (in response to a GetData call, for example) it must replace the entire collection in the client's local data cache.
- The collection's `syncLevel` attribute indicates the `syncLevel` of the collection maintained by the client. If the client has no current `syncLevel` (because it has never performed a `getData` for that channel, or perhaps because it has lost the information), it should use the `syncLevel` of 0. In contrast, collections returned by a `GetData` command will never have a `syncLevel` of 0.

## 7.4.2 Response message

If the client's SetData request is received and successfully processed, the server must immediately return a binary `update` message that contains the following items:

- All the feed items submitted in the client's original SetData request, if they were accepted.
- Any additional updates that occurred as side-effects of the request. For example, if a SetData request contained a new `feedapp-pref-select` item for a given channel—that is, a new filter—the update that the server returns for that channel should contain any additional feed items that fall within the user's new filters for that channel.

If there are no updates to a channel after a SetData call, the server must return an *empty* delta collection for that channel—that is, a collection with no feed items. This provides a way for the server to acknowledge that it received and processed the SetData command, even though it resulted in no changes to the channel feed item collection.

If the server encounters an error while processing the SetData command it must return one of the following, depending on whether the error was a high-level request error, or a channel-specific error:

- For high-level request errors, the server must return a URL-encoded message with the `result` property set to a non-OK value. See Error handling for details about valid error string values
- For channel-specific errors, the server must return an `fc-result-info` feed item within the collection of the channel where the error occurred. For details on the schema for `fc-result-info` items, see [Feed error items \(fc-result-info\)](#).

### 7.4.3 Operations performed with SetData

The client uses SetData to perform several types of operations. This section describes those operations and provides an overview of the request/response model for each.

The types of operations a client can perform with SetData are limited to the following:

- Subscription changes (add, modify, delete)
- Filter and settings data changes (add, modify, delete)

### 7.4.4 Subscription changes

To add, modify, or remove a user's channel subscriptions, a client sends the server an update to the Subscription channel's feed item collection. The update message contains one or more new or modified `feedapp-sub` items. The server interprets these updates as subscription change requests and takes the appropriate action.

#### 7.4.4.1 Add subscription

To create a new subscription, the client must send the server a Subscription feed update that contains one `feedapp-sub` item for each channel it wishes to subscribe to. The ID of each `feedapp-sub` item MUST be the ID of the desired channel, as reported in the channel's corresponding entry in the Catalog channel's feed.

If a subscription request is successful, the server must respond with an update to client's the Subscription feed, containing the following feed items:

- A `feedapp-info` item containing general channel information
- A `feedapp-sub` item containing information about the new subscription instance.

Additionally, the server must return the channel's initial feed item collection, which is comprised of the following:

- A `feedapp-bininfo` item that contains the channel's root content items (see [Root content items \(feedapp-bininfo\)](#)).
- Zero or more `feedapp-localeinfo` items that contain localized channel resources (see [Localized resources \(feedapp-localeinfo\)](#)).
- Zero or more `feedapp-pref-select` items containing the channel's default filters (see [Filters \(feedapp-pref-select\)](#)).
- Zero or more `feedapp-pref-setting` items containing the user's initial settings data (see [Settings \(feedapp-pref-setting\)](#)).
- The channel's custom feed items, extracted from the channel's feed item collection according to the channel's default filter, if provided by the channel publisher.

If the server encounters an error while fulfilling the subscription request it must return a binary update containing an `fc-result-info` item that describes the Error condition. ( See the *Error Handling* section for details on error values and their meanings.)

The properties of a `feedapp-sub` item that a client may send in a `SetData` request are a subset of those defined by the full `feedapp-sub` item schema (see [Channel subscription properties \(feedapp-sub\)](#)). The table below lists those `feedapp-sub` properties that a client can send in the request.

Property	Data type	Description								
<code>activated</code>	Boolean	<p>Specifies the initial activated setting for the subscription.</p> <p>If set to <code>true</code>, the client will immediately begin receiving data updates from the server for the subscription.</p> <p>If set to <code>false</code>, the client will be subscribed to the channel but will not receive any data updates.</p> <p><b>Default:</b> <code>true</code>.</p>								
<code>trial</code>	Boolean	<p>Specifies whether to start a normal or trial subscription.</p> <p><b>Default:</b> <code>false</code>.</p>								
<code>state</code>	String	<p>The client must set this property to "deliver" when requesting a subscription (see <a href="#">Subscription states</a>).</p>								
<code>price</code>	String	<p>The subscription price indicated in the Catalog feed maintained by the client. The client must send this when requesting a new subscription.</p> <p>If provided in the request, the server must compare the price received from the client to the price that it's maintaining. If they are not the same then the subscription operation must fail, and the server must return the client a full update for the Catalog feed. The client may then update subscription price and display new price to the user before re-trying the operation.</p> <p>Prices are formatted localized format (for example, EUR2.00).</p> <p><b>Default:</b> <code>null</code> (no price checks performed)</p>								
<code>period</code>	String	<p>The channel's subscription period, as reported by the client's Catalog feed. A subscription period is indicated by a period indicator (for example, "D" for day or "W" for week) concatenated with an integer that acts as a period multiplier. For example, a subscription period of three months would be represented with a <code>period</code> value of "M3".</p> <p>The table below lists the valid subscription period indicators:</p> <table border="1"> <thead> <tr> <th>Period indicator</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>D</td> <td>Day</td> </tr> <tr> <td>W</td> <td>Week</td> </tr> <tr> <td>M</td> <td>Month</td> </tr> </tbody> </table>	Period indicator	Meaning	D	Day	W	Week	M	Month
Period indicator	Meaning									
D	Day									
W	Week									
M	Month									

		Q	Quarter
		Y	Year
		P	Perpetual
		<p><b>Note:</b> If the subscription is perpetual (P) then any integer multiplier should be ignored.</p> <p>If provided in the request, the server must compare the subscription period received from the client to the one that it's maintaining. If they are not the same then the subscription operation must fail, and the server must return the client a full update for the Catalog feed. The client may then update the subscription period and display new period to the user before re-trying the operation.</p> <p><b>Default:</b> null (no checks performed).</p>	

For example, suppose the client wished to add and activate two new, non-trial subscriptions for channels with IDs ABC and 456. To do this it would make a SetData call containing a binary update message with the following properties:

```
ItemCollection(id='ffff')
  FeedItem(type='feedapp-sub', id='ABC')
    state='deliver'
    price='US2.00'
    period='M1'
  FeedItem(type='feedapp-sub', id='456')
    state='deliver'
    price='US1.00'
    period='Y1'
```

#### 7.4.4.2 Modify subscription

To modify an existing subscription, the client must send the server a binary `update` to the Subscription feed containing a modified version of an existing `feedapp-sub` item. A client can modify subscriptions in the following ways:

- Activate a subscription that's inactive, or de-activate one that's currently active.
- Request that a trial subscription be promoted to a non-trial.

A subscription that's either in a `cancelled` or `expired` subscription state may not be modified, only deleted (see [Delete subscription](#)). However, a client can send new subscriptions requests for channels in `cancelled` and `expired` states. In this case, a new subscription is created.

The table below lists and describes the `feedapp-sub` properties that a client can set when modifying a subscription, and the effects of those modifications.

Property	Data type	Description
----------	-----------	-------------

activated	Boolean	<p>Specifies whether the client should receive updates for the associated channel (<code>true</code>) or not (<code>false</code>).The <code>activated</code> property may only be modified for non-cancelled and non-expired channel subscriptions.</p> <p>If a user's subscription is inactive, the client can request that it be made active by setting this property to <code>true</code> in the <code>SetData</code> request message. When transitioning from an inactive subscription to an active one, the server must return a full update for the newly activated channel in its response.</p> <p>If a user's subscription is active, a client may request that it be made inactive by setting this property to <code>false</code> in the request. When transitioning from an active subscription to an inactive one, the server must stop sending updates for that channel to the client.</p>
trial	Boolean	<p>Specifies whether the subscription is a trial or not.</p> <p>A client may request that a trial subscription be promoted to a non-trial subscription by setting the <code>feedapp-sub</code> item's <code>trial</code> property to <code>false</code> in the request. This will end the current trial, and start a new, non-trial subscription.</p> <p>If the subscription modification is successful, the server must return a modified <code>feedapp-sub</code> item representing the new subscription, such as new <code>begin</code> and <code>end</code> dates, etc.)</p> <p>Note that toggling <code>trial</code> from <code>false</code> to <code>true</code> is illegal.</p>

For example, if a client wished to deactivate a channel with an ID of `ABC`, it would send the following feed item collection to the server in a `SetData` call:

```
ItemCollection(id='ffff'):
  Item(type='feedapp-sub', id='ABC'):
    activated='false'
```

If an illegal modification is attempted, the server must return an `fc-result-info` item in the Subscription feed that describes the error.

### 7.4.4.3 Remove subscription

To unsubscribe a user from a channel, the client must send the server a Subscription feed update that contains one `feedapp-sub` item for each channel it wishes to unsubscribe to, with each item's `delete` field set to 1. This instructs the server to delete the feed item representing the subscription which, in turn, causes the side effect of deleting the subscription itself.

In response, the server must return a binary update message that contains deleted versions of all the feed items associated with the successfully unsubscribed channels—namely, one `feedapp-info` and `feedapp-sub` per unsubscribed channel.

If the server encounters an error while performing the unsubscribe operation, it must return an `fc-result-info` item that describes the error. The ID of this item must be set to the ID of the channel whose unsubscribe operation failed. For details on `fc-result-info` items, see the *Error Handling* section for details on error values and their meanings.

### 7.4.5 Filter changes

To modify a channel's filters, the client sends the server a binary update containing a combination of new or modified instances of `feedapp-pref-select` items for the specified channel. The binary update message the client sends must be marked as *non-delta* and include the full set of filters being maintained by the client for that channel. For example, if the channel's feed item collection contains two `feedapp-pref-select` items, and the client wishes to add a third, it must send all three `feedapp-pref-select` items in its request.

To delete all the filters for a given channel, the client must send the server a single `feedapp-pref-select` item that contains no properties.

A client MAY send updates for both a channel's settings and filters in the same request, and the server MUST be able handle these updates in same request.

For a list of valid `feedapp-pref-select` properties, see [Filters \(feedapp-pref-select\)](#).

*Note:* It is at the discretion of each channel to determine if it will allow filters to be set or modified.

### 7.4.6 Settings data changes

To modify the settings for a given channel, the client sends the server a feed update for that channel containing a combination of modified and/or new instances of `feedapp-pref-setting` items.

The feed item collection used in the message must be marked as *non-delta*, and include the full set of `feedapp-pref-setting` items. For example, if the client is already maintaining two settings for a given channel, and the client wishes to add a third, it must send all three `feedapp-pref-setting` items in the `SetData` command.

To delete all `feedapp-pref-setting` items, the client issues a `setData` message containing a single `feedapp-pref-setting` item that contains **no** properties.

*Note:* It is at the discretion of each channel to determine what, if any, settings it will allow to be set.

## 7.5 Server-initiated communications

Although the majority of the protocol is directed by a client's HTTP requests, a server can notify a client with an SMS message to start a specific HTTP request. For example, suppose a server has "targeted" a client with a new

channel subscription and wants the client to make a new GetData call to re-synchronize its Subscription feed data. To do this, the server sends the client an SMS message with a binary payload that the client interprets as a request to immediately issue a GetData request.

The formatting of SMS messages varies by device and operator network. However, the message payload sent by a server must be 10 bytes long. Each byte must be an ASCII character representing a hexadecimal value. These ASCII values can then be converted to five bytes (two ASCII hex characters per byte), which encode two fields: a command code that specifies the action the client should take and a sequence number. The table below describes these two encoded data fields.

Field	Data type	Description
cmd	Byte	A command code, instructing the client on what to do after receiving the SMS.  Legal values are 0 (immediately re-login) and 1 (immediately re-synchronize all application feeds using getData).
seq	32-bit integer	The sequence number of this message. This value is encoded using network byte ordering (big endian). Over the course of a session, the value of this property will increment every time a new signal is sent. The value of the sequence number must be monotonically increasing for the duration of the session. If client receives a sequence number that is not greater than the previous value, the client should ignore the SMS.

For example, a message payload containing *010000A96B* contains a command a resynchronize command (*0x01*), and a sequence number of *43368* (*0x0000A96B*).

## 8 Binary transport format

Feed item collections are encoded in a binary message format before transport over the network. A binary message update is defined as a sequence of records, drawn from the following three record types:

**BeginFeed**—Indicates the beginning of a feed item collection associated with a specific channel. All ItemData and PropData records between this BeginFeed record and the next belong to the same feed item collection.

**ItemData**—Indicates the beginning of a feed item. All PropData records, if any, between this and the next ItemData record belong to the same feed item.

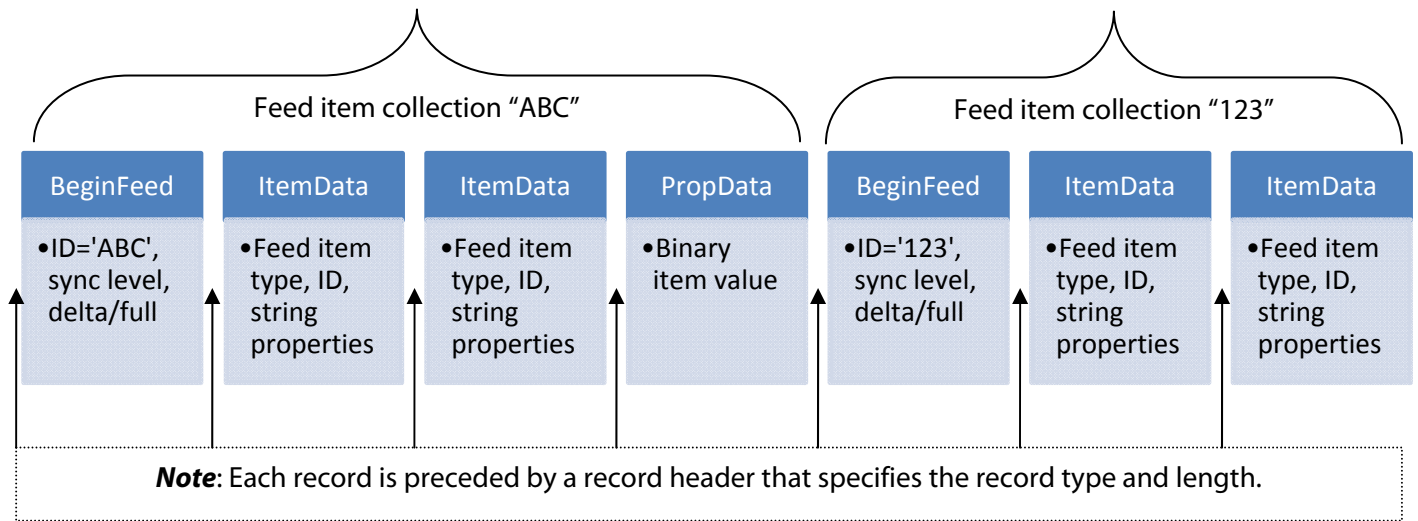
**PropData**—Contains the data for a single feed item property and belongs to the feed item specified by the nearest previous ItemData record.

Each of these records is preceded by a record header that specifies the record type that follows it (BeginFeed, ItemData, or PropData), and the length of that record in bytes (excluding the size of the header).

Each record encodes one or more serialized properties. For example, a BeginFeed record encodes the ID of the channel represented by the record (a variable-length string), followed by the channel's sync level (four bytes), and a

one-byte property that indicates if the collection is a delta or full update. Upon receipt of a binary update, a client or server must de-serialize the binary record sequence.

The following diagram illustrates the basic structure of a binary update record sequence.



## 8.1 Data types

These properties contained by each record are composed of the following data types:

- **Byte**—8-bit unsigned
- **Integer**—32-bit unsigned
- **Timestamp**—32-bit unsigned, containing the number of seconds since 00:00:00, January 1, 1970, UTC.
- **String**—Variable length character sequence composed of a 16-bit unsigned number that specifies the *length* of the string in bytes, followed by *length* bytes. Null-termination of strings is *not* required.
- **Byte-sequence**—Variable length byte sequence composed of a 32-bit unsigned number specifying the *length*, followed by *length* bytes.

Note the following:

- Byte (8-bit) alignment is guaranteed.
- All multi-byte values are serialized in network order.
- Strings are encoded as a sequence of bytes, with no reference to the character encoding. Because it is assumed that each client will support a default character encoding (which the server has knowledge of), explicitly specifying the character encoding is not necessary.
- Strings and byte-sequences are both variable length properties, whose lengths are encoded as properties within the encoded packet. Developers should take care when writing message parsers to handle these data types to avoid vulnerability to buffer-overflow exploits.

## 8.2 Record header format

All records begin with a five-byte record header that specifies the type and length of the record being represented and

The table below lists and describes the record header's fields and data types.

Data Type	Description
Byte	The type of record (BeginFeed, ItemData, or PropData) being represented according to the following mapping:  1: BeginFeed record 2: ItemData record 3: PropData record
Integer	The number of bytes that make up the record, excluding this header.

## 8.3 BeginFeed record format

This BeginFeed record indicates the beginning of a feed item collection. All ItemData and PropData records between this and the next BeginFeed record must belong to the same feed item collection.

The following table lists the fields and their data types encoded by each BeginFeed record, in the order they appear the hexadecimal byte sequence.

Data Type	Description
String	The ID of the channel to which the feed item collection belongs.
Integer	The new synchronization level for the channel's feed item collection.
Byte	Indicates if the feed item collection is to be interpreted as a full or delta update. Two possible values for this field are: <ul style="list-style-type: none"><li>• 1: Indicates that the collection is a delta update.</li><li>• 0: Indicates that the collection is a full update.</li></ul>

## 8.4 ItemData record format

The ItemData record indicates the beginning of a single feed item. Any PropData records between this and the next ItemData record belong to the same feed item.

Feed item properties are encoded differently depending on whether the type of the value is a string property or a binary asset property, such as an image or SWF file.

If the value is a string-property, then the property is specified directly within this ItemData record. After the item's last required field (the item's ID), there is a pair of fields for every string property (corresponding to the property name and value). There can be zero to  $N$  of these field pairs.

If the value is a binary asset-property, this ItemData record must be followed immediately by a contiguous sequence of zero or more PropData records that contain values for all asset-properties of this ItemData record.

The table below lists the fields that are encoded by each ItemData record in the order they appear in the update message:

Type	Description
Integer	The sync level at which this item took on the values expressed in this message
Integer	A value of 1 indicates that this item represents a deletion record; 0 indicates that it is not a deletion record.
String	Specifies this feed item's <b>type</b> string.
String	Specifies the <b>id</b> of the item
Integer	The number of bytes used to record all the text properties in this message.
String	The name of string <i>property 1</i> .
String	The value of string <i>property 1</i>
String	The name of string <i>property N</i>
String	The value of string <i>property N</i>

## 8.5 PropData record format

A PropData record contains the data for a single property belonging to the feed item specified by the nearest previous ItemData record.

Type	Description
String	The name of the property. A string with more than one character must be specified if a property exists.
String	The MIME type of this property, formatted according to the "type/subtype" convention. A valid value for <i>type</i> must be specified if a property exists.  <i>Note:</i> Other parameters that are part of the MIME type specification are not allowed.
Byte-sequence	The value of this property. The byte sequence is one or more bytes long and extends to the end of the record. All byte values are valid in this sequence and no values are escaped. The byte sequence starts out with a 4-byte length indicator.

## 8.6 Sample update message

In this section we'll deconstruct a sample binary transport update message captured with a network monitoring tool. The transaction in question was new subscription request from the client. As discussed previously (see [Add subscription](#)), a new subscription request takes the form of a SetData call from the client to the server containing an update to the Subscription channel's feed item collection. The update must be a `feedapp-sub` item whose ID is set to the ID of the channel being subscribed to (12c0 in this example) and whose string properties specify the initial subscription properties.

Below is a representation of this update in a human-readable format:

```
FeedItemCollection (id:'ffff',synclevel=3184141751, delta=true)
  FeedItem(type:'feedapp-sub', id:'12c0')
    Properties:
      trial=false
      activated=true
      state=deliver
      period=P
```

And below is the same feed item collection encoded into the binary update format.

```
0000 01 00 00 00 0B 00 04 66 66 66 66 c2 f2 49 e7 01
0010 02 00 00 00 59 c2 f2 49 e7 00 00 00 00 0b 66
0020 65 65 64 61 70 70 2d 73 75 62 00 04 31 32 63 30
0030 00 00 00 3a 00 09 61 63 74 69 76 61 74 65 64 00
0040 04 74 72 75 65 00 05 74 72 69 61 6c 00 05 66 61
0050 6c 73 65 00 06 70 65 72 69 6f 64 00 01 50 00 05
0060 73 74 61 74 65 00 07 64 65 6c 69 76 65 72
```

The table below deconstructs this sequence of bytes as they appear above in left-to-right “reading order”.

<i>Raw byte sequence</i>	<i>Interpretation</i>
01 00 00 00 0b	Record header; first byte indicates that the following record is a BeginFeed record (0x01) that is 11 (0x0000000B) bytes long.
00 04 66 66 66 66 c2 f2 49 e7 01	BeginFeed record; the ID of the channel being updated is a string composed of 4-bytes (0x0004) represented by the byte sequence 0x66666666, or “FFFF”; the new sync level for this channel is 3184141751 (0xBDCA25B7); lastly, the update is a delta update (0x01).
02 00 00 00 59	Record header; next record is an ItemData record that is 73 bytes long (0x00000049).
c2 f2 49 e7 00 00 00 00 00 0b 66 65 65 64 61 70 70 2d 73 75 62 00 04 31 32 63 30 00 00 00 3a 00 09 61 63 74 69 76 61 74 65 64 00 04 74 72 75 65 00 00 00 3a	ItemData record; the record’s timestamp (in seconds since epoch) is 3270658535 (0xC2f249e7), which represents Tuesday, 22 Aug 2073 20:15:35 GMT; the item is not a delete record (0x00000000); the feed item type is <code>feedapp-sub</code> , represented by the 11 byte hex string 0x666565646170702d737562; the item ID is a string that is four bytes in length (0x0004) represented by the hex string 0x31326330, which decodes to “12c0”.  The ItemData record contains one or more string properties (name/value pairs) whose combined length is 58 bytes (0x0000003A). These string properties are encoded immediately following the ItemData record.
00 09 61 63 74 69 76 61 74 65 64 00 04 74 72 75 65	String property; the name of the first feed item property is the 9 byte (0x0009) string named <code>activated</code> (0x616374697661746564 in hex notation); its value is provided by the 4-byte (0x0004) hex string that follows, 0x74727565, which decodes to <code>true</code> .
00 05 74 72 69 61 6c 00 05 66 61 6c 73 65	String property; the name of the next feed item property is a 5-byte (0x0005) string named <code>trial</code> (hex value 0x747269616c, whose value is provided by the 5-byte (0x0005) hex string that follows,

	0x74727565, which decodes to <i>false</i> .
00 06 70 65 72 69 6f 64 00 01 50	String property; the name of the next feed item property is a 6-byte (0x0006) string named <i>period</i> (0x706572696f64), whose value is provided by the single-byte (0x0001) hex string that follows, 0x50, which decodes to <i>P</i> .
00 05 73 74 61 74 65 00 07 64 65 6c 69 76 65 72 00 07 64 65 6c 69 76 65 72	String property; the name of the next feed item property is a 5-byte (0x0005) string named <i>state</i> (0x7374617465), whose value is provided by the 7-byte (0x0007) hex string 0x50 that decodes to <i>deliver</i> .

## 9 Use cases and scenario diagrams

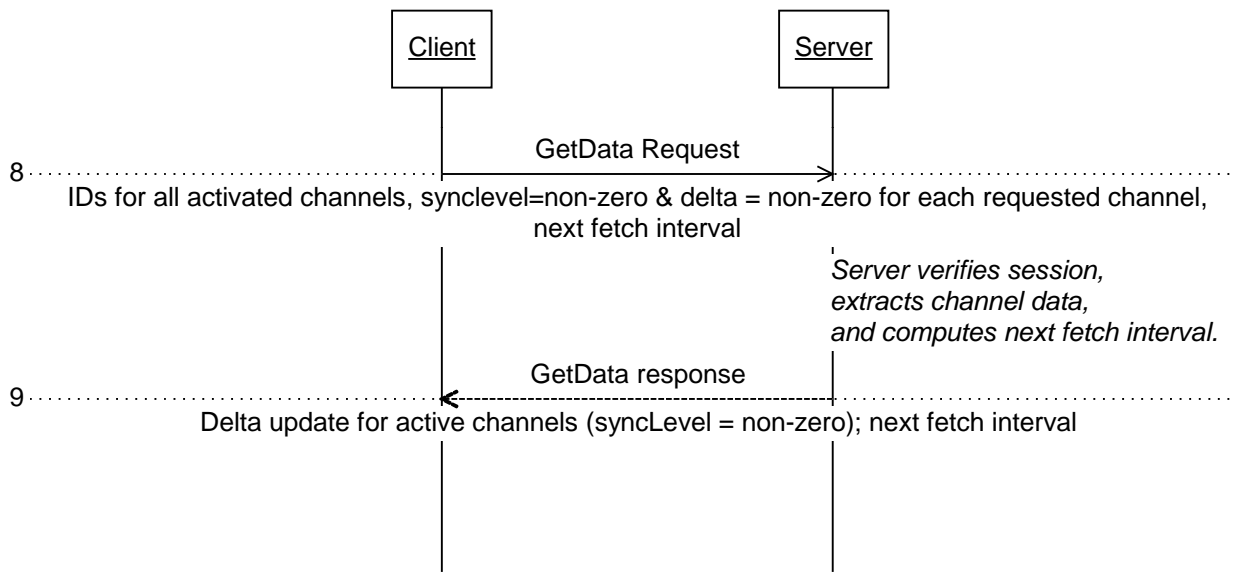
### 9.1 Initial login

In this scenario, the client is starting for the first time on the device and has no session or device profile information. In this case the client must transmit its device capabilities to the server in the login request, along with its authentication credentials. After a successful login a client must request the Subscription feed from the server.

The following sequence diagram illustrates this process in detail:



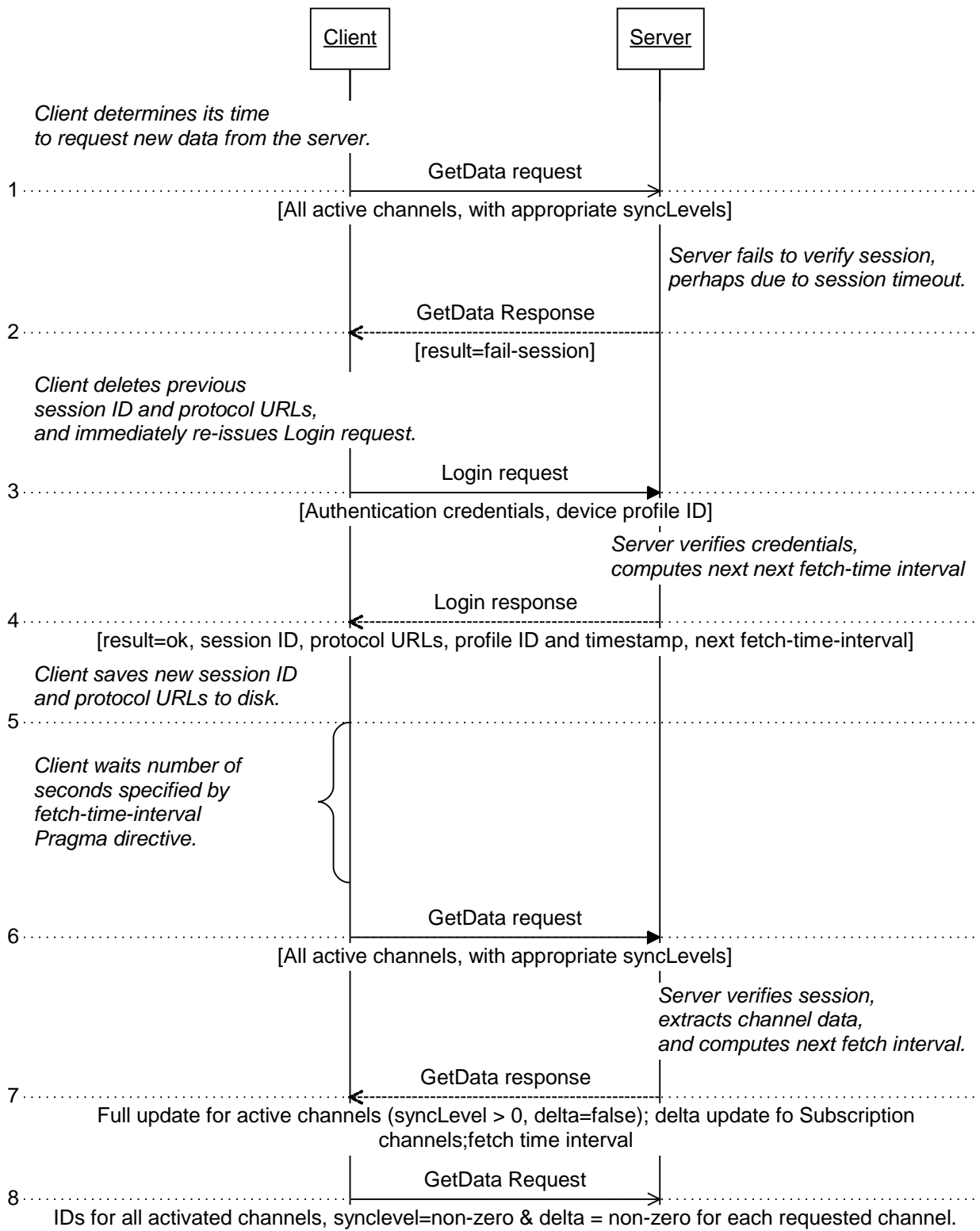
## First-time login



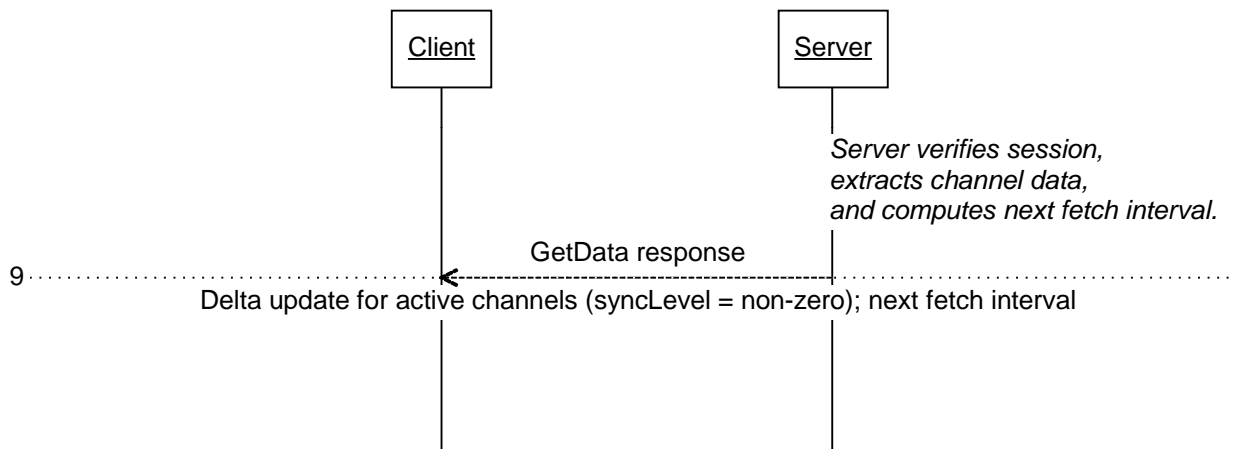
### 9.2 Login successful, no session

In this use case, the client realizes it is time to synchronize channel data with the server (based on the last `flashcast-fetch-time` Pragma directive reported by the server), but the client's session with the server is no longer valid for some reason. The following sequence diagram illustrates this process in detail.

## Login, invalid session



## Login, invalid session

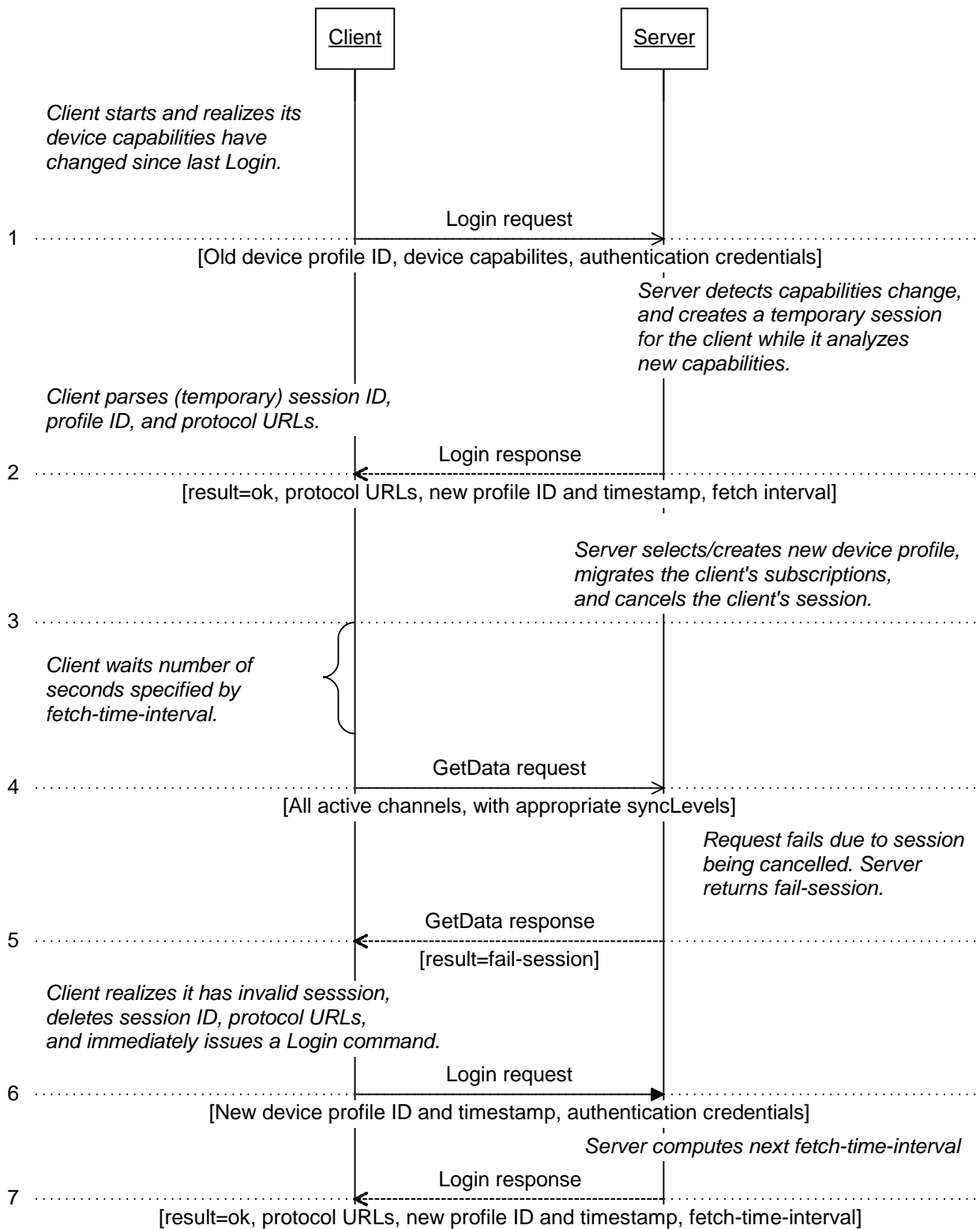


### 9.3 Login successful, capabilities change

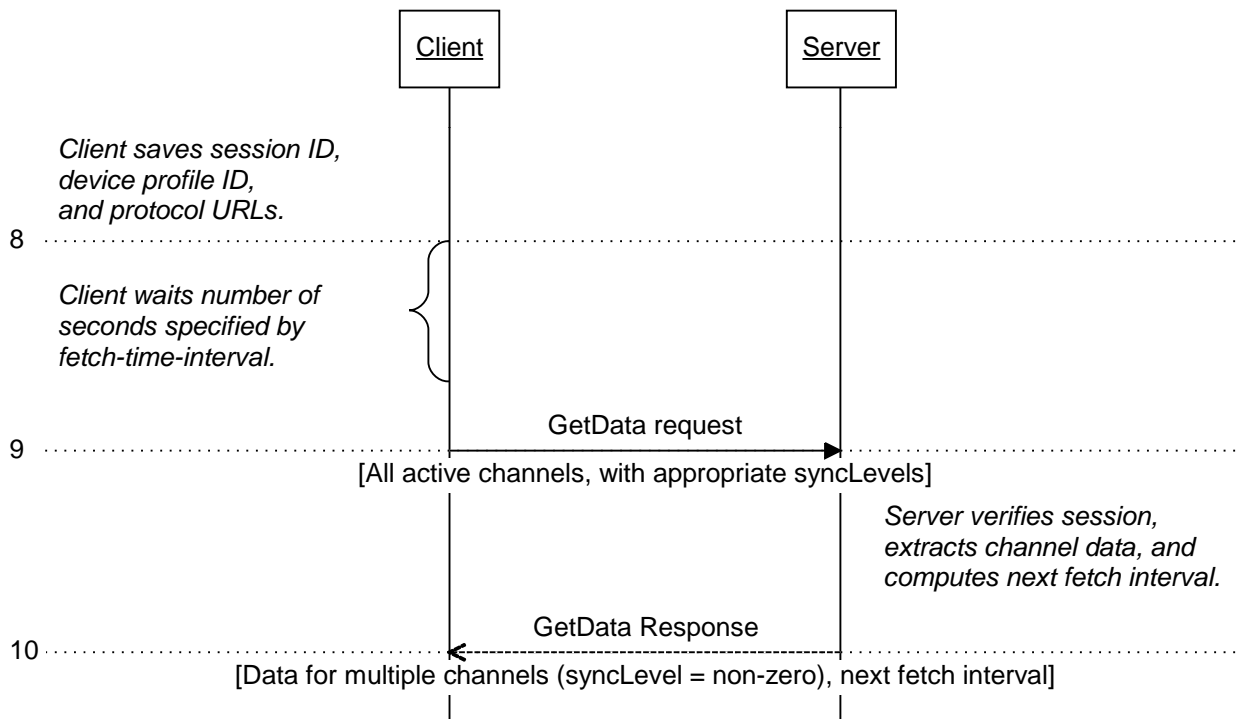
In this scenario the client starts and realizes that its capabilities have changed since the last time it logged in. The client must retransmit its device capabilities to the server. The server then must select or create a new device profile based on the new device capabilities, and migrate the user's subscriptions according to the client's new device capabilities. Because this process may take some time, the server **MUST** ignore any GetData requests from the client while the profile upgrade is in process. Otherwise, the channel content the server selects may not be appropriate for the device.

The following sequence diagram illustrates this process in detail.

## Login successful, capabilities change



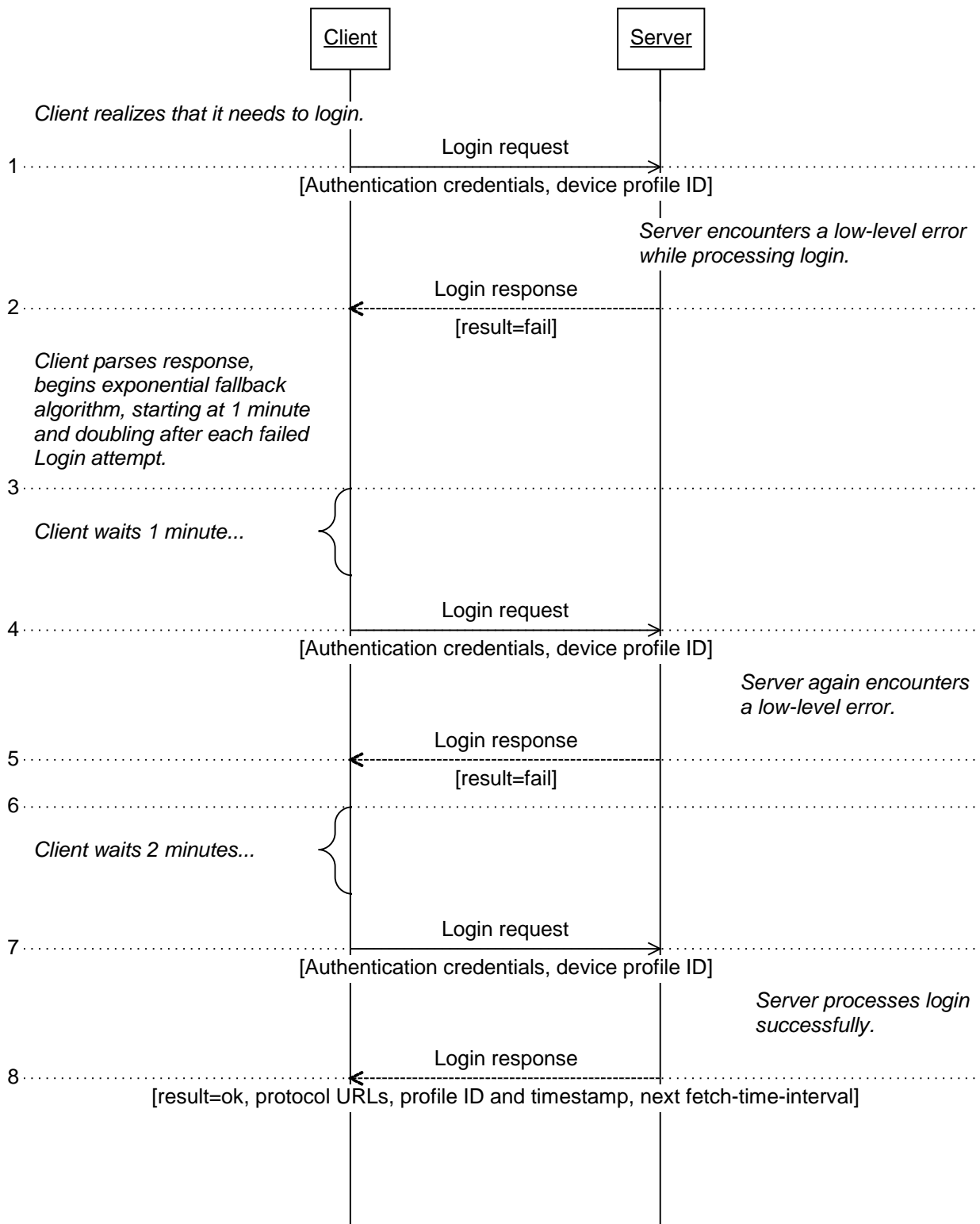
## Login successful, capabilities change



### 9.4 Login failure, fail (generic)

In this scenario, the client attempts to login but the server encounters a low-level error—for example, a generic network error occurred while the server was reading the request. Once the client parses the server response and determines that a generic login error has occurred, it begins an exponential fallback algorithm, attempting to login after 1 minute, then doubling the interval (2 minutes, 4 minutes, etc.) after each failed attempt. The following sequence diagram illustrates this process in detail.

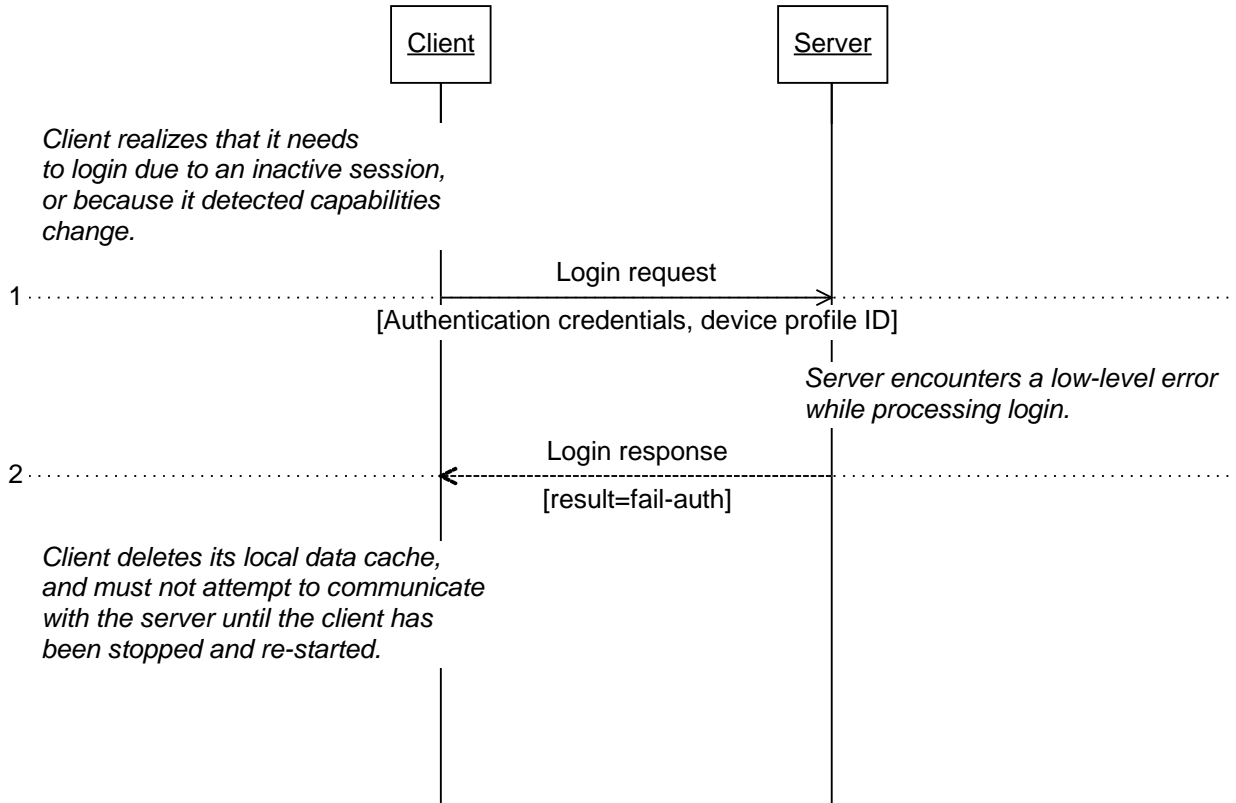
## Login failure, general error



### 9.5 Login failure, fail-auth

In this scenario, the client realizes it needs to login with authentication credentials, perhaps because it doesn't have an active session, or has detected a device capabilities change. The server receives the login request but is unable to validate the specified credentials, so it returns a `fail-auth` message in its response. In this case, the client must not attempt to communicate with the server again, unless the client application is exited and restarted. The following sequence diagram illustrates this process in detail

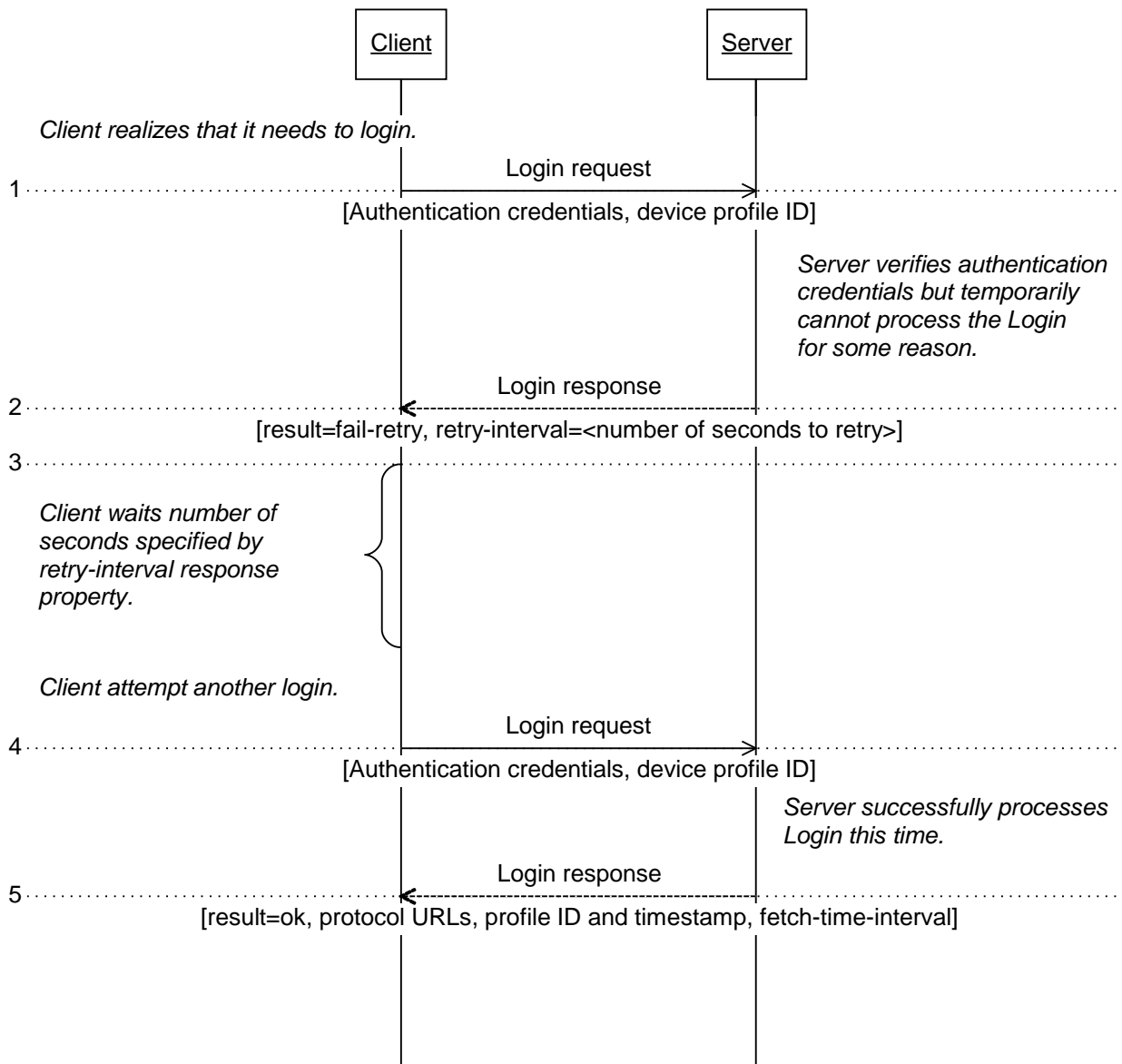
### Login failure, authorization error



### 9.6 Login failure, fail-retry

In this login scenario, the server verifies the client's authentication credentials, but is temporarily unable to process the login for some reason. Perhaps the server has run out of memory, is in the process of initializing, or has encountered some other error. In this case, the server returns a `fail-retry` message along with the computed fetch interval the client should wait before attempting another login command. The following sequence diagram illustrates this process in detail.

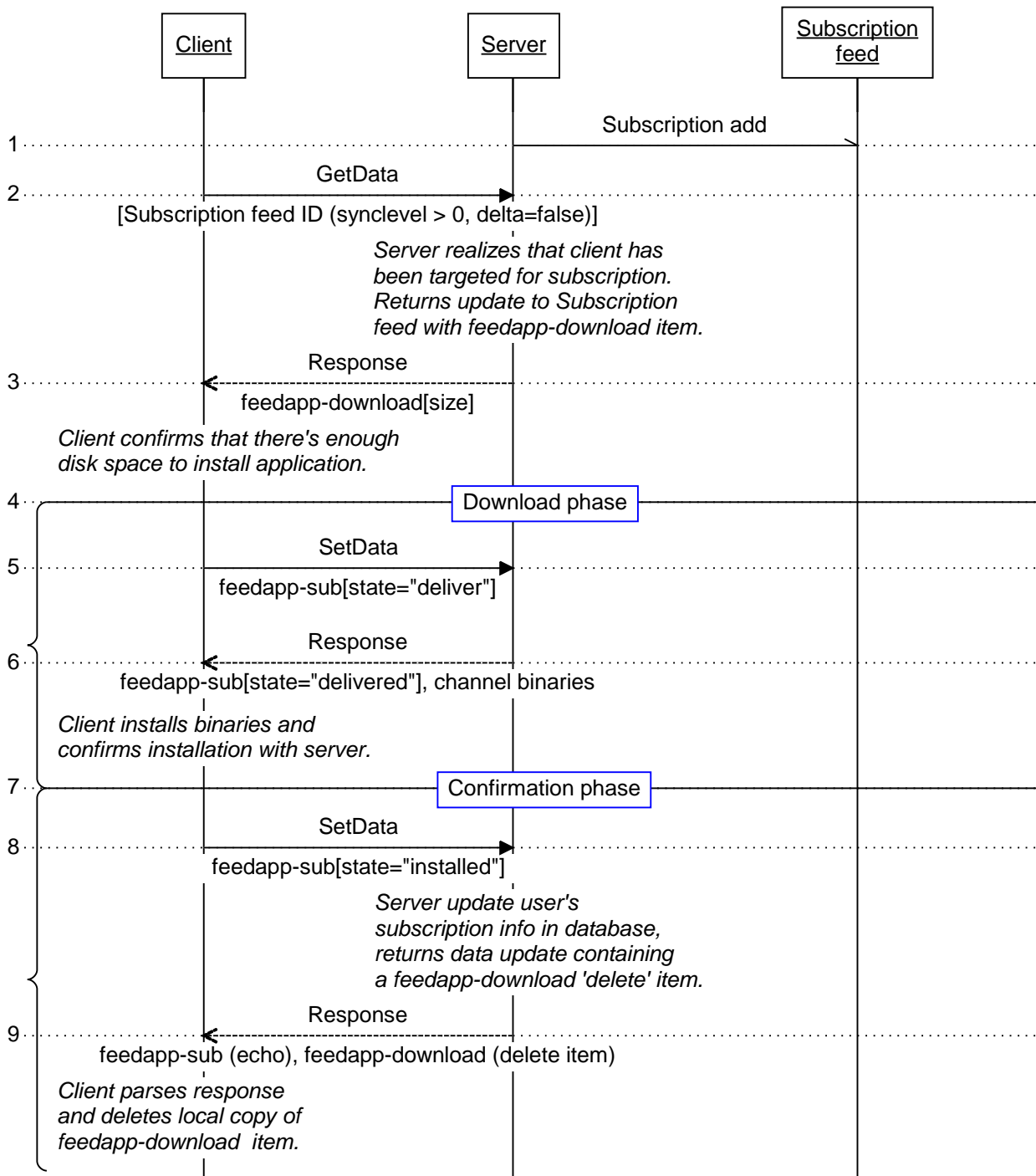
## Login failure, retry



## 9.7 Server-initiated subscription add

In this subscription scenario, a device has been targeted for a subscription on the server. As described previously (see [Subscription targeting](#)), a server can send a client a `feedapp-download` item in an update for the Subscription channel's feed item collection. The client interprets this as a request to initiate a new subscription request with a `SetData` command. The following sequence diagram illustrates this process in detail.

## Server-initiated subscription request



## 9.8 Client-initiated new subscription

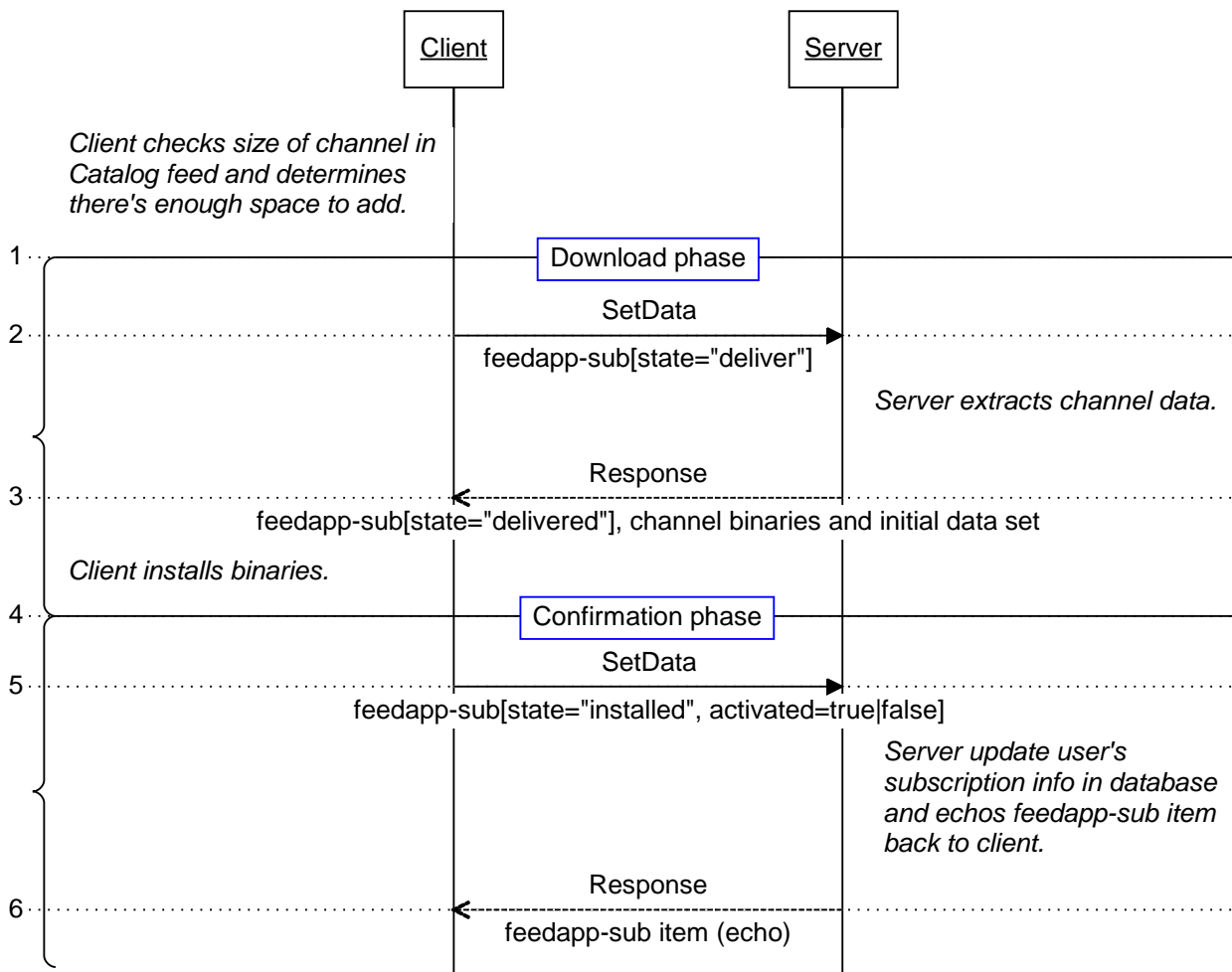
In this case, a client makes a new subscription request to the server, specifically one that has not been “targeted” for download. Before a client makes any new subscription requests, it should ensure that it has sufficient storage space to install the application’s binaries by checking the `feedSize` property in the channel’s corresponding `feedapp-info` entry in the Catalog channel’s feed item collection.

To subscribe to a new channel, the client sends the server a `feedapp-sub` item representing the desired the channel subscription, with the feed item's `state` property set to `deliver`. The server responds with the channel's binary assets and its initial data set. In the same response message, the server also echoes the `feedapp-sub` item with its `state` property set to `delivered`, by including it in the subsequent response.

The client then sends the same `feedapp-sub` item back to the server with its `state` property set to `installed`, letting the server know that the assets were successfully installed and the channel can being operation. The client also sets the subscription feed item's `activate` property to `true` or `false`, indicating whether the subscription should be started in activated or deactivated mode. Finally, and assuming no error cases were encountered, the server echoes the subscription item with its `state` property set to `subscribed`. The client- and server-side representations of the subscription are now in sync; a non-transitional state has been reached.

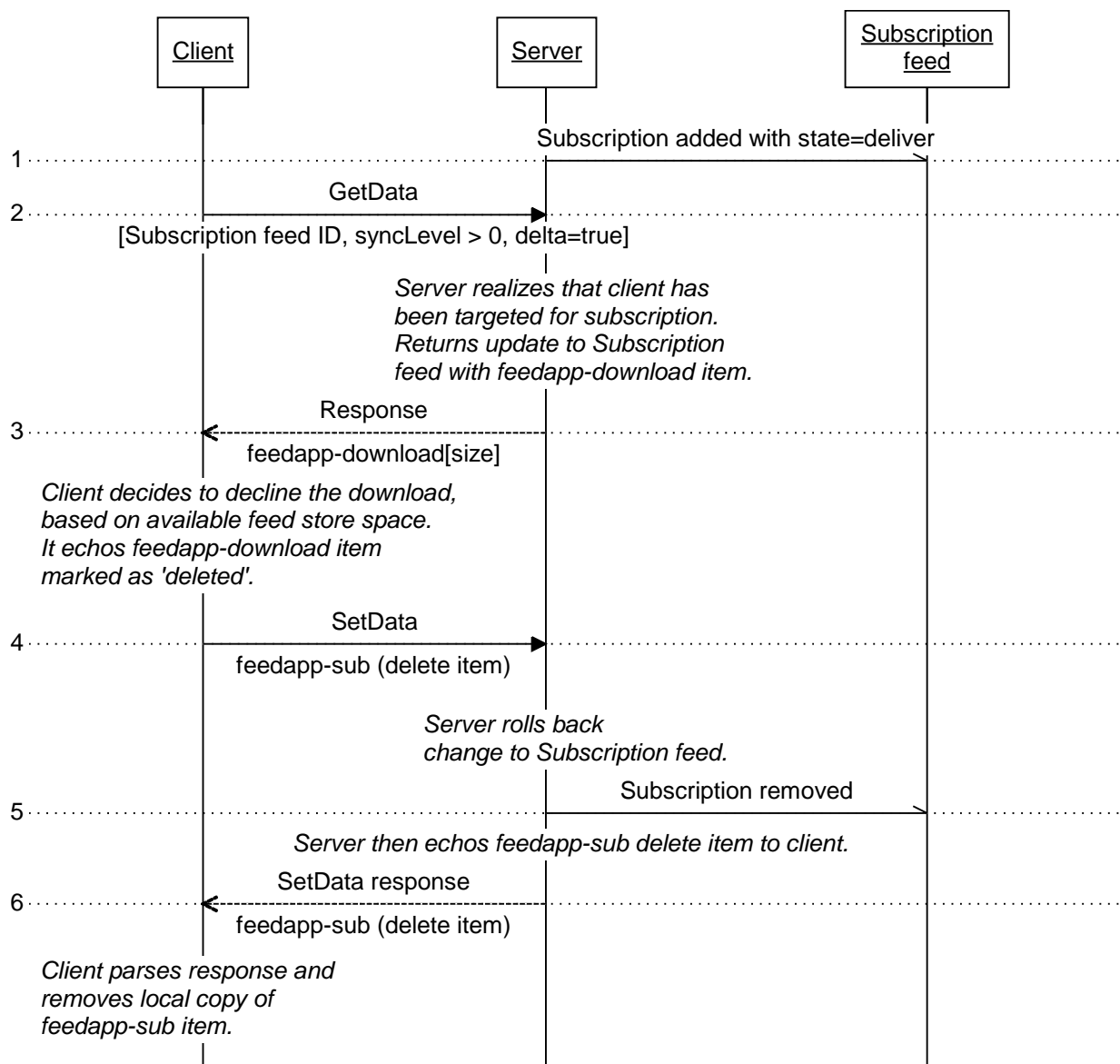
The following sequence diagram illustrates this process in detail. Also see [Channel download and confirmation](#).

### Client-initiated subscription request



## 9.9 Server-initiated subscription add, client declines download

In this scenario, a server has targeted a client to install a new channel, but the client declines the request because it doesn't have sufficient space to install the binaries. The following sequence diagram illustrates this process in detail.



## 10 Appendix A: Feed item schemas

This section provides the feed item schemas used by the system channels (Subscription and Catalog), and for the “well-known” feed items used to represent standard channel assets, such as the channel’s root content, filters and settings data.

Feed items fields are composed of the following data types:

- **Byte**—8-bit unsigned
- **Integer**—32-bit unsigned

- **Timestamp**—32-bit unsigned, containing the number of seconds since 00:00:00, January 1, 1970, UTC.
- **String**—Variable length character sequence composed of a 16-bit unsigned number that specifies the *length* of the string in bytes, followed by *length* bytes.
- **Byte-sequence**—Variable length byte sequence composed of a 32-bit unsigned number specifying the *length*, followed by *length* bytes.

## 10.1 Subscription feed schema

The Subscription feed’s schema is made up of the following feed item types, each containing a different bundle of subscription state information:

- `feedapp-sub` items contain information about the user’s current channel subscriptions.
- `feedapp-info` items contain public information about the user’s subscribed channels as a whole, rather than about the individual subscription the user holds.
- `feedapp-download` items are delivered by the server to “target” a device with a new subscription (see [Subscription targeting](#)). It combines the properties `feedapp-sub` and `feedapp-info` items.
- `feedapp-preview` items contain a short preview clip of the associated `feedapp-info` item.

Each channel subscription is represented by a combination of one `feedapp-info` item and one `feedapp-sub` item. Both feed items must share the same item ID, which is equal to the ID of the channel being represented.

### 10.1.1 Channel information (feedapp-info) items

Each `feedapp-info` item in the Subscription feed describes the public properties of a channel the user is subscribed to. It contain properties pertaining describe to the channel as a whole, rather than to the individual subscription this user holds.

The ID of each `feedapp-info` item must be set to the ID of the channel that it is representing. The properties that each `feedapp-info` item may contain may exist are:

Property name	Data Type	Description
<code>name</code>	String	The name of the channel presented to users.
<code>desc</code>	String	A short description of the channel.
<code>longDesc</code>	String	A long description of the channel.
<code>acl</code>	String	Represents the ACL (Access Control List) of the channel. A value of <code>u</code> indicates that the channel can be unsubscribed, while an empty string means it cannot.
<code>type</code>	Integer	The channel’s subscription type. Legal values are 0 (free), 1 (premium), or 2 (reserved by client).
<code>access</code>	String	The access privileges of the channel encoded as a hexadecimal bitmask. For more information about the content and formatting of this field, see <a href="#">Channel permissions</a> .
<code>accx</code>	String	The extended access privileges of the channel, encoded as a hexadecimal bitmask (see <b>access</b> , above).
<code>accxx</code>	String	The OEM extended access privileges of the channel encoded as a

		hexadecimal bitmask (see access, above).														
period	String	<p>The channel's subscription period indicated by a subscription period indicator (for example, "D" for day, "W" for week, and so forth) concatenated with an integer that acts as a period multiplier. For example, a subscription period of three months would be represented with a period value of "M3".</p> <p>The table below lists the valid subscription period indicators:</p> <table border="1"> <thead> <tr> <th>Period indicator</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>D</td> <td>Day</td> </tr> <tr> <td>W</td> <td>Week</td> </tr> <tr> <td>M</td> <td>Month</td> </tr> <tr> <td>Q</td> <td>Quarter</td> </tr> <tr> <td>Y</td> <td>Year</td> </tr> <tr> <td>P</td> <td>Perpetual</td> </tr> </tbody> </table> <p><b>Note:</b> If the subscription is perpetual (P) then any integer multiplier should be ignored.</p>	Period indicator	Meaning	D	Day	W	Week	M	Month	Q	Quarter	Y	Year	P	Perpetual
Period indicator	Meaning															
D	Day															
W	Week															
M	Month															
Q	Quarter															
Y	Year															
P	Perpetual															
price	String	The price of a subscription to this channel, in the user's default currency. <b>Default:</b> 0.														
trialDays	Integer	The number of days of trial use available to this user. <b>Default:</b> 0.														
begin	timestamp	The time at which the subscription begins, for channels that have a fixed lifespan. <b>Default:</b> none														
end	timestamp	The time at which the channel ends, for feeds that have a fixed lifespan. All subscriptions to this channel will end on or before this date. <b>Default:</b> none.														
displayOrder	integer	A number specifying the order that this channel should be displayed relative to other channels within the Now Playing channel.														
contentType	String	The type of channel contained by the feed item. Valid values are channel, homescreen, ui, subscription, nowPlaying, or discovery. For more information about content types, see <a href="#">Channel namespaces and content types</a> .														
ns	String	The namespace that this channel belongs to. For more information about content types, see <a href="#">Channel namespaces and content types</a> .														
baseURL	String	The URL to which direct HTTP access is allowed for this channel.														
feedHeaderSize	Integer	Specifies the initial size, in bytes, of the header section within the channel's data file on the client. If this property is not present, the client should use a default value of 5120 bytes.														
feedSize	Integer	<p>Specifies the size, in bytes, of the data section of the channel's feed. If not present, the client should use a default value of 104,448 bytes.</p> <p>Notice that the channel's data file will be larger than this value since it also contains the bytes for the header plus some free space that is used for garbage collection (twice the header size).</p>														

<code>scratchpadHeaderSize</code>	Integer	Determines the initial size, in bytes, of the header section within the scratchpad file. If this property is not present, the client should use a default value of 256.
<code>scratchpadSize</code>	Integer	Determines the size, in bytes, of the data section of the channel's data file. If not present, the client should use a default value of 1024.  If present, the actual the scratchpad data will be larger than this value since it also contains the bytes for the header plus some free space that is used for garbage collection (twice the header size).

### 10.1.2 Channel subscription (feedapp-sub) items

Each `feedapp-sub` item describes the state of the user's subscription to the associated channel.

Note that when a client sends a `feedapp-sub` item update in a `SetData` call the properties allowed in the request are a subset of those shown below. Namely, only `activated`, `trial`, and `state` should be included in `SetData` calls to server. For more details about adding subscriptions, see [Add subscription](#).

Property	Data type	Description
<code>activated</code>	Boolean	Determines whether the subscription is actively receiving data updates from the server. When set to <code>true</code> , <code>GetData</code> and <code>SetData</code> requests are accepted. When set to <code>false</code> , <code>GetData</code> and <code>SetData</code> requests result in an error.  Default: <code>true</code> .
<code>trial</code>	Boolean	Signifies if the subscription is a trial. If <code>trial</code> is set to <code>true</code> , the feed item's <code>begin</code> and <code>end</code> refer to the trial period.  Default: <code>false</code> .
<code>begin</code>	Timestamp	The subscription's start date.  Default: none.
<code>end</code>	Timestamp	The subscription's end date.  Default: none.
<code>subType</code>	String	The subscription's type. The only value currently defined for this property is <code>i</code> (the letter "i").
<code>state</code>	String	Specifies the state of the subscription. Can be one of the following string values (see <a href="#">Subscription states</a> for descriptions of each state): <ul style="list-style-type: none"> <li>• <code>subscribed</code></li> <li>• <code>unsubscribed</code></li> <li>• <code>active</code></li> </ul>

		<ul style="list-style-type: none"> <li>• expired</li> <li>• cancelled</li> <li>• deliver</li> <li>• delivered</li> <li>• installed</li> </ul> <p>Default: None.</p>
clientError	String	Specifies the latest client error code reported by the client for this channel. For a list of error codes, see <a href="#">Appendix B: Client error codes</a> .

### 10.1.3 Subscription download (feedapp-download) items

A client's subscription feed may contain one or more `feedapp-download` items used to "target" devices with a new subscription (see [Subscription targeting](#)). The server uses these items to inform the client that it has been requested to download a specific channel. This could be a new subscription or update to existing subscription.

Feed items of this type combine the properties of both `feedapp-sub` and `feedapp-info` feed items. The two properties that exist in both `feedapp-sub` and `feedapp-info`—namely, the `begin` and `end` subscription properties—are taken from `feedapp-sub` and not from `feedapp-info`.

Property name	Data Type	Description
name	String	The name of the channel presented to users.
desc	String	A short description of the channel.
longDesc	String	A long description of the channel.
acl	String	Represents the ACL (Access Control List) of the channel. A value of <code>u</code> indicates that the channel can be unsubscribed, while an empty string ("") means it cannot.
type	Integer	The channel's subscription type. Legal values are 0 (free), 1 (premium), or 2 (reserved by client).
access	String	The access privileges of the channel encoded as a hexadecimal bitmask. For more information about the content and formatting of this field, see <a href="#">Channel permissions</a> .
accx	String	The extended access privileges of the channel, encoded as a hexadecimal bitmask (see <b>access</b> , above).
accxx	String	The OEM extended access privileges of the channel encoded as a hexadecimal bitmask (see <code>access</code> , above).
period	String	The channel's subscription period indicated by a subscription period indicator (for example, "D" for day, "W" for week, and so forth) concatenated with an integer that acts as a period multiplier. For example, a subscription period of three months would be represented with a <code>period</code> value of " <b>M3</b> ".  The table below lists the valid subscription period indicators:

		<b>Period indicator</b>	<b>Meaning</b>
		D	Day
		W	Week
		M	Month
		Q	Quarter
		Y	Year
		P	Perpetual
		<b>Note:</b> If the subscription is perpetual (P) then any integer multiplier should be ignored.	
price	String	The price of a subscription to this channel, in the user's default currency. <b>Default:</b> 0.	
trialDays	Integer	The number of days of trial use available to this user. <b>Default:</b> 0.	
begin	timestamp	The time at which the subscription begins, for channels that have a fixed lifespan. <b>Default:</b> none	
end	timestamp	The time at which the channel ends, for feeds that have a fixed lifespan. All subscriptions to this channel will end on or before this date. <b>Default:</b> none.	
displayOrder	integer	A number specifying the order that this channel should be displayed relative to other channels within a channel carousel or list.	
contentType	String	The type of channel contained by the feed item. For more information about content types, see <a href="#">Channel namespaces and content types</a> .	
ns	String	The namespace that this channel belongs to. For more information about content types, see <a href="#">Channel namespaces and content types</a> .	
baseURL	String	The URL to which direct HTTP access is allowed for this channel.	
feedHeaderSize	Integer	Specifies the initial size, in bytes, of the header section within the channel's data file. If this property is not present, the client should use a default value of 5120 bytes.	
feedSize	Integer	Specifies the size, in bytes, of the data section of the channel's feed. If not present, the client should use a default value of 104,448.  Notice that the channel's data file will be larger than this value since it also contains the bytes for the header plus some free space that is used for garbage collection (twice the header size).	
scratchpadHeaderSize	Integer	Specifies the initial size, in bytes, of the header section within the scratchpad file. If this property is not present, the client should use a default value of 256.	
scratchpadSize	Integer	Specifies the size, in bytes, of the data section of the channel's data file. If not present, the client should use a default value of 1024.  If present, the actual the scratchpad data will be larger than this	

		value since it also contains the bytes for the header plus some free space that is used for garbage collection (twice the header size).
<code>activated</code>	Boolean	Determines whether the subscription is actively receiving data updates from the server. When set to <code>true</code> , <code>GetData</code> and <code>SetData</code> requests are accepted. When set to <code>false</code> , <code>GetData</code> and <code>SetData</code> requests result in an error.  Default: <code>true</code> .
<code>trial</code>	Boolean	Signifies if the subscription is a trial. If <code>trial</code> is set to <code>true</code> , the feed item's <code>begin</code> and <code>end</code> refer to the trial period.  Default: <code>false</code> .
<code>subType</code>	String	Indicates if the channel was subscribed to individually <code>i</code> or as part of a channel-package <code>f</code> .
<code>state</code>	String	Specifies the state of the subscription. Can be one of the following string values (see <a href="#">Subscription states</a> for descriptions of each state): <ul style="list-style-type: none"> <li>• <code>subscribed</code></li> <li>• <code>unsubscribed</code></li> <li>• <code>active</code></li> <li>• <code>expired</code></li> <li>• <code>cancelled</code></li> <li>• <code>deliver</code></li> <li>• <code>delivered</code></li> <li>• <code>installed</code></li> </ul> Default none.
<code>clientError</code>	String	Specifies the latest client error code reported by the client for this channel. For a list of error codes, see <a href="#">Appendix B: Client error codes</a> .

### 10.1.4 Channel preview (feedapp-preview) items

Feed items of type `feedapp-preview` provide the client with a short preview clip of the channel represented by the `feedapp-info` item with the same item ID. Each preview item contains either a binary-asset property name `clip` that contains the preview clip, or a string property named `url` that specifies the URL where the client can retrieve the preview clip. It will contain of these properties, never both.

Property name	Data Type	Description
<code>clip</code>	Binary asset	The preview clip in binary form (typically SWF).
<code>url</code>	String	The URL where the preview resides.

## 10.2 Catalog feed schema

The Catalog channel's feed item collection describes all the available categories, and all the channels that belong to each category. For each category, the Catalog feed contains one `category` feed item whose ID is the category's logical name ("news", for example), and whose properties provide additional details about the category, such as the category name to display to users. For each category feed item, there must exist one or more feed items whose type is identical to the corresponding category item's ID. These "channel" feed items describe the public properties of each channel belonging to that

### 10.2.1 Category item (category) schema

Within the Catalog channel's feed item collection, categories are represented as feed items of type `category`. The ID of each `category` feed item is its logical category name. Each category item contains the following properties:

<i>Property</i>	<i>Type</i>	<i>Description</i>
<code>displayName</code>	String	A name used when to display the category name on the client.
<code>displayOrder</code>	Integer	A number specifying the order that this category should be displayed to the user, relative to other categories.

### 10.2.2 Channel item schema

For each `category` feed item defined in the Catalog feed (see Category item (category) schema), the Catalog feed must contain an additional set of feed items whose type string is equal to the ID of the given category. For example, if there exists a `category` feed item whose ID is `games-category`, then there must exist an additional feed item whose type is `games-category`. These so-called *channel* items describe the channels in the specified category. The ID of each channel feed item must be the ID of the channel it represents.

Below is a representation of the Catalog channel's feed item collection described previously.

```
FeedItemCollection(id='<catalog-channel-ID>') {
  FeedItem(type='category', id='games-category')
    <...`category` feed item properties...>
  FeedItem(type='games-category', id='<channel-ID>')
    <...Channel item properties>...
  FeedItem(type='games-category', id='<channel-ID>')
    <...Channel item properties>...
}
```

The properties that each channel feed item may contain are identical to those that belong to `feedapp-info` items in the client's Subscription feed. The only difference is the context in which they appear—either as a channel to which a user is currently subscribed (`feedapp-info` items), or as a channel to which the user may subscribe (channel items in the Catalog feed). For a full list of properties that each channel feed item may have, see Channel information (`feedapp-info`) items.

## 10.3 Filter items (feedapp-pref-select)

Each channel's feed item collection may contain a collection of zero or more `feedapp-pref-select` items. Together, these items constitute the subscriber's filters (see [Filter \(select\) feed items](#)).

Each `feedapp-pref-select` item may contain the following properties:

Field	Type	Description
<code>type</code>	String	Specifies the type of feed item to which the filter applies.
<code>items</code>	String	An optional, space-delimited list of encoded item IDs. These correspond to the collection of item elements in a filter.
<code>props</code>	String	An optional, space-delimited list of encoded property names. These correspond to the collection of prop elements in a filter.

The ID of each `feedapp-pref-select` item must be a concatenation of the associated channel's ID with an integer that appears in no other `feedapp-pref-select` items for the same channel. For example, suppose the feed item collection of a channel with an ID of 'ABC' contained two filters. The following is representation of how these feed items might be identified.

```
FeedItemCollection(id='ABC')
  FeedItem(type='feedapp-pref-select', id='ABC.1')
  FeedItem(type='feedapp-pref-select', id='ABC.2')
```

The space-delimited list of item IDs or property names must have the following characters escaped with their URL-encoded values:

Character	URL-encoded replacement
' (space)	%20
% (percent sign)	%25
:(colon)	%3A

## 10.4 Settings items (feedapp-pref-setting)

Settings are a simple repository of persistent name-value pairs, available for channel-specific use. Each channel may contain one or more named sets, or *bags*, of settings. Each settings bag is encoded within the channel's feed item collection as a `feedapp-pref-setting` item. The ID of each `feedapp-pref-setting` item must be set to the name of the settings bag, which can be any unique string. The item may contain one or more string properties (name/value pairs) that contain the actual settings data used by the channel.

For example, below is a representation of a channel's settings data that defines two settings bags, one with an ID of "scores", and the other with an ID of "options". The `scores` settings bag defines two properties named `highScore` and `saveScores`; the `options` settings bag defines properties named `gameServer`, `volume`, and `updateInterval`.

```
<itemcollection id="555">
  <itemtype name="feedapp-pref-setting">
    <item id="scores">
      <prop name="highScore" value="2342"/>
      <prop name="saveScores" value="1"/>
    </item>
    <item id="options" >
      <prop name="gameServer" value="defaultServer"/>
```

```

        <prop name="volume" value="50"/>
        <prop name="updateInterval" value="2"/>
    </item>
</itemtype>
</itemcollection>

```

## 10.5 Root content items (feedapp-bininfo)

A root content item is the client-side application used to interpret and display the channel's data—typically, this will be a binary object in the SWF format. The root content item is encoded as a `feedapp-bininfo` item within the feed item collection of the channel that it's associated with.

Each channel may have zero or one `feedapp-bininfo` items, each representing a root content item. The ID of the item must be set to the channel's ID, and may contain the following properties:

Property name	Type	Description
<code>hasIcon</code>	Boolean	Specifies if the item contains the icon property (true) or not (false).
<code>icon</code>	Binary asset	The icon image for the channel. May be omitted, in which case <code>hasIcon</code> should be set to <code>false</code> .
<code>fcaVersion</code>	String	A string that contains the version number of the root content item contained within this structure. If <code>fcaVersion</code> is set to 0, the root content is not present (that is, the <code>fca</code> property will be omitted).
<code>fca</code>	Binary asset	The root content asset for the channel. May be omitted, in which case <code>fcaVersion=0</code> .

## 10.6 Localized resource items (feedapp-localeinfo)

Localized resource items can be used to deliver and support preloaded language pack SWF files for a given channel. These items are represented as `feedapp-localeinfo` items within the channel's feed item collection. The ID of each of each item of this type is composed of a locale and an item ID, separated by a period. If there no item ID set then the period should be omitted and only the locale attribute will be present.

For example, suppose that the server returns two localized resource items for US English locales. The IDs of those items might look like the following:

```

feedapp-localeinfo(id=en-US.1)
feedapp-localeinfo(id=en-US.2)

```

Each `feedapp-localeinfo` feed item contains a single binary-asset property that contains the locale-specific SWF.

Property name	Type	Description
<code>resource</code>	Binary asset	The resource SWF for this combination of channel and locale combination.

When a localized resource is removed from a channel’s feed item collection, and the client requests a delta update for that channel, the server must return a delete item representing that deleted resource. The server must not return a delete item when the client requests a full update.

## 10.7 Feed error items (fc-result-info)

If the server encounters an error while processing some channel-specific command, it returns a feed item of type `fc-result-info` in its response to the client. This item is included in the feed item collection belonging to the channel in which the error occurred. For example, if a client attempts to subscribe to a channel but provides an incorrect channel ID in the `SetData` call, the server will return an `fc-result-info` item in the Subscription channel’s feed item collection.

Each `fc-result-info` item may contain the following properties.

Property	Type	Description
<code>result</code>	String	Indicates the error condition.  The possible values of this property are identical to those that the server returns for high-level request errors, as described in <a href="#">Request layer errors</a> . The client should behave identically as described in that section for all the possible values of <code>result</code> .
<code>desc</code>	String	Provides a description of the error.
<code>retry-interval</code>	Integer	Specifies the number of seconds the client should wait before retrying the previous request.
<code>operation</code>	String	Indicates what operation the server was attempting execute when the error occurred. Possible values are as follows: <ul style="list-style-type: none"> <li><code>getData</code>—Server was handling a <code>GetData</code> operation.</li> <li><code>setData</code>—Server was handling a <code>SetData</code> operation.</li> <li><code>subscription</code>—Server was handling a subscription change operation.</li> <li><code>preference</code>—Server was handling a preference change operation.</li> </ul>

## 11 Appendix B: Client error codes

The client must send the server a numeric error code when it encounters certain errors. The client must report any errors it encounters in an update to the Subscription channel’s feed item collection. The following error ranges and code have been established:

- **0:** No error, everything is normal
- **1-99:** This is a critical error that should cause the client to show a permanent error dialog screen for the given channel until the error is resolved. The server should no longer return data to a client when this class of errors has been set for a given channel.

- **101-199:** This is a serious error that should cause the client to show a temporary error dialog screen for the given channel. Typically, this error can only be cleared by updating the root content item (the SWF application). The following errors have been defined in this range:
  - **100:** Detected a corrupt channel SWF.
  - **101:** The channel SWF is too big.
  - **102:** Missing channel SWF. The client's local data cache does not contain a SWF for the current channel.
  - **104:** The client's local data cache is running out of space and cannot create a new feed or increase the size allocated for an existent feed. The client must also send this error code when the feed's size is invalid. The client must deactivate the channel until it is able to free some disk space, or an update arrives from the server with valid feed dimensions.
- **200-299:** Errors in this range are considered recoverable errors without need for immediate administrator intervention to clear the error. There is no specific server reaction to these errors other than recording them to a log file. The following errors have been defined in this range:
  - **200:** Client ActionScript stack overflow.
  - **201:** ActionScript stuck error.
  - **202:** Client has crashed for an unexpected reason.
  - **204:** The Flash player is out of runtime memory.
  - **205:** The client did not receive a next update time from the server, and does not know when to next poll for data.
  - **206:** Header overflow received from server.
  - **208:** The client's local data cache has overflowed with too much data for a given channel. The client must stay in the overflow state until an update arrives from the server that deletes some items from the client's data cache.
  - **209:** No valid system UI found.

## 12 Appendix C: Channel permission bits

The following table lists and describes the channel permissions that are used. The order of the permissions listed in this table (top to bottom) corresponds directly to the order that the corresponding bit flags will appear when encoded for transport. For more information, see [Channel permissions](#).

Permission Name	Description
<b><i>Standard permission bits</i></b>	
System	Full access, no restrictions.
MyData	Channel can access its own feed data.
AllData	Controls access to the feed data of other channels., also if on, banner data access is allowed. <sup>(2)</sup>
NetSys	Controls access to system commands., also if on, banner data access is allowed.
FsCommand	Controls channel's ability to call ActionScript FSCommand.
GetSetting	Controls channel's ability to read settings data.

SetSetting	Controls channel's ability to write settings data.
Scratchpad	Controls access (read/write) of own scratchpad data.
SharedData	When on, banner data access is allowed.
Standalone	Controls whether channel can run in stand-alone mode.
Launch	Controls access to launching native applications through the fscommand(Launch) API provided by the client.
Dialog	Controls access to the Dialog APIs (for example, openDialog, closeDialog).
SwitchHomeScreen	Gives the feed permission to set the currently active homescreen.
NetworkRead	Controls access to ActionScript API for loading network files from an http/https address.
NetworkWrite	Controls access to ActionScript API for sending data to a http/https address
LocalFile	Controls access to the ActionScript APIs for loading files from the local file system (LoadMovie, LoadVars)
SocketRead	Controls access to ActionScript API for reading from XML Sockets
SocketWrite	Controls access to ActionScript API for writing to XML Sockets
NetworkVideo	Controls access to video resources on remote network
GetURLAll	When true, all GetURL operations are allowed
GetURLSWF	Only applicable to External standalone mode. Controls whether a SWF is allowed to replace itself with a getURL call to another SWF movie. In other modes, please refer to the internal GetURLSWFDSMode bit.
GetURLHTTP	Controls whether the channel can make getURL calls over HTTP.
GetURLHTTP_POST	Controls whether the channel can post data in getURL calls over HTTP.
GetURLHTTPS	Controls whether the channel can make getURL calls over HTTPS.
GetURLHTTPS_POST	Controls whether the channel can post data in getURL calls over HTTPS.
GetURLTel	Controls access to getUrl(tel:) command and system://confirmtel command
AllowedToBeScripted	Controls the channel SWF can be controlled with ActionScript by other channel SWFs.
AllowedToScriptLocalFile	Controls whether the channel SWF can control local SWFs on the device with ActionScript.
AllowedToScriptOtherFeeds	Controls whether this channel may control other channel SWFs with ActionScript.
AllowChildInheritPermissions	Controls whether child movies loaded into the main SWF inherit its parent's full permissions.
GetFilter	Controls read access to preferences of caller's own feed
SetFilter	Controls write access to preferences of caller's own feed
GetDeviceID	Controls whether channel SWF may read the ID of the device.
SetOtherSetting	Controls write access to the setting data of other feeds
SetOtherScratchpad	Controls write access to scratchpad data of other feeds
SetOtherFilter	Controls write access to preferences data of other feeds
GetURLSWFDSMode	In any non-ESA mode, this bit controls whether a SWF is allowed to replace itself with a getURL call to another SWF movie.
TurnOffSMSConfirmation	Turn off the system confirmation dialog for getUrl(sms:)
TurnOffTelConfirmation	Turn off the system confirmation dialog for getUrl(tel:)
GetURLLocal	Allow GetURL to load local content (url with "file:" prefix)
SubscriptionRead	Controls GetFilter, GetSettings, GetScratchpad's access to subscription feed if

	a feed id is specified.
SubscriptionWrite	Controls SetFilter, SetSettings, SetScratchpad's access to subscription channel's feed, if a feed ID is specified.
GetOtherSetting	Controls read access to settings data of other channels.
GetOtherScratchpad	Controls read access to scratchpad data of other channels.
GetOtherFilter	Controls read access to preferences data of other channels.
AllowExitParameters	When on, allows current application to use "exitFlashCast" fscommand with parameters so the application state stack can go back to home or root .
<b>Adobe Extended Permissions</b>	
SystemInformationRead	Controls whether channel has read access to all system information on the device.
SystemInformationWrite	Controls whether channel has write access to all system information on the device.
CallLogRead	Controls whether channel has read access to the device's call log.
CallLogWrite	Controls whether channel has write access to the device's call log.
ContactsRead	Controls whether channel has read access to the device's contacts list.
ContactsWrite	Controls whether channel has write access to the device's contacts list.
MessagingRead	Controls whether channel has read access to the device's messaging in-box.
MessagingWrite	Controls whether channel has write access to the device's messaging in-box.
AppLauncherRead	Allows an application to use the device extension ApplicationLauncher class functions launchApplication() and launchDialog()..
AppLauncherWrite	Not used.
AppMsgRead	Allows an application to use the device extension ApplicationMessaging class function receiveApplicationSMS().
AppMsgWrite	Allows an application to use the device extension ApplicationMessaging class function sendApplicationSMS().

### 13 Appendix D: Device profile classes and capabilities

This appendix lists the device profile classes and capabilities that are recognized by an Adobe Mobile Server or an Adobe Mobile Client. Another client-server implementation may come up with their own device profile class names and capabilities.

<b>Profile class type</b>	<b>Capabilities in this profile class type</b>
platform	<p>version: A string that specifies the client version.</p> <p>For example:</p> <pre>version=2.1</pre>
player	<p>swfversion: An integer that specifies the SWF version supported by the client.</p> <p>For example:</p> <pre>swfversion = 6</pre>
encoding	<p>encoding: A string that specifies the character encoding used by the client. Standard</p>

	value such as ISO-8859-1 for Western European (ISO) as defined by IANA.
locale	<p>locale: A comma-delimited list of locales in the standard format (for example, en-US, zh-CN, and so forth).</p> <p>For example:</p> <pre>locale = en-US,es-ES</pre>
audio	<p>af: A comma-delimited list of supported audio formats, separated by commas.</p> <p>For example:</p> <pre>af = mid,mp3,qcp</pre>
video	<p>vf: A comma-delimited list of supported video formats, separated by commas. For example:</p> <pre>vf = mpeg4,3gpp2</pre>
image	<p>if: A comma-delimited list of supported image formats, separated by commas.</p> <p>For example:</p> <pre>if =bmp,gif,bci</pre> <p>ejpeg: Integer value that specifies the client's level of support for displaying embedded JPEGs inside SWF files. Possible values are 1 (minimum support), 2 (medium support), and 3 (maximum support).</p> <p>For example:</p> <pre>ejpeg:= 1</pre>
display	<p>width: Width of device's display in pixels (integer).</p> <p>height: Height of device's display in pixels (integer).</p> <p>pixel: Standard value such as 565, 555, and so on (integer).</p> <p>For example:</p> <pre>width=240 height=480 pixel=565</pre>
network	<p>g: Network generation such as 1G, 2G or 3G (float), for example:</p> <pre>g = 2.5</pre>
handset	<p>model: A string that specifies the device model.</p> <p>For example: model=Samsung_a950</p>

	<p>os: A string that specifies the device's OS.</p> <p>For example: <code>os = brew3x.</code></p> <p>endian: A string that specifies the byte order (big-or little-endian) used by the OS.</p> <p>For example: <code>endian=little</code></p> <p>platformid: A string that specifies the platform's ID.</p> <p>For example: <code>platformid=3927</code></p>
extra	<p>A string containing a comma-delimited list of additional supported formats, such as SMS or email. For example:</p> <p><code>extra=sms, email</code></p>