Research White Paper

WHP 194

June 2011

The Universal Control API v.0.6.0

Specification for the behaviour of a universal control server running on a set-top box, and the clients that connect to it

J.P. Barrett, M.E. Hammond, S.J.E. Jolly

BRITISH BROADCASTING CORPORATION

BBC Research & Development
White Paper WHP 194

The Universal Control API v.0.6.0

James P. Barrett Matthew Hammond Stephen Jolly

Abstract

The Universal Control API is a RESTful Web API designed to allow the core functionality of any network-connected set-top box, personal video recorder, Internet radio or similar device to be controlled from client software running on another device on the home network. This note specifies the REQUIRED and RECOMMENDED behaviour for Universal Control server and client implementations. Informative sections are included where necessary to explain the context and reasons for the normative decisions that have been made. This White Paper replaces a previous one describing an earlier version of the API.

Additional key words: remote, control, interface, HTTP, REST, accessible, IP

BBC Research & Development
White Paper WHP 194

The Universal Control API v.0.6.0

James P. Barrett Matthew Hammond Stephen Jolly

Contents

White Papers are distributed freely on request.

Authorisation of the Head of Research is required for publication.

BBC Research & Development
White Paper WHP 194

The Universal Control API v.0.6.0

James P. Barrett Matthew Hammond Stephen Jolly

1 Introduction

The Universal Control (UC) API is a protocol that allows a media device to be controlled by client software via an IP network. The protocol has been designed with a television set-top box, digital video-recorder, or Internet radio as the expected server device, although the API is expected to be applicable to a wider range of devices. The security and authentication mechanisms have been defined with these three kinds of device and in-home use in mind. Server and client implementations are possible on a wide range of devices. The API has been designed to conform as far as possible with RESTful web API design patterns, and has been designed to minimise the processing required by either client or server.

A document describing the design decisions taken in the process of defining the UC API has been published to complement this specification [1].

In this document, we use the word "server" to refer to a piece of software that implements the server aspects of the UC API, and the word "box" to refer to the hardware that the server controls. In many implementations, the server will run as a background process on the box. We also talk about the box's "built-in interface". By this we mean any user interface functionality that comes with the box as sold. For example, the "built-in interface" of a set-top box could be the menu system it presents on the television connected to it, and the infra-red remote control that controls that menu system. The built-in interface of an Internet radio could be a display and buttons built into the radio itself, along with a GUI controlled by those buttons.

In this document, we use the word "client" to refer to a single instance of a piece of software that implements the client aspects of the UC API, and "client device" to refer to the hardware on which the client is running. The protocol has been designed with the intention that it should be possible to implement clients as web, widget or native applications running on mobile phones, computers and embedded devices. See the overview document [1] for more details.

This specification is divided into several parts. Section 2 describes the data model that underlies the Universal Control API. Section 3 and section 4 describe the behaviour of a compliant Universal Control server. Section 5 describes the behaviour of a compliant Universal Control client. Implementers of either servers or clients are advised to read all these parts. In addition, section 6 contains some guidelines for interactive application authors who wish to use the API extension mechanism defined in section 4.44 to enable direct communication with compatible Universal Control clients on the local network.

1.1 API Version

This document describes version 0.6.0 of the Universal Control API. Version numbers starting with "0" indicate a version of the specification which is considered by its authors to be implementable but not final. The authors particularly welcome feedback regarding defects or enhancements to these versions of the API.

1.2 Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119[2].

1.3 Date and Time Format

In this document we make references to "Internet standard timestamp format". In all cases this means the format called "`date-time`" specified in [3], section 5.6. This is a profile of the ISO 8601 standard. The level of precision in a given timestamp SHOULD reflect the accuracy of the underlying timing data from which it is generated.

1.4 Image Formats

This API defines various opportunities for servers to indicate to clients that an image representing a data entity is available, by providing them with the URL of a suitable image. It is RECOMMENDED that the images linked to in this fashion be in JPEG or PNG format, and be no larger than 512×512 pixels. It is RECOMMENDED that the ratio of the longest to the shortest side of each image does not exceed 2:1.

2 Data Model

The Universal Control API has its own data model for describing media and its metadata, which is designed to be simple yet flexible, with good mappings between it and the data models of existing media distribution systems such as DVB, analogue television, DLNA, disc-based media such as DVDs or CDs, and Video On-Demand (VOD) services. The suitability of the model is by no means limited to these examples, and by making the UC data model as general as possible, it is hoped that it will be equally applicable to new kinds of service too.

Because the UC API is highly modular, servers can be written that do not present information to clients about a box's media. Implementers of such servers can completely ignore this section.

In this section, the various entities in the data model are described. Some informative examples of mappings between the UC data model and those of existing systems are then given in section 2.7.

Servers SHOULD only provide information about entities for which there is a reasonable expectation of availability via the box. For example, servers SHOULD NOT provide clients with information about television channels and the programmes on them if those channels are not received by the box.

2.1 Sources

2.1.1 Introduction

This section is purely informative.

A "source" is a distinct conceptual source of media such as a DVB service, a video on demand service on the internet, a DAB service, or any storage media available to the box, such as an internal hard-drive. Each source has the following optional or required properties associated with it:

- A `source-id` string (required), unique to that source, that is used to identify that source within the API.
- A human readable `name` (required), used by the client to represent that source to the user.
- An `sref` URI (optional, but recommended if available) that identifies the origin of the source's content, explained in section 2.1.2.

- Boolean `live`, `linear`, `follow-on`, and `audio-only` attributes (optional) that are explained below.

- A human-readable `owner` string (optional), used by the client to represent the source's owner to the user.

- A "logical channel number" (optional), which allows media sources with a "channel numbering" convention, such as TV channels in the UK, to retain that numbering system in client user interfaces. See below for more details.

- A `logo-href` URL (optional), which points to an image that the client can use to identify the source to the user.

- An `owner-logo-href` URL (optional), which points to an image that the client can use to identify the owner of the source to the user.

- A `default-content-id` (optional), which is an `content-id` (see section 2.2) identifying the source's "default piece of content", if it has one. The default content is the content which would be automatically selected if the user asked the box to present that source. It can change at any time. In the context of a source representing a TV channel, the default content could be the programme that is currently being broadcast, for example.

- An unlimited number of "links". A link is a URL and a human-readable description, similar to the concept of a hyperlink. Servers can associate one or more links with a source to give clients access to related information about it. A source representing a TV channel could have a link that pointed to a web page about that channel maintained by the broadcaster, for example.

The logical channel number (LCN) is a concept inherited from the implementation of DVB in the United Kingdom, in which such a number is associated with each DVB service (a DVB service is the conceptual equivalent of a TV channel), and the user can typically select a service for viewing on a STB by entering the logical channel number using the infra-red remote control. The number is usually also used to provide the order in which services are listed in the box's electronic programme guide and other UI components.

The "owner" property is an example of a "human-readable identifier" in the API. Such identifiers are strings that are intended to be human-readable, but multiple sources from the same owner always use exactly the same string representation of the owner, allowing clients to group them together.

Three properties in the list above can be associated with each source by the server in order to convey information to the client about ways in which it is appropriate to treat the source:

1. The "live" property informs clients that a source has a concept of a current playback position that exists whether or not media from the source is being presented to the user. This property tells clients that playback may not start at the beginning of a piece of AV content (see Section 2.2 for a definition of AV content) when a client instructs the server to present the source (or a piece of AV content from it) on an output.

2. The "linear" property informs clients that all of the pieces of AV content associated with a source have both start and end times, and that the start and end times of different pieces do not overlap. This property tells clients that it is appropriate to present information about AV content from that source in the form of a time-line, or as a row in a traditional "grid" electronic programme guide, and also that it is likely to be appropriate to make a request to present this source without asking the user to select from amongst multiple potential pieces of content to present beforehand.

3. The "follow-on" property tells clients that when one of the pieces of content from the source stops presenting to the user presentation of another piece of content begins immediately, without interrupting the flow of media. This gives clients the information they need to manage the box behaviour when some content stops presenting on an output, and ensure that it matches the user's desired behaviour. It also tells clients that the source is probably not appropriate for inclusion in playlists.

4. The "audio-only" property tells clients that this source will never contain video content.

A TV channel or a radio station would be good examples of sources with the first three of these properties. Each of them represents a continuous stream of media, with a "current playback position" corresponding to whatever they may be broadcasting at a given point in time. Each of them has a schedule consisting of non-overlapping programmes, and playback of one programme follows the previous one without interrupting the flow of media (although depending on how a broadcaster has chosen to define the start and end times of its programmes, there may be times when media is being presented that does not correspond to a specific piece of content, such as when an "ident" is being presented that identifies a TV station to its viewers). The radio station would also have the "audio-only" property. Conversely, the source representing the recordings in the local storage of a PVR may well have none of these properties. A source representing a DVD in a DVD player could have the "follow-on" property but none of the other three. A source representing a CD in a CD player could have the "follow-on" and "audio-only" properties.

See section 2.7 for some examples of how real-world media sources can be represented within the UC data model. See section 4.16 for more details of the way that servers generate these properties in resource representations. See also the XML Schema definition of `source_t` in Appendix D.

2.1.2 Implementation

This section (and all other sections in this document not marked as "purely informative") is normative.

When responding to requests and sending responses the server MUST refer to individual sources using `source-id`s. A `source-id` is a string containing only a characters from a restricted subset of ASCII/UTF-8. Specifically, in ABNF[4] this is specified as:

```
source-id  = id-element

id-element = 1*( unreserved / pct-encoded )
unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
```

The `id-element` in this definition is used as the basis for the definition of many other kinds of identifier in this specification[1].

If a server implementer wishes to make use of an existing identifier string as an `id-element` in the UC API (examples of existing identifier strings could include DVB service names, TV-Anytime CRIDs, or the fully qualified domain name (FQDN) of a VOD service), and that string meets all the requirements for the identifier in question other than the restrictions on valid characters, then the server MAY use "percent encoding" [5] to convert it into an acceptable format.

Each `source-id` MUST NOT be associated with more than one source on the box, and once a server has used a particular `source-id` in processing a request or forming a response it MUST continue to identify the same source until the box is powered down. The box MUST retain this association of `source-id`s with sources for as long as the box is able to do so, including retaining this association during periods when the box is powered off if the box has sufficient persistent storage

[1] Characters valid in an `id-element` are a subset of those that are valid in a URI path component as described in the URI syntax[5]

to permit this, since clients may request the playback of sources assuming that the associations have not changed. Each source SHOULD NOT have more than one `source-id` at one time.

When multiple sources share the same owner, and the server includes details of source owners in resource representations, the same human-readable string value SHOULD be used to represent the owner in each case. The server MUST NOT use the same string value to identify two different source owners.

If supplied, the `sref` attribute SHOULD contain a URI that represents the origin of the content represented by the source. For example, if the source represents a DVB service, the `sref` attribute SHOULD be the DVB locator [6] for that service (eg. `dvb://233a.1048.1048`). If there is no standardised way of generating a URI that represents a particular kind of source, the server MAY provide an implementation-specific `sref`, which SHOULD be an identifier for that source with global uniqueness (for example, the canonical URL of an Internet Video On-Demand service).

2.2 Content

2.2.1 Introduction

This section is purely informative.

Each source in the UC data model can have an unlimited number of pieces of content associated with it. These pieces of content may be presented one after the other in a source with the `linear` property, or be available simultaneously from local storage or from some kind of on-demand service. A given piece of content is associated with precisely one source.

We define the term "content" very specifically. There are two types of content, "AV" content refers to particular instances of particular audiovisual productions, such as specific broadcasts or files, and not to the productions themselves, whilst "interactive" content refers to particular instances of particular pieces of interactive media, such as specific "apps" stored on the box's storage, or specific broadcasts of MHEG interactive applications. For example, the episode of "Doctor Who" broadcast on 22nd May 2010 on BBC One (Cambridge region) at 18:15 is a piece of AV content in the context of the Universal Control API. The episode of "Doctor Who" available on the BBC iPlayer at the URL "`http://www.bbc.co.uk/iplayer/episode/b00sj9sq/Doctor_Who_Series_5_The_Hungry_Earth/`" is also a piece of AV content. "Doctor Who", series 5, episode 8 is *not* a piece of content in the context of the UC data model, even though the two aforementioned examples are instances of that episode. "Doctor Who" is also not a piece of content. As general guidance content should be modelled as interactive if it is designed to respond to user interaction other than that which is expected of ordinary audio-visual material (such as pausing, fast-forward, etc ...), or if it does not present any media to the user at all (such as a service which runs in the background).

Pieces of interactive content may be informally referred to as "apps" whilst pieces of AV content may be informally referred to as "programmes". All content must be either interactive or AV; no content can be both.

Each piece of content has the following optional or required properties associated with it:

- A `content-id` (required) that uniquely identifies the piece of content within the scope of its source.
- An `cref` URI (optional) that identifies the origin of the content, explained in section 2.2.2.
- A `logo-href` URL (optional), which points to an image that the client can use to identify the source to the user.
- A `synopsis` (required if available), which provides a short textual summary or description of the content.

- A `global-content-id` (required if available) that gives an identifier for the media in the content with a scope wider than that of the box, such as a TV-Anytime "Content Reference Identifier" (CRID). See section 2.2.2 for more details.
- A `series-id` (required if available) that identifies a series that the content belongs to. See section 2.2.2 for more details.
- A `global-series-id` (required if available) that gives an identifier for the series to which the media belongs with a scope wider than that of the box, such as a TV-Anytime "Content Reference Identifier" (CRID). See section 2.2.2 for more details.
- A human-readable `title` (required) that the box can use to represent the content to the user.
- A `interactive` (required) property, which is true if the content is interactive, and false if it is AV content.
- A `presentable` (required if available) property which is true if the content can immediately be presented on an output, and false otherwise.
- A `acquirable` (required if available) property which is true if the content is such that the box can through some action cause it to become available to acquire at a later time, and false otherwise (for example an application which is available to download, but cannot be run until it has been; or a television programme which is currently visible on the schedule, and can be scheduled for future recording).
- `start` time for the content (required if available for AV content which is part of a linear source).
- `duration` for the piece of content (required if available for AV content).
- `presentable-from` and `presentable-until` properties (required if available), representing times between which the content is expected to be available for presentation.
- `acquirable-from` and `acquirable-until` properties (required if available), representing times between which the content is expected to be available for acquisition.
- A `presentation-count` property (required if available) that indicates how many times the content has *begun* being presented.
- A `last-presented` property (required if available) that indicates the most recent time at which the content *stopped* being presented.
- A `last-position` attribute (required if available for AV content) which indicates the point within the AV content at which presentation most recently stopped.
- A list of the "media components" that make up the content (required if available for AV content) – see section 2.4.
- A list of the `category-id`s of the categories into which the content falls (required if available) – see section 2.3.
- A list of `control-profiles` (see 7) describing those button-presses to which a piece of interactive content will respond (required if available for interactive content).
- A `extension` property (required) which is true if the content is interactive and implements the Universal Control API extension mechanism (see 4.44) and false otherwise.

- An `associated-content-id` (required if available) which contains the `content-id` of another piece of content directly related to this one. See below for more information about this mechanism.
- A `associated-source-id` (required if available) which contains the `source-id` of of the source in which the associated-content-id is valid.
- An unlimited number of "links". A link is a URL and a human-readable description, similar to a hyperlink. Servers can associate one or more links with a piece of content to give clients access to related information about it. A piece of content representing a TV programme could have a link that pointed to a web page about that programme maintained by the broadcaster, for example.

The concept of "associated content" is introduced above in the definition of the `associated-content-id` and `associated-source-id` attributes. These attributes are intended to provide a mechanism for associating AV content and interactive content. Each piece of AV content MAY be associated with a piece of interactive content via this mechanism, and vice-versa. Server implementers MUST NOT use this mechanism to associate two pieces of AV content with one another, or to associate two pieces of interactive content with one another.

In addition some content is considered "stored content". This is content which is stored on limitted capacity storage media accessible to the box to which content can be saved and from which content can be deleted. Such content will only be found in sources contained in the source-list "uc_storage", and all sources in that source-list will contain only stored content. It is highly likely that the actual media represented by content will often be copied from other sources to storage sources, and that the newly created item of content which results will be an item of stored content (this happens, for example, when a television programme is recorded on the PVR or when an application is downloaded from an application store). There is no requirement that the newly created piece of content have the same "content-id" as the original content from which it was "acquired", and in fact it is highly unlikely that this will generally be the case. Note that there need not be a 1:1 correspondence between "stored-content" and files in physical filesystems on the box's hard-drive or other media. That detail is implementation-dependent.

See section 2.7 for some examples of how content from real-world media sources can be represented within the UC data model. See section 4.19 for more details of the way that servers generate these properties in resource representations. See also the XML Schema definition of `content_t` in Appendix D.

2.2.2 Implementation

An individual piece of content within a source MUST be identified using a `content-id`. The format for a `content-id` is as follows:

```
content-id      = id-element
```

Each source MUST be associated with at least one piece of content. The server MUST NOT identify two pieces of content in the same source using the same `content-id`. The server MAY use the same `content-id` to identify two pieces of content (different or otherwise) in different sources.

Each piece of content MAY be associated with a single locally-scoped `series-id`. In the context of the UC API, a "series" is a collection of individual pieces of content that may be from any source. No information about a series is made available via the API other than the content it contains. The `series-id` can be used by clients to obtain information about one or more pieces of content in a given series (from the "uc/search/sources/{sid}" resources, defined in section 4.22), and to

request the acquisition of content in a given series that is not yet available for presenting (via the "uc/acquisitions" resource, defined in section 4.37). The format for a `series-id` is as follows:

```
series-id = id-element
```

Each piece of content MAY also be associated with a single `global-content-id` and OPTIONALLY also a single `global-series-id`, which MUST both take the form of a valid URI (see [5]). The purpose of these identifier is to allow content and series to be referenced using an identifier with a scope larger than that of the box. For example, a TV-Anytime CRID identifying a television programme or a series might be obtained from the box by a client. If the broadcaster also supplies a CRID resolution service, or a web-based VOD service that uses the same identifiers, the client may be able to display the content to the user directly in circumstances where the user prefers that to viewing it on one of the box's outputs.

Any server implementer wishing to make use of pre-existing global identifiers for content which do not take the form of a URI SHOULD employ the following urn-scheme to encapsulate them[2]:

```
uc-global-id      = "urn:X-UCID:" [ domain ] ":" id-specific-part

id-specific-part  = 1*( urn-unreserved / urn-sub-delims / pct-encoded / ":" )
urn-unreserved    = ALPHA / DIGIT / "-" / "." / "_"
urn-sub-delims    = "!" / "$" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="

domain            = subdomain 0*("." subdomain)
subdomain         = 1*char  0*(1*"-" 1*char)
char              = ALPHA / DIGIT
```

the `domain` component SHOULD be present and SHOULD be a fully-qualified domain-name registered to the identifier provider in the global DNS, eg. "bbc.co.uk" if one is available; otherwise it MUST be omitted, which indicates that the identifier does not have global scope. (It may still be unique within a scope that extends beyond the data obtained from a single box however; indeed this must be the case if the identifier is to be useful.) The `global-content-id`s and `global-series-id`s are only useful for their intended purpose if a client has a way of obtaining or using them which is unrelated to the UC server.

Note that the `id-specific-part` defined here is identical to that of the `id-element` described above, with the additional restriction that it MUST NOT contain the character '~'. This additional restriction is present because that character is forbidden in URNs.

In addition, a `global-app-id` is REQUIRED for pieces of interactive content. The `global-app-id` MUST be of the form:

```
global-app-id       = [ domain ] ":" appid-specific-part
appid-specific-part = id-element
```

Restrictions on the generation of `global-app-id`s apply, but their exact nature depends on whether or not the interactive content implements the API extension mechanism defined in section 4.46, which defines a way for individual pieces of interactive content to extend the API at runtime and exchange arbitrary information directly with remote clients:

- If the interactive content implements the API extension mechanism, the `global-app-id` MUST NOT be generated by the box, and MUST be obtained in its entirety from the

[2]The NSS employed here, "X-UCID" is experimental (hence the "X-" prefix) and hence is expected to be replaced with a replacement NSS in a future revision of this spec (see [7] for details).

interactive content itself (or from metadata associated with it). The reason for this is to give clients a reliable way to identify interactive content whose extension APIs they are aware of. The `domain` component MUST be present and SHOULD be a fully-qualified domain name registered to the interactive content provider in the global DNS. As a whole, the `global-app-id` MUST uniquely identify the interactive content with global scope; it is the responsibility of the interactive content provider to ensure this. See section 6 for more information.

- If the interactive content does not implement the API extension mechanism (or if the box does not implement it), the `global-app-id` MAY be generated by any means, but SHOULD be the identifier supplied by the interactive content provider if such an identifier is available. If the identifier used does not have global scope, the `domain` component MUST be absent.

Each piece of content contained within a source SHOULD be associated with a single `cref` attribute, if a suitable standardised URI scheme exists to identify pieces of content from the kind of source from which this content is obtained. The value of this attribute SHOULD be a URI that represents the origin of the content. For example, if the source represents a DVB service, the `cref` attribute of the content associated with that source SHOULD be a DVB locator that identifies the appropriate DVB event from that service, as defined in the DVB locator standard [6]. If there is no standardised way of generating a URI that represents content from a particular kind of source, the server MAY provide an implementation-specific `cref`, which SHOULD be an identifier for that source with global uniqueness, and scope large enough to cover the local network (for example, the canonical URL of a programme supplied by an Internet Video On-Demand service).

2.3 Categories

2.3.1 Introduction

The UC data model incorporates a hierarchy of categories for content. As described in section 2.2, each piece of content may have a list of categories associated with it. The API also defines a "uc/categories" resource (see section 4.36) that gives clients access to the tree structure of the category hierarchy, and human-readable names and (optionally) images that can be used when representing categories to the user.

The use of categories is particularly important for sources with very large numbers of pieces of content whose availabilities overlap substantially in time, such as video-on-demand services and application download services: associating each piece of content with one or more categories from the hierarchy gives users (of clients that make use of the information) a useful way to narrow down the very large lists of content that may be associated with a source, or the results of a query to the "uc/search/categories" resource (see section 4.28).

A common use for category hierarchies is to convey genre information, but this is by no means the only kind of categorisation scheme that may be employed. Server implementers may choose to make "Genre" just one of a number of top-level categories, which may also include "Featured Programmes", "Popular Programmes" or perhaps "My Recommendations" categories, depending on the desire of the server implementer, and the richness of the information available to them.

If they so desire, server implementers can implement a "hierarchy" that is simply a list of top-level categories, with no subcategories at all: for example, this may be appropriate if the number of pieces of content in the sources made available via the UC API is small, or if the source of categorisation information available to the server is not hierarchical itself.

An illustrative example of a category hierarchy is shown in Figure 1. Note that the root of the tree is not considered to be part of the tree itself. The tree shown includes both genres and non-genre categories. The "Genres" category itself does not make sense as a category for content, but is used in this example to separate genres and non-genre categories. Note the apparent repetition of the "Factual" genre: it appears both as a sub-genre of the "Children's" genre, and as a genre in its

Figure 1: A tree diagram illustrating an example category hierarchy. Human-readable category names are used to represent the categories in this diagram.

own right. This is permissible: category hierarchies should be designed such that the categories at a particular level of the hierarchy make sense in the context of their parent category (if any), and as described below, unique category names are only required for the sub-categories of a particular parent. As a consequence of this, client implementers are advised to include representations of a category's parents when presenting a category to the user. (See section 5.11 for some suggestions regarding how this may be achieved.)

It is permissible (but not required) for content to be associated with categories that have sub-categories.

It is permissible for the server to support categories (including the implementation of the "uc/categories" resource described in section 4.36, by which the category hierarchy is made available to the client), but for there to be no categories in its category hierarchy. Server implementers are advised to avoid this whenever possible.

2.3.2 Implementation

Categories in the server's category hierarchy are subject to the following requirements:

- Each category in the hierarchy MUST have a single parent (which MUST be either another category or the hierarchy root), and MAY have any number of children (also known as "sub-categories").
- Each category MUST have associated with it a human-readable category name, which MUST be unique amongst the children of that category's parent.
- Each category MAY have associated with it a `logo-href` URL which points to an image that the client can use to represent the category to the user.
- Each category that the server might later associate with one or more pieces of content MUST have a `category-id`. It MUST be the case that no two categories in the hierarchy have the same `category-id`.
- A category that will never be associated with content but which has nevertheless been included in the hierarchy (to make it more easily navigable by users for example, as is the case for the "Genres" category shown in Figure 1) SHOULD NOT have a `category-id`.

The hierarchy root is not a category: it MUST NOT have any of the properties mentioned above.

The format for a `category-id` is as follows:

```
category-id = id-element
```

For the purposes of reporting and matching categories, it is assumed that a piece of content that is a member of a sub-category is automatically a member of that category's parent category. When reporting a list of the categories with which a piece of content is associated, a server SHOULD NOT report any categories that are parents (or grandparents etc) of another category in the list. When matching categories supplied by a client in a query to the server, the server SHOULD also match categories that are children (or grandchildren etc) of the categories supplied.

2.4 Media Components

2.4.1 Introduction

This section is purely informative.

Since the addition of sound to film, it has been common for media to be comprised of multiple synchronised components of different media types. In the UC data model, we consider these components to be the properties of individual pieces of AV content. Three types of component are modelled: video, audio, or subtitles. Interactive content associated with a piece of AV content is not treated as a media-component, instead it is treated as a separate piece of associated content. A given piece of AV content always has at least one media-component associated with it, but the server may not be aware of what its media-components are until they are presented to the user.

Each media-component has the following optional or required properties:

- An enumerated string `type` (required) that identifies what kind of media component is being represented, that takes one of the values "video", "audio", or "subtitles".
- A `media-component-id` (required) that uniquely identifies the media component within the scope of the piece of AV content it is a component of.
- A human-readable `name` (optional) that clients can use to represent the component (perhaps in conjunction with its type) to the user. This is particularly important if two components of the same type are associated with the same piece of content. For example, audio components conveying the main programme audio in stereo and surround sound variants could be identified as "Main Audio (stereo)" and "Main Audio (surround sound)".)
- A human-readable identifier of the language in which the component is provided (optional), if appropriate. (For example, video components conveying BSL and Makaton signed versions of the content could be identified as "Signed: BSL" and "Signed: Makaton" respectively.
- An enumerated string representation of the aspect ratio of a video component (optional), if appropriate. This may take the values:
  - "4:3"
  - "14:9"
  - "16:10"
  - "16:9"
  - "21:9"

  The value "source" permitted by the XML schema (see Appendix D) is not a valid aspect ratio for a media component.

- An enumerated string representation of the intent of the component (required if applicable), if the component has one of the intents described below. At present, this property can take the following values:
  - "admix" if the component contains a mixture of main programme sound and "audio description", intended for the benefit of people who are blind or otherwise visually impaired.
  - "hhsubs" if the component contains subtitles intended for the benefit of people who are deaf or have other hearing impairments.
  - "signed" if the component contains sign language.
  - "iimix" if the component contains an alternative audio mix intended to provide improved intelligibility, especially for listeners with hearing impairments.
  - "commentary" if the component contains a commentary on the content of the programme.
- An enumerated string indicating the underlying format of a video component (optional). At present the values "HD", "SD" and "S3D" (for stereoscopic 3D content) are permissible.
- An enumerated string indicating the underlying format of an audio component (optional). At present the values "mono", "stereo" and "surround" are permissible.
- A boolean value indicating whether a video component is in colour ("true") or black-and-white ("false") (optional).
- A boolean value identifying whether or not the component is "default" (optional, but assumed to be "false" if not specified). See below for details of what this implies.

The server and/or the box may determine that certain media components of a piece of content are "default". For example, a piece of content that represents the current item on a TV channel might have a video component, two audio components in different languages and a subtitle component. In this case, the video component and the audio component that corresponds to the user's preferred language might be considered "default" and the subtitle component and the other audio component might be considered not to be "default". These decisions are implementation-specific and may be based on almost any criteria, such as box factory settings and default preferences expressed via the box's built-in interface. The effect of a component being declared to be default is that unless a client explicitly requests otherwise, default components will be presented to the user when a piece of content is presented on an output, and non-default ones will not.

See section 4.10 for details of how a client requests a non-default component. See section 4.19 for information about how servers can inform clients about media components contained in pieces of content. See also the XML Schema definition of `component-reference_t` in Appendix D.

2.4.2 Implementation

Each piece of AV content contained within a source MUST contain at least one "media component". The media components associated with a piece of content MAY not be known until that content is being presented to the user. Examples of media components include each video stream, each audio stream, and each subtitle track in a transmitted programme. When processing requests and forming responses the server MUST use `media-component-id`s of the form:

```
media-component-id = id-element
```

to identify each media component.

The server MUST NOT use the same `media-component-id` for two media components in the same piece of content. The server MAY use the same `media-component-id` for two components in different pieces of content.

The server SHOULD use the `name` attribute of media components to provide human-readable component names to clients, particularly if two or more components of the same type are present in the same piece of content (for example, a main programme audio track and an audio description track). It SHOULD be the case that no two components of the same piece of content have the same `name` attribute, even if it would be possible for clients and users to distinguish between them using other properties of the components.

2.5 Source Lists

2.5.1 Introduction

This section is purely informative.

A "source list" is simply a collection of sources, designed to allow sources to be grouped together in various ways that help the user locate them. For example, sources that correspond to DVB services and VOD services on an STB could be included in two different lists, allowing clients to present them separately. Every source that the server represents via the API must appear in at least one source list (but can appear in more than one list).

Each source list has the following optional or required properties:

- A `list-id` (required) that uniquely identifies the source list within the scope of the box.
- A human-readable `name` (required) that clients can use to represent the source list to the user.
- A `logo-href` URL (optional), which points to an image that the client can use to identify the source to the user.
- An unlimited number of "links". A link is a URL and a human-readable description, similar to a hyperlink. Servers can associate one or more links with a source list to give clients access to related information about it. A UC source list representing a DVB network could have a link that pointed to a web page about that network, for example.

A source list with the `list-id` "uc_default" is required to be implemented on every UC server. A source list with the `list-id` "uc_storage" is required to be implemented if the server implementer wishes to represent media on storage devices (such as USB disks or an internal hard disk) via the API.

See section 2.7 for some examples of how real-world media sources can be grouped together into lists within the UC data model. See section 4.15 for more details of the way that servers generate these properties in resource representations. See also the XML Schema definition of `sources_t` in Appendix D.

2.5.2 Implementation

Each "source list" MUST have a `list-id` of the form:

```
list-id        = id-element
```

The `list-id`s "uc_storage" and "uc_default" are reserved as defined below. Each `list-id` MUST be unique amongst the `list-id`s on the box. The same `list-id` MUST continue to be usable for addressing the same list for as long as the box is switched on, and SHOULD persist for longer periods. The content of each list MAY change, however.

Every box MUST implement a "source list" with the reserved `list-id` "uc_default", and MAY implement any number of other "source lists". Choosing which sources are on each list is left up

to the implementer, but it is RECOMMENDED that the "uc_default" source list should contain all the sources that a user might reasonably expect to be available to the box.

Every `source-id` that is valid for use within this API MUST be contained in at least one "source list". This means that on boxes which only maintain the "uc_default" list, this list will contain all valid sources on the box, on other implementations this may not be the case.

If the box has access to and control over storage media, (for example, an internal hard-drive or flash memory card) then this capability MAY be represented by one or more sources in a source list with the reserved `list-id` "uc_storage" and MAY be present in one or more other source lists. The reserved `list-id` "uc_storage" MUST NOT be used for any other list.

The server MAY assign a "logical channel number" to any or all sources, as described above. If the box assigns a channel number to some sources in its built-in user interface, then it SHOULD use the same number as the value of the `lcn` attribute of the `source` element that represents that source in representations returned by this API. If a source with an LCN is present in more than one source list, the server MUST give it the same `lcn` value in each list. The LCN attribute should not be used simply to impose an order on lists of sources.

2.6 Acquisitions

2.6.1 Introduction

This section is purely informative.

An "acquisition" is a piece of information maintained by the box which identifies a piece of content which is not currently available for presentation, but is expected to be "acquired" at some point in the future so that it will become available for presentation; once available it will then be expected to remain so for an extended (but undefined) period of time.

For example, an "acquisition" can represent the box's expectation that it will record a particular television programme to its local storage once it starts showing, or the expectation that it will download a specific application or on-demand programme from a given Internet service and store it to local storage media for later use.

Acquisitions come in two forms, `content-acquisitions` represent the expectation that a box will acquire a particular piece of content when it becomes available, and `series-acquisitions` represent the expectation that the box will create a `content-acquisition` for each piece of content in a particular series when such content becomes known to the box. Each content-acquisition has the following optional or required properties:

- An `acquisition-id` (required) which must uniquely identify the acquisition within the scope of the box.
- A `source-id` and `content-id` (required) which identifies the content which will be acquired.
- A `priority` flag (optional) which must be either "true" or "false".
- A `series-linked` flag (required) which must be either "true" or "false".
- A `speculative` flag (required) which must be either "true" or "false".
- An `active` flag (optional) which identifies whether the content is currently in the process of being acquired.

and each series-acquisition has the following optional or required properties:

- An `acquisition-id` (required) which must uniquely identify the acquisition within the scope of the box.
- A `series-id` (required) which identifies the series the acquisition will acquire content from.

- A `priority` flag (optional) which must be either "true" or "false".
- A `speculative` flag (required) which must be either "true" or "false".

We define here the act of "booking" an acquisition to be the acceptance of an acquisition request by the box, and the corresponding reservation of any box resources (such as an RF tuner) required to fulfil it, if these are known at the time the acquisition request is made. A server "books" (or fails to book) an acquisition on behalf of a client; acquisitions may also be booked automatically, in response to a user request to acquire a whole series, for example, or as part of a "push-VOD" service whereby the broadcaster or a third party supply the box with a list of content to be acquire.

We define "rescheduling" a content-acquisition to be the process of changing the `source-id` and `content-id` of the acquisition to those of a different piece of content which has the same `global-content-id` as that of the original piece of content. This will generally be done to avoid a conflict between two acquisitions.

2.6.2 Implementation

Each acquisition (whether a "content acquisition" or a "series acquisition") is identified within the API by an `acquisition-id`, of the form:

```
acquisition-id        = id-element
```

Each `acquisition-id` MUST be unique amongst the `acquisition-id`s on the box. The same `acquisition-id` MUST continue to be usable for addressing the same acquisition for as long as the box is switched on, and SHOULD persist for longer periods.

It is RECOMMENDED that servers which implement "series acquisitions" use a different naming convention for the `acquisition-id`s associated with"series acquisitions" to the one they use for `acquisition-id`s associated with "content acquisitions", such that the two can be automatically and trivially distinguished (eg the string prefixes `s` and `c` respectively).

Each time a piece of content with the same `series-id` as that associated with a booked "series acquisition" becomes acquirable the box SHOULD create a new "content acquisition" for the piece of content, unless there is a good reason not to (such as such an acquisition already existing for that piece of content, or the content already being stored on the box). Each time the box creates a "content acquisition" as a result of a "series acquisition" in this fashion it MUST set its "series-linked" property to "true"; all content-acquisitions created any other way MUST have their "series-linked" property set to "false".

If an acquisition was booked as a result of a direct request from a user, or as a result of a "series acquisition", the box SHOULD set the acquisition's "speculative" property to "false", otherwise it MUST set that property to "true".

Acquisitions which do not have the "priority" property set to "true" MAY be rescheduled automatically to avoid conflicts (if another location for the same content can be found), or if an opportunity to make the acquisition earlier arises, acquisitions which are "priority" acquisitions SHOULD NOT be so rescheduled.

It may be a the case that to book a requested acquisition would require resources that are already in use by an existing acquisition. For example, on a box which can only acquire one piece of content at a time from DVB sources, the user may request the acquisition of such a piece of content whose broadcast overlaps in time with an existing DVB acquisition. The behaviour of the box in this scenario will depend upon whether either of the acquisitions is considered a "priority" acquisition. The box SHOULD follow the steps indicated below:

1. if the new acquisition would not be a "priority" acquisition then the box SHOULD attempt to acquire the newly-requested content from a different source or at a different time, if doing so would resolve the conflict. If the box is able to do this then the booking of the new acquisition may proceed accordingly, and assuming the creation succeeds then the request to create the new acquisition is considered to have succeeded. Otherwise continue:

2. if the original acquisition is not a "priority" acquisition then the box SHOULD attempt to acquire the originally-requested content from a different source or at a different time, if doing so would resolve the conflict. If the box is able to do this then the original acquisition should be updated accordingly, and booking of the new acquisition may proceed. Otherwise, continue:

3. The request to create the new acquisition has failed: the original acquisition SHOULD NOT be altered, and the new acquisition SHOULD NOT be booked.

When a content acquisition finishes and the content becomes presentable, the acquisition MUST be removed from the list of acquisitions maintained by the box, and then a piece of content representing this content MUST be added to an appropriate source in the reserved "uc_storage" source list defined in section 4.15 (which SHOULD trigger a notifiable change in "uc/storage" if that resource has been implemented). Servers MAY have to generate a new `content-id` for the content at this point.

2.7 Source Modelling

This section is purely informative.

One of the most important decisions that a server implementer must make is how to map a particular set of media sources with a common kind of origin into the UC model of media components, content, sources and source lists. This section contains some examples of how this might be accomplished in certain cases:

- A DVB network such as Freeview in the UK could be modelled such that individual DVB services (colloquially known as "television channels") are represented in the UC model as individual sources in a single source list. DVB events (usually equivalent to the colloquial concept of "programmes") could be represented as pieces of content in the UC model, and DVB components could be represented by UC media components. The source would have the live, linear, and follow-on properties, and much of the content within it would be AV content and would have start times. If the service also carried MHEG interactive applications (or similar) then these would be represented as interactive content in the same source.

- A VOD service could be modelled as a single source, with each distinct stream selectable by the user represented as pieces of content in the UC model. If multiple streams of the same programme are available that differ only in bit-rate, or in the codecs used, then all such streams could be modelled as a single piece of content, with the appropriate stream option selected automatically by the server or box when a client selects the piece of content in question. A source list of all the VOD services known to the box could be provided.

- An application download service could be modelled as a single source with none of the live, linear, or follow-on properties, containing only interactive content. The content itself might have acquirable-from and acquirable-until times, and some pieces of content might be available for acquisition at any one time, but none would be available for presentation.

- A DVD player could be modelled as a single source with one or more pieces of content, perhaps with an piece of interactive content corresponding to the DVD's menu system and pieces of AV content representing the individual features on the disc. Both the source and content could be removed from any of the source lists that contain them if no DVD was present in the player.

- Recorded TV content on a hard disk inside the box can be represented as individual pieces of content associated with a source representing the hard disk in the "uc_storage" source-list that are reserved for this purpose: see section 4.15 for details.

- A DLNA-compliant Digital Media Server (DMS) on the same network as the box could be modelled as a single source, with each distinct media object in the DMS's ContentDirectory service represented as a piece of content in the UC model.

- A USB drive containing media files could be modelled as a single source that is dynamically added to a suitable source list (perhaps the reserved "uc_storage" source list described in section 4.15) when the drive is plugged into the box. Each media file on the drive that is in a format supported by the box for playback could be represented as an individual piece of content.

In general, one can consider the UC model to be hierarchical: a source list contains multiple sources (which may each appear in multiple source lists); a source contains multiple pieces of content; a piece of AV content contains multiple media components. When considering how to model a set of media sources, a recommended approach is for the server implementer to examine the kinds of metadata that entities at each level of this hierarchy can have associated with them and make mappings into the UC model accordingly.

3 Server Implementation Considerations

Throughout this document all server resources which may be valid targets for HTTP requests will be described using URIs relative to the root URI of the server (ie. a resource with a URI of "`http://192.168.0.2:48875/uc/foo/bar`" would be referred to with the relative URI "uc/foo/bar").

3.1 Server Technology Requirements

Any Universal Control Server MUST function as a fully compliant HTTP 1.1[8] server which SHOULD operate on port 48875.

Server implementors should give careful consideration to the values that the server returns for the Cache-Control header in response to GET requests to different resources.

The server MUST comply with the Cross Origin Resource Sharing[9] specification, and the "list of origins" required by that specification MAY be unbounded.

The server MUST implement a cross-domain policy file, as described in [10], which MUST be sent to clients in response to a GET request to the HTTP server for "/crossdomain.xml". The server SHOULD serve this document with the Content-Type header "text/x-cross-domain-policy". The file MUST permit access to the API by requests from applications served from the same origins as those from which the server permits access via CORS-compliant requests, and SHOULD NOT permit access from any other origins.

A server SHOULD NOT implement any form of HTTP authentication. The server uses its own security scheme (see section 3.3 below).

A server MUST implement advertisement of itself as a service via the DNS-SD[11] format carried via Multicast DNS[12]. See section 3.2 below.

3.2 Server Discovery

3.2.1 Introduction

This section is purely informative. For normative details, see subsequent sections.

The UC API defines two "discovery" mechanisms, by which clients receive the information they need to know in order to connect to the server. It also defines an optional security scheme (see Section 3.3). Servers must implement both discovery mechanisms, and may implement the security scheme if the implementer so desires.

The two discovery mechanisms are as follows:

- Automatic discovery: the client discovers the IP address and port of the server via DNS-SD carried over Multicast DNS.

- Manual discovery: the user obtains a "pairing code" from the server's built-in user interface, which encodes the server's IP address and port. If the server implements the security scheme, then the pairing code also encodes a "Short Shared Seceret" (SSS), which is an opaque value chosen by the server (randomly or pseudo-randomly), and can be used as part of the pairing mechanism described in Section 3.3. The user enters the pairing code into the client, which decodes the server's IP address and port (and possibly the SSS) from it.

If the server implementer chooses not to implement the security scheme, then the server will accept and process requests from any client on the home network. Otherwise, clients must authenticate themselves when requested to do so by the server. The authentication mechanism to be used is designed to be similar to HTTP digest [13] authentication.

To clarify how the security and discovery mechanisms defined in this API interact, the process of setting up a new client is described below, for each combination of discovery mechanism and security scheme presence, in order of increasing complexity.

Automatic Discovery of an Unsecured Server

If the client can use automatic discovery, and the server is unsecured, the process of setting up a new client is as follows:

1. The client obtains a list of servers on the network using DNS-SD carried over Multicast DNS.

2. The client presents the list of servers to the user; the user identifies the correct server from the list.

3. The client opens a connection to the server using the IP address and port obtained from the Multicast DNS service.

This scenario gives the user the best possible experience, and requires the fewest steps.

Manual Discovery of an Unsecured Server

If the client cannot use automatic discovery, and the server is unsecured, the process of setting up a new client changes:

1. The client requests a pairing code from the user.

2. The user requests a pairing code from the box using the box's built-in user interface.

3. The server/box generates a pairing code containing only its IP address and port, and conveys them to the user, perhaps by displaying them on a screen.

4. The user enters the pairing code into the client.

5. The client opens a connection to the server using the IP address and port decoded from the pairing code.

This process requires the user to obtain and enter a pairing code from the box. This will be more challenging for some users than selecting the server from a list, and hence we recommend that clients use automatic discovery whenever possible. Unfortunately, automatic discovery is not currently possible on some key platforms, including basic implementations of Java 2 Mobile Edition (J2ME), and AJAX applications running in web browsers.

Manual Discovery of a Secured Server

If the client cannot use automatic discovery, and the server is secured, the process of setting up a new client is different again:

1. The client requests a pairing code from the user.
2. The user requests a pairing code using the box's built-in user interface.
3. The server generates a pairing code containing its IP address, port and a newly generated random SSS, and conveys them to the user, perhaps by displaying them on a screen.
4. The user enters the pairing code into the client.
5. The client opens a connection to the server using the IP address decoded from the pairing code.
6. The client and server perform the pairing process described in Section 3.3.

Here, the user experience benefits of combining the authentication and discovery information into a single code are obvious.

Automatic Discovery of a Secured Server

If the client can use automatic discovery, and the server is secured, the process of setting up a new client is as follows:

1. The client obtains a list of servers on the network using DNS-SD carried over Multicast DNS.
2. The client presents the list of servers to the user; the user identifies the correct server from the list.
3. The client opens a connection to the server using the IP address and port obtained from the Multicast DNS service, and makes a request to the "uc" resource.
4. The client determines from the result of that request that the server implements the security scheme, and requests that the user enter a pairing code.
5. The user requests a pairing code using the box's built-in user interface.
6. The server generates a pairing code containing its IP address and port and a new randomly-generated SSS, and conveys them to the user, perhaps by displaying them on a screen.
7. The user enters the pairing code into the client.
8. The client and server perform the pairing process described in Section 3.3.

This process is the most complex, and requires the user both to select the correct server from a list, and to obtain and enter a pairing code.

Technically, this process involves a redundant step, as the IP address and port of the server are conveyed both by DNS-SD and in the pairing code. Eliminating the redundancy would involve either eliminating the DNS-SD look-up, or removing the IP address and port from the pairing code. Retaining the redundancy results in the best possible user experience, however, for the following reasons:

Firstly, DNS-SD carried over Multicast DNS gives clients a way to reconnect to servers when the server's IP address changes without requiring the user to enter another pairing code, so retaining it is an advantage on home networks, where IP addresses are typically assigned by DHCP and may change frequently. More generally, mandating support for it in servers allows clients that support the technology to use information about the servers available on the network at any given time to provide an optimal user experience in many different scenarios.

Secondly, removing the IP address and port from the pairing code would mean that only clients capable of automatic discovery could connect to the server, and displaying more than one code to the user (one code for discovery and a separate code for authentication, for example) would increase the complexity of the scheme from the perspective of the user.

Finally, if the IP address and secret are included in the same code, the client can use the IP address as an additional check to determine whether or not the pairing code was obtained from the desired server or not.

3.2.2 Implementation of Discovery Schemes

Automatic Discovery

A server MUST advertise its existence via multicast DNS [12] according to the DNS-SD standard [11], using the protocol name `_universalctrl._tcp` in the `.local` domain. All servers SHOULD advertise with weight and priority 0, MUST use the server name (see section 3.5) as the service name, and MUST provide a textual property named "server_id" which contains a textual representation of the server's unique ID (see the definition in section 3.5). All servers SHOULD provide a textual property named "path" which if included MUST be the absolute URI of the Universal Control "uc" resource for that server (see section 4.3).

Manual Discovery

A server MUST support the concept of a "pairing code" and provide a means to convey pairing codes directly to the user. Exactly what a pairing code is, what information it contains and how pairing codes are managed are detailed in this section from a server perspective. See section 5.5 for a client perspective.

A pairing code is a string representation of a binary code generated from the server's IP address, the port on which the server accepts client HTTP requests, and (if the server supports the security scheme) an 8-bit SSS. How the binary code is generated is described in Appendix A. The string representation is created by converting the binary code to base-32, using letters and digits as described in Appendix B. The box MAY include leading `0`s in this representation. If the code itself is 0 then the box MUST represent it with a string containing nothing except one or more `0`s. The box MUST NOT present an empty string to the user for use as a pairing code.

The device on which a server runs MUST provide a mechanism using its in-built interface to "present" a pairing code to the user when the user requests one. One such mechanism is the display of a dedicated "pairing screen" on a visual display built into the device or connected to it, such as a television. If the server implements the security scheme described in this specification then when the pairing code is presented to the user, the server MUST generate a new 8-bit random SSS and include it in the pairing code.

3.3 The Security Scheme

The Security Scheme described here is OPTIONAL, the rest of this section assumes it is being implemented.

3.3.1 Secure Pairing

This section (3.3.1) is purely informative.

The mechanism described below is designed to establish a simple but sufficiently secure optional authentication mechanism such that a server is able to ensure that all requests made to it originate from a client which has been authorised to connect to it by someone with physical access to the box.

To implement this scheme the server will need to maintain a list of authenticated clients, each of which is associated with three pieces of information that the server stores persistently:

- A machine readable "client-id" consisting of an 128-bit unsigned integer which is always presented as a string of the format specified for UUIDs in RFC 4122 (see [14]).
- A human readable "client-name" consisting of a string which is between 0 and 63 characters long, and in ABNF is specified as:

```
client-name = 1*( unreserved / pct-encoded )
```

- A "Large Server-generated Secret" (LSGS) consisting of a string of 64 octets generated by a cryptographically strong pseudo-random number generator.

In addition the server will need the ability to generate a non-predictable "Short Shared Secret" (SSS), which takes the form of an 8-bit integer, and be able to present this integer to the user as part of a pairing-code (see Section 3.2). A new SSS will need to be generated each time a new pairing code is presented to the user. The SSS SHOULD also be generated by a cryptographically strong pseudo-random number generator.

3.3.2 The Pairing Process

The pairing process for a server which supports the Security Scheme proceeds as follows:

1. The user commands the server to present the pairing code using the box's built-in interface.
2. The server generates a SSS and uses it to construct a pairing code which is then presented to the user.
3. The user enters the pairing code into the client device.
4. The client device decodes the pairing code extracting the SSS and the IP address of the server.
5. The client makes a request to the resource 'uc/security' as described in section 4.4, supplying a client-id and a client-name.
6. Upon receiving this request the server takes down the pairing code from the box's built-in interface.
7. The server generates an LSGS and temporarily associates it with the client-id and client-name specified, then encrypts it using the SSS (as described below) and returns it to the client. If this LSGS is not used for authentication (as described below) in a short period of time it will be automatically removed from the list of valid credentials.
8. The client decodes the LSGS using the SSS and makes a request to the server using the LSGS as specified below.
9. If the LSGS used to authenticate the client's request is correct then it is permanently associated with the client-id and client-name (unless de-authorised by user action). If it is incorrect then the box notifies the user using its built-in user interface. The box must not automatically return to step 2 without pro-active interaction by the user with the built-in user interface such as using button presses to confirm the notification or re-command the box to display a new pairing code. otherwise clients may be able to implement brute force attacks.
10. All further client requests are then made using the LSGS as described in Section 3.3.3.

3.3.3 Server Request Responses

Unless specified otherwise in the description of the individual resource (such as for GET requests to 'uc' and POST requests to 'uc/security') all requests which may be made to the server which do not automatically return a 405 response are considered to require client authentication.

When a server receives a request which requires client authentication it should immediately check the request's `X-UCClientAuthorisation` header, and validate the information contained it it according to the algorithm described below. If the request does not have such a header or the validation fails then the server MUST immediately follow the following procedure:

1. The server generates a nonce value which is a string containing only lowercase hexadecimal characters and MUST be at least 40 characters long. The generation of a nonce value is left up to server implementations, but MUST be non-predictable, and each nonce-value MUST be valid only for a limited period of time, either because it is stored after generation, or because the generation algorithm uses a truncated timestamp as part of the generation requirements. It is not required that each nonce be useable only for one request, but this is permissible and more secure (see RFC 2617 Section 4.3[13] for a discussion of the use of nonces in a similar scheme).

   For example a valid mechanism for generating a nonce would be

   $$nonce = timestamp \sqcup \mathrm{SHA1}(timestamp \sqcup \text{“:”} \sqcup method \sqcup \text{“:”} \sqcup uri \sqcup \text{“:”} \sqcup private\ data)$$

   where *timestamp* is a truncated numeric timestamp represented as a 10 digit hexadecimal string, *method* is the method of the request as an octet string, *uri* is the relative uri of the resource, *private data* is some private data known only to the server, and $\sqcup$ implies a string concatenation operation.

   A nonce of this form can be easily checked for validity by using the timestamp included in it to reform it and checking if they match, whilst it can be checked for staleness by simply checking if the timestamp is recent enough.

2. The server sets a stored nonce-count value for the generated nonce if one doesn't already exist, and sets it to 1. If nonces are useable once only then this value may be hard-coded to always be 1.

3. The server selects an appropriate iteration count for the authentication. Higher values slow the response from clients but increase security, it is recommended that the server use a value between 1 and 10 for this parameter.

4. The server returns a 402 error with an `X-UCClientAuthenticate` header as described below.

The format of an `X-UCClientAuthenticate` header is as follows:

```
challenge        = "Authenticate" 1*WSP digest-challenge

digest-challenge = "nonce" "=" <"> 40*( LCHEXDIG ) <"> "," *WSP
                   "iteration" "=" <"> 8LCHEXDIG <">
                   [ "," *WSP "stale" "=" ( "true" / "false" ) ]
```

where the "stale" parameter SHOULD be absent or set to "false" except in circumstances where the 402 error is being returned due to the fact that a client request was made with an out-of-date nonce value, in which case it MUST be "true".

Upon receiving a 402 error with an `X-UCClientAuthenticate` header a client SHOULD follow the following procedure:

1. The client generates a client nonce value, the generation of which is entirely implementation dependent, but it MUST be at least 40 characters long and contain only lowercase hexadecimal digits, and it SHOULD be unpredictable.

2. The client increments the integer nonce-count ("nc") it has stored for this particular nonce and uses the new value, or if it has no stored nonce-count for this nonce (because it's the first time it's been used) it stores the value 1 and uses that.

3. The client uses the client nonce value, the nonce-count, its 128-bit client-id and the data provided by the server to generate a 20 octet digest for the request according to the algorithm described in the next section.

4. The client repeats the request with an `X-UCClientAuthorisation` header as described below.

The format of an `X-UCClientAuthorisation` header is as follows:

```
credentials      = "Authenticate" 1*WSP digest-response

digest-response  = "nonce" "=" <"> 40*( LCHEXDIG ) <"> "," *WSP
                   "iteration" "=" <"> 8LCHEXDIG <"> "," *WSP
                   "uri" "=" <"> URI <"> "," *WSP
                   "digest" "=" <"> 40LCHEXDIG <"> "," *WSP
                   "nc" "=" <"> 8LCHEXDIG <"> "," *WSP
                   "client-id" "=" <"> client-id <"> "," *WSP
                   "cnonce" "=" <"> 40*( LCHEXDIG ) <">
client-id    = 8LCHEXDIG "-" 4LCHEXDIG "-" 4LCHEXDIG "-"
               4LCHEXDIG "-" 12LCHEXDIG
```

where the "uri" parameter is the URI for the resource to which the request is being made, and all other parameters are set as described above.

Upon receiving a request which requires client authentication and has a correctly formatted `X-UCClientAuthorisation` header the server MUST immediately follow the following procedure to validate the authentication credentials provided:

1. The server MUST reject any request for which the "uri" value is not recognised as referring to the same resource as the URI to which the request was made.

2. The server MUST reject any request for which the server has no LSGS stored for the specified client-id.

3. The server MUST reject any request for which the "nonce" value is not recognised as either currently valid, or having been valid in the past.

4. The server MUST reject any request for which the "iteration" value is not the one it specifies in challenges.

5. If the nonce-count provided is not one which has already been used for this nonce then the server MUST replace the stored nonce-count with the new one if one is stored, or render the nonce invalid for further requests if each nonce is one-time-only.

6. The server MUST calculate a digest using the data provided in the request and the LSGS using the algorithm described below. If this digest does not match the one provided in the client's request the server MUST reject the request.

7. If the digests do match and the "nonce" value is currently valid then the server MUST accept the request.

8. If the digests match but the "nonce" value is one which is no longer valid or the nonce-count is one which has already been seen for this nonce then the server MUST reject the request, but MAY set the "stale" parameter to "true" in the challenge response as a clue to the client that the request may be retried with the new nonce-value. The server MUST do this for the most recently replaced nonce for a particular request, and MAY do this for other expired nonces at the discretion of the server implementor.

where "rejecting" the request means that the server MUST immediately return a 402-error with a `X-UCClientAuthenticate` header as described above.

3.3.4 The Digest Algorithm

The mechanism for calculating the digest is to apply the PBKDF2-HMAC-SHA1 algorithm (see [15, Section 5.2]) to the following parameters:

- `P`: The LSGS as a string of octets.
- `S`: The string formed as follows:

  ```
  method ":" uri ":" nonce ":" message-body ":" nonce-count ":" cnonce
  ```

  where "method" is the HTTP method used in the request, "uri" is the URI to which the request will be made (for the client) or the URI provided by the client (when the calculation is done by the server), "nonce" is the nonce value provided in the challenge as a string of lowercase hexadecimal digits (exactly as it is provided in the challenge), "cnonce" is the client generated nonce value as a string of lowercase hexadecimal digits (exactly as it is provided in the challenge), "nonce-count" is the nonce-count for this nonce-value as a string of 8 lowercase hexadecimal digits (exactly as the client provides it in the request), and "message-body" is the entire body of the request (but not any headers) exactly as it appears in the request.
- `c`: The iteration count specified in the challenge
- `dkLen`: Always 20 octets.

Library functions for implementing this algorithm exist for most platforms, and MAY be used where available without modification. For the convenience of developers implementing the algorithm themselves we explain in Appendix C how to implement it using only standard logical operators and the standard SHA-1 hashing function.

3.3.5 De-authorisation of Clients

If the server implements the security scheme then the box SHOULD provide a mechanism through its built-in interface for listing authorised clients and allow them to be de-authorised by the user. On this screen the `client-id` and pairing-code MUST NOT be shown, instead each client MUST be identified by the associated client name (with percent encoding undone as usual).

A server which implements the security scheme MAY also implement the "uc/credentials" resource and allow the de-authorisation of devices through DELETE requests to sub-resources of that resource. If it does so then the server MUST allow a particular `client-id` to be de-authorised in response to a request authenticated using that `client-id`, all other restrictions are left up to the server implementation. See section 4.42 for details.

3.4 Request Restriction

The restriction scheme described here is OPTIONAL, the rest of this section assumes it is being implemented.

The purpose of this scheme is to allow the server implementor to require confirmation or authorisation from the user before it proceeds. This is particularly useful when the request may result in unexpected side-effects, or when the request requires the provision of a secret "PIN" known to the user in order to authorise it (such as the acquisition of pay-per-view content).

3.4.1 Overview

This section is purely informative.

A particular request to a particular resource may be declared "restricted" according to any criterion that the server developer desires. It is intended that the server developer make only requests which require individual authorisation (such as the booking of the acquisition of pay-per-view content) restricted, and not that all requests be considered restricted.

There are two types of restriction which a request may have upon it:

- A request may require *confirmation*, requiring only the agreement of the user to continue.
- A request may require *authorisation*, requiring that the user enter a pre-shared decimal PIN to confirm their identity before proceeding.

In all cases the server has the option of sending a specific message for the client to display to the user whilst asking for their confirmation, or authorisation, and also of presenting the same information to the user via the box's built-in interface.

3.4.2 Server Restricted Request Responses

All restricted requests MUST also comply with the provisions of Section 3.3.3 if the server implements the security scheme.

If the security scheme is not implemented, or if it is implemented and this request has already been correctly authenticated, and the request is restricted then the server MUST immediately check the request for a valid `X-UC-Restriction-Credentials` header as described later in this section. If one is not found (which will usually be the case unless a challenge has already been made) then the server MUST immediately follow the following procedure:

1. The server does not immediately perform the action normally associated with the restricted request, and sets a timeout (the length of which is left up to the server implementor and may vary between resources). The intention is that if the action is not correctly confirmed or authorised before the timeout expires then a default behaviour will occur. In most circumstances the server will do nothing when the timeout occurs, but in some it may continue with the original request.

2. The server generates a nonce value which is a string containing only lowercase hexadecimal characters and MUST be at least 40 characters long with the same restrictions and recommendations as for the nonces used for the security scheme (see Section 3.3.3). The nonce used for this purpose MUST NOT be the same nonce used for security scheme-related authentication challenges for the same request.

3. If the request is to require authorisation then the server selects an appropriate iteration count. Higher values slow the response from clients but increase security, it is recommended that the server use a value which is at least 5000 for this parameter.

4. If the request is to require authorisation then the server ensures that it has access to an appropriate decimal PIN known to the user which can be used to authorise this request. Different requests MAY require different PINs. The PIN(s) SHOULD be four digits or longer in length.

5. The server generates an appropriate message to present to the user to explain why confirmation or authorisation is required.

6. The server returns a 402 error with an `X-UCRestriction-Challenge` header as described below.

The general format for a `X-UCRestriction-Challenge` header is as follows:

```
challenge  = confirm / authorise

confirm    = "Confirm" 1*WSP
             "nonce" "=" <"> 40*( LCHEXDIG ) <">
             "," *WSP "message" "=" quoted-string
authorise  = "Authorise" 1*WSP
             "nonce" "=" <"> 40*( LCHEXDIG ) <">
             "," *WSP "message" "=" quoted-string
             "," *WSP "iteration" "=" <"> 8LCHEXDIG <">
```

where the "nonce" quoted string is the nonce in lowercase hexadecimal representation, the "message" parameter is a quoted string which is to be conveyed to the user via the client, and the "iteration" parameter is a lower-case hexadecimal representation of the number of iterations selected.

Upon receiving a 402 error with an `X-UCRestriction-Challenge` header a client SHOULD examine whether the server has asked for Confirmation or Authorisation, and take action accordingly as set out below. In both cases, however, the client SHOULD eventually repeat the request with a valid `X-UCRestriction-Credentials` header of the following form:

```
credentials    = confirmation / authorisation / abort

confirmation   = "Confirm" 1*WSP
                 "nonce" "=" <"> 1*( LCHEXDIG ) <">
authorisation  = "Authorise" 1*WSP
                 "nonce" "=" <"> 40*( LCHEXDIG ) <"> "," *WSP
                 "iteration" "=" <"> 8LCHEXDIG <"> "," *WSP
                 "uri" "=" <"> URI <"> "," *WSP
                 "digest" "=" <"> 40LCHEXDIG <">
                 ["," *WSP "client-id" "=" <"> client-id <"> ]
abort          = "Abort" 1*WSP
                 "nonce" "=" <"> 1*( LCHEXDIG ) <">
```

Confirmation If the server has requested Confirmation then the client SHOULD follow the following procedure:

1. The client SHOULD present the message included in the challenge to the user along with a choice of whether to continue with the action or abort it.

2. If the user chooses to continue with the action then the client SHOULD repeat the original request with a `X-UCRestriction-Credentials` header using the `confirmation` option.

3. If the user chooses to abort the request then the client SHOULD repeat the original request with a `X-UCRestriction-Credentials` header using the `abort` option.

If the server receives a restricted request which requires confirmation with a correctly formed `X-UCRestriction-Credentials` header then the server MUST parse it. If it uses the `authorise` option, or includes a "nonce" value other than one which the server has generated as part of a challenge for the same request to the same resource then the server MUST immediately return a 402 error with no `X-UCRestriction-Challenge` header. If the request does include a valid nonce for the request, but the time for which that nonce was valid has now expired or the nonce has been rendered invalid by other means then the server MUST also immediately return a 402 error with no `X-UCRestriction-Challenge` header.

If the server receives such a request using the `confirmation` option with a nonce which is still valid for this request then the server MUST immediately proceed by processing the request as normal, returning whatever status code the normal processing would require and MUST immediately render that nonce invalid for further requests.

If the server receives such a request using the `abort` option with a nonce which is still valid for this request then the server MUST NOT process the request, even if that would be the normal default behaviour when the timeout occurs, and MUST immediately render the nonce invalid for further requests, it MUST then immediately return a 410 response to the request.

If the timeout is reached for a particular nonce and the nonce has not already been rendered invalid for further requests then the server MUST immediately render it invalid for further requests and MUST continue with it's normal default behaviour, which will normally be to take no further action, but MAY be to process the original request as normal, but return no data to the client (since the connection will already have been closed).

Authorisation If the server has requested Authorisation then the client SHOULD follow the following procedure:

1. The client SHOULD present the message included in the challenge to the user along with a prompt to enter a numeric PIN, and an option to abort the action.

2. If the user chooses to enter the PIN then the client SHOULD immediately construct a 40 character string consisting of the lower-case hex representation of a digest calculated using the mechanism described later in this section, and then repeat the original request with a `X-UCRestriction-Credentials` header using the `authorisation` option providing the nonce and iteration count from the challenge, the URI of the request itself, and the digest just calculated. If the client made use of an LSGS in calculating the digest then it MUST include its client-id in the `X-UCRestriction-Credentials` header, and if it did not then it MUST NOT.

3. If the user chooses to abort the request then the client SHOULD repeat the original request with a `X-UCRestriction-Credentials` header using the `abort` option.

If the server receives a restricted request which requires authorisation with a correctly formed `X-UCRestriction-Credentials` header then the server MUST parse it. If it uses the `confirmation` option, or includes a "nonce" value other than one which the server has generated as part of a challenge for the same request to the same resource then the server MUST immediately return a 402 error with no `X-UCRestriction-Challenge` header. If the request does include a valid nonce for the request, but the time for which that nonce was valid has now expired, the nonce has been rendered invalid by other means, or the iteration count listed in the header does not match the iteration count used in the original challenge then the server MUST also immediately return a 402 error with no `X-UCRestriction-Challenge` header. If the header includes a client-id, but no LSGS is stored by the server for that client-id then the server MUST immediately return a 402 error with no `X-UCRestriction-Challenge` header. Otherwise the server MUST immediately calculate a digest using the algorithm described later in this section (using its stored LSGS if a client-id was

included) and compare it to the digest provided in the header. If the values do not match then the server MUST immediately return a 402 error with no `X-UCRestriction-Challenge` header.

If the server receives such a request using the `authorisation` option with a nonce which is still valid for this request, iteration counts which match, and a digest which validates correctly then the server MUST immediately proceed by processing the request as normal, returning whatever status code the normal processing would require and MUST immediately render that nonce invalid for further requests.

If the server receives such a request using the `abort` option with a nonce which is still valid for this request then the server MUST NOT process the request, even if that would be the normal default behaviour when the timeout occurs, and MUST immediately render the nonce invalid for further requests, it MUST then immediately return a 410 response to the request.

If the timeout is reached for a particular nonce and the nonce has not already been rendered invalid for further requests then the server MUST immediately render it invalid for further requests and MUST continue with it's normal default behaviour, which will normally be to take no further action, but MAY be to process the original request as normal, but return no data to the client (since the connection will already have been closed).

The mechamism for calculating the digest is to apply the PBKDF2-HMAC-SHA1 algorithm (see [15, Section 5.2]) to the following parameters:

- `P`: The string formed as follows:

  ```
  PIN [ ":" LSGS ]
  ```

  where "PIN" is the string formed by taking the decimal representation of the PIN entered by the user (for the client) or the pre-shared PIN stored on the box (for the server), and "LSGS" is a valid LSGS obtained by this client for use with this server (for the client) or the LSGS stored by the server for the client-id which the client specified in its request (for the server). If the client has access to such an LSGS then it SHOULD use it, the server MUST use the LSGS if the client included a client-id and MUST NOT otherwise.

- `S`: The string formed as follows:

  ```
  method ":" uri ":" nonce ":" message-body
  ```

  where "method" is the HTTP method used in the request, "uri" is the URI to which the request will be made (for the client) or the URI provided by the client (when the calculation is done by the server), "nonce" is the nonce value provided in the challenge, and "message-body" is the entire body of the request (but not any headers).

- `c`: The iteration count specified in the challenge

- `dkLen`: Always 20 octets.

3.5 Server Identification

Each distinct UC server MUST retain two pieces of information for use by clients in identifying it: a human-readable server name and a globally-unique `server-id`, defined below. Two or more UC servers are distinct if they control different underlying hardware, or control the same hardware but have different information in their persistent storage, such as different authenticated client lists.

Each server name MAY be any human-readable string, and SHOULD be chosen to make it as easy as possible for the user to identify the server in question from a list of all the UC servers on the network. Since this string will be used as a DNS-SD service name (see [11]) the name MUST be unique on the local network, but need not be unique globally. This local uniqueness can be achieved

by appending digits to the end of duplicate names if required, though it is RECOMMENDED that the server instead uses an alternative name which is more meaningful to users, perhaps by allowing a user to configure the name through the box's built-in interface. The server SHOULD NOT include a representation of the `server-id` in its name. The name MUST be a UTF-8 string between 0 and 63 characters long which MAY contain any character.

Each `server-id` MUST be a 128-bit integer. Distinct UC servers MUST have different `server-id`s. It is intended that client software be able to use a `server-id` to identify a given server uniquely, regardless of the network to which the server or client may be connected to at any given time, and the `server-id` therefore SHOULD NOT change in ways that break this assumption. For example, the `server-id` MUST NOT change when the IP address of the server changes, and MUST NOT change merely because the server has been power-cycled. It is RECOMMENDED that the `server-id` be a UUID as specified in RFC 4122[14], and the guarantees of uniqueness contained therein are considered sufficient to satisfy the relevant foregoing requirements. When presented in a textual form (such as in the DNS-SD record for the server or in the `sid` attribute of the `ucserver` element in the representation returned by the `uc` resource) the server MUST represent the `server-id` in the UUID string representation format described in RFC 4122[14], furthermore the server MUST use only lowercase letters for this representation.

4 Specification of Server Resources

Throughout these descriptions a resource "representation" is defined to be an XML document in the UTF-8 character set conforming to the Universal Control schema (see Appendix D), which contains a single top-level element of type `response`. When a client makes a request to a server that does not result in an error, such a representation MUST be returned as an HTTP response body with content-type `application/xml`. See below for additional recommendations on the format of response bodies.

When a client request results in an error, the server's response to that request MUST take the form of an HTTP response body with content-type `application/xml` conforming to the Universal Control XML schema, which contains a single top-level element of type `error`. The server MUST return this error document instead of a valid resource representation as defined in the following paragraph. The server MUST set the `code` attribute of this element to the same integer value as the HTTP response code being returned. The server MAY include a human readable error message as plain text inside the `error` element.

When returning a representation as a response to a client request the server MUST set the `resource` attribute of the `response` element to the canonical path of the resource to which the request was made, including any query parameters, as a URI relative to the root URI of the server (ie. if the full canonical URI of the resource is "http://192.168.0.2:48875/uc/foo/bar?q=baz" then the relative URI used will be "uc/foo/bar?q=baz")[3]. The relative URI MUST begin with "uc" and have any "dot segments" removed using the algorithm defined in section 5.2.4 of RFC 3986 [5]. Unless specified otherwise the response code generated by the server when returning a representation MUST be `200`.

A UC-enabled box is defined for the purposes of this specification to be "switched on" when electrical power is being supplied to all the box components that the server needs to interact with when handling client requests. A box is defined to be "switched off" when power is not being supplied to any such component. A box is defined to be in a low-power "standby" mode when power is being supplied to the resources the server needs to respond to HTTP requests, but the box does not meet the definition of "switched on". Unless otherwise noted, the resource definitions in sections 4.3 onwards MUST be complied with whenever the server is switched on. When the box is switched off the server MUST NOT (and cannot) respond to any client request. When the

[3]See section 4.1 for details of an additional restriction that applies if the HTTP verb is being overridden by means of a query parameter.

box is in standby mode the server MAY be running, and if it is then it MUST respond with a 400 error to all client requests except where noted in individual resource descriptions (in particular, the "uc/power" resource described in section 4.6).

If the server receives a request to a resource which does not exist on the server (such as an OPTIONAL resource which has not been implemented) then the server MUST return a `404` error. In response to a request made to a resource which does not respond to the verb specified in the request, the server MUST return a `405` error.

A server that complies with this specification MUST NOT implement resources with the URL path element prefix "uc/" other than those defined in this document. (This prohibition does not apply to resources implemented as part of the API extension mechanism described in section 4.44.)

Normative descriptions of the XML format of resource representations are given in this section. In some cases, different restrictions apply to the representations sent to clients and those received from them by a server. These are noted in the text. In addition, an XML Schema description of the resource representations is given in Appendix D, which is also normative. Implementers must refer to both sources in order to create compliant implementations.

Servers MAY represent boolean values described in this specification using either "true" or "1" to represent boolean TRUE and either "false" or "0" to represent boolean FALSE. Servers MUST accept either form in client requests.

Servers MUST parse URL query strings using the character "&" to separate parameter/value pairs, and the character "=" to separate each parameter from its value. For example, the URL "`http://stb.example/uc/example_resource?a=b&c=d`" should be interpreted as containing a parameter "a" with value "b" and a parameter "c" with value "d".

The following restrictions on XML generated by the server are RECOMMENDED in order to ensure that the XML sent over the network is small, reducing bandwidth, CPU and power utilisation:

The server SHOULD NOT include namespace information in the XML it generates. The server SHOULD NOT include an `<?xml ... ?>` tag at the start of the XML it generates. The server SHOULD NOT produce XML with any non-required white-space characters. When whitespace is required it SHOULD consist of a single character. To improve readability, example XML in this specification does not follow these guidelines on whitespace.

4.1 Overriding the HTTP Method on a Request

The server MUST treat any request with a query parameter of `method_` as being a request of the type specified by the value of that query parameter, regardless of what the verb used in the request actually is. For example the server MUST treat a POST request with a query parameter `method_=PUT` as a PUT request. This requirement exists to allow the implementation of client software on platforms which restrict the choices of HTTP verb a request can be sent with. The query parameter and its value MUST be ignored in any and all processing of the query parameters in a client request, and MUST NOT be included in the query string returned to the client in the value of the `resource` attribute of the `response` element, as defined in section 4.

4.2 Notifiable Changes

Some resources are described below as being capable of undergoing "notifiable changes". Some resources are required to undergo notifiable changes under some circumstances, others MAY. If any resource is implemented in such a way that it can undergo notifiable changes then the "uc/events" resource MUST be implemented. In this case each time the criteria for a notifiable change in another resource occur, it is REQUIRED that the server update certain retained data used when processing requests to the "uc/events" resource, and the server MUST respond to all requests which have been made to that resource and not yet responded to if it is able.

This requirement is explained in detail in section 4.5.

Resources MAY generate notifiable changes in circumstances other than those explicitly mentioned in the resource descriptions, unless forbidden to by the resource description. This is left up to individual server implementations.

4.3 The uc Resource (required)

This resource MUST be implemented by all servers. Representations of this resource MUST comply with the schema for a response element containing a `ucserver` element defined in Appendix D.

The server MUST respond to requests to this resource without asking for or requiring authentication even if the server supports the security scheme. This overrides any other requirement for resources to issue authentication challenges.

The server MUST respond to a GET request to this resource by returning a representation containing an `ucserver` element such that:

- The `ucserver` element MUST have `name` and `server-id` attributes which MUST be set to the name and `server-id` of the box (see section 3.5).
- If the server implements the security scheme then the `ucserver` element MUST have a `security-scheme` attribute which MUST be set to `true`.
- If the server does not implement the security scheme then the `ucserver` element MAY have a `security-scheme` attribute. If present this attribute MUST be set to `false`.
- The `ucserver` element MUST have a `version` attribute which MUST be set to the version number given in the version of this document that the server complies with (see section 1.1).
- The `ucserver` element MAY have a `logo-href` attribute, which, if present, MUST be set to the URL of an image which can be used to represent the server (which SHOULD have a scheme of either `http` or `https`). This specification does not make recommendations regarding the format, size or quality of these images, or of other images mentioned herein.
- The `ucserver` element MUST contain a `resource` element for each of the OPTIONAL resources "uc/events", "uc/time", "uc/power", "uc/outputs", "uc/remote", "uc/feedback", "uc/source-lists", "uc/categories", "uc/sources", "uc/search", "uc/acquisitions", "uc/storage", "uc/credentials" and "uc/apps" which the server implements.
- Each `resource` element MUST have an `rref` attribute set to the relative URI of the resource it represents.

If the "uc/events" resources is not also implemented then this resource MUST be implemented in such a way that it never undergoes a notifiable change (even if the data which backs up its representations can change). If the "uc/events" resource is also implemented then it MUST undergo a notifiable change whenever the server's name, `server-id`, support for the security scheme, or supported optional resources change, and also whenever the URL of the image to be used as a logo for the box changes.

When the box is in standby mode, if the server implementation is still functioning then the server SHOULD respond normally to requests made to this resource.

An example XML document that fulfils the requirements for a representation of the "uc" resource is as follows:

```
<response resource="uc">
  <ucserver name="Example XML Server"
        logo-href="http://stb.example/logo.png"
        server-id="f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
        security-scheme="true"
```

```
        version="0.6.X">
    <resource rref="uc/events"/>
    <resource rref="uc/power"/>
    <resource rref="uc/time"/>
    <resource rref="uc/outputs"/>
    <resource rref="uc/remote"/>
    <resource rref="uc/feedback"/>
    <resource rref="uc/sources"/>
    <resource rref="uc/source-lists"/>
    <resource rref="uc/search"/>
    <resource rref="uc/categories"/>
    <resource rref="uc/acquisitions"/>
    <resource rref="uc/apps"/>
    <resource rref="uc/storage"/>
    <resource rref="uc/credentials"/>
  </ucserver>
</response>
```

4.4 The uc/security Resource (required)

This resource MUST be implemented by all servers which implement the security scheme and MUST NOT be implemented by any servers which do not. Representations of this resource MUST comply with the schema for a response element containing a "security" element defined in Appendix D.

Upon receiving a properly authenticated GET request to this resource the server MUST immediately return a 204 response. An improperly authenticated GET request is treated as normal for an improperly authenticated request.

POST requests to this resource are an exception to the normal Client Authentication rules, and any `X-UCClientAuthorise` headers included in them MUST be ignored by the server.

Upon receiving a POST request to this resource whilst there is no pairing-code being presented to the user using the box's built-in interface the server MUST immediately return a 404 error.

Upon receiving a POST request to this resource whilst a pairing-code is being presented without a single query-parameter "client-id" and a single query-parameter "client-name" the server MUST immediately return a 400 error.

Upon receiving any other POST request the server MUST immediately parse the "client-id" as a string representation of an 128-bit number conforming to the format specified for UUIDs in RFC 4122 (see [14]), and MUST immediately parse the "client-name" parameter as a percent-encoded string. If this is impossible the server MUST immediately return a 400 error.

The server MUST then generate a new random 64-octet LSGS, and MUST then temporarily store the client-name and LSGS for the given client-id, and MUST stop the presentation of the pairing-code on the box's built-in interface.

The server MUST then generate an "encrypted-LSGS" (E-LSGS) by performing a logical XOR of each octet of the LSGS with the octet which makes up the currently active SSS.

The server MUST then return a representation containing a `security` element such that:

- The `security` element MUST have a single `key` attribute, the value of which is the 256 character string formed by giving the lowercase hexadecimal representation of the E-LSGS.

The combination of "client-id" and LSGS generated here MUST remain valid for making requests to the server for at least 10 seconds, and if a request authenticated using them is made before they become invalid then they SHOULD be stored persistently and their validity SHOULD be extended indefinitely (barring de-authorisation). If a request with a correct client-id is made with an incorrect LSGS before either the credentials are automatically invalidated or a correct

request is made then the box MUST immediately remove the LSGS for this client-id from those which are currently valid, and notify the user that an invalid authentication attempt has occurred.

In both cases, the box MUST NOT automatically display a new pairing code without pro-active interaction by the user with the built-in user interface. such as using button presses to confirm the notification or re-command the server to display a new pairing code.

This resource never undergoes notifiable changes.

An example XML document that fulfils the requirements for a representation of the "uc/security" resource is as follows:

```
<response resource="uc/security">
  <security
    key="ed6898d098c9c90b6eb445bd7ccfdbb15cb6d1fe040cc8674d5edec7e453e10c15720bb7
65dafdbd39b7af608782d159c93a2e16e46c1d0f9e6e241c7c9643cbaca12"/>
</response>
```

4.5 The uc/events Resource (Recommended)

4.5.1 Introduction

This section is purely informative.

The "uc/events" resource implements an asynchronous "HTTP Long Polling" mechanism for notifying interested clients of changes to the box's internal state in near real-time. Such schemes are also known as "Comet" techniques [16]. This resource is used to notify the client of changes to the state of the "notifiable" UC resources. "Notifiable" resources are those implemented in such a way that they are capable of undergoing "notifiable" changes. Every time a server notifies one or more clients that notifiable changes have occurred to one or more resources, we say that a " client notification" has occurred.

As a non-normative hint to server implementors, one efficient way to implement the "uc/events" resource is to keep a table in memory that records a `notification-id` for each implemented notifiable resource. When changes to that resource take place, the `notification-id` associated with that resource is updated to the "current" `notification-id`, and any waiting client requests are updated as described above. When new client requests are processed by the server, the `notification-id` in the `since` parameter of the request can be compared to each item in the table and a list built of all notifiable changes with a stored `notification-id` which are not "earlier than" or "equal to" the given one, ie. all changes that have taken place since the given `notification-id` was "current".

4.5.2 Notification IDs

In order to keep track of notifiable changes and client requests this resource makes use of unique identifiers called `notification-id`s. A `notification-id` is a string representation of an integer. The server MAY use any string representation of `notification-id` integers, so long as it meets the requirements that there should be a 1:1 mapping between string and integer representations, and:

```
notification-id        = id-element
```

The server SHOULD represent `notification-id`s internally as 64-bit unsigned integers.

The server MUST create and maintain internally a variable that stores a single "current" `notification-id`. It is RECOMMENDED that the server stores the current `notification-id` persistently, ie that the value be retained when the box is power cycled. If this is the case, the `notification-id` MAY be initialised to any valid value when the box is turned on for the very first time, and then increased in value as client notifications occur, as described below. The consequence of not storing the current `notification-id` persistently is that there is a chance that clients may be sent inaccurate event notifications in the server's first response to each client after a power-cycle.

To minimise the probability of this occurring, the server implementer SHOULD follow the guidance in section 4.5.4 if they cannot or do not wish to store the current `notification-id` persistently.

4.5.3 Implementation

Implementing this resource is REQUIRED if any other implemented resources are capable of undergoing notifiable changes, and SHOULD NOT be implemented otherwise. The rest of this section is written assuming that this resource is being implemented. Representations of this resource MUST comply with the schema for a response element containing an "events" element defined in Appendix D. This section defines behaviour for the "uc/events" resource when the box is in "standby" mode that overrides the general behaviour mandated in section 4.

Upon receiving a GET request to this resource which has a query parameter named `since` whose value is a `notification-id` which is equal to or lower in value than the current `notification-id` at the point in time when the server processes the request, the server MUST construct a list containing the relative URIs of every resource that has undergone a notifiable change since the `notification-id` in the query parameter started being the server's current `notification-id`.

If the list is non-empty then the server MUST then add one to the current `notification-id`. Immediately afterwards, the server MUST respond to the client with a 200 response code and a representation that meets the following requirements:

- The response element MUST contain one `events` element.
- The `events` element MUST have its `notification-id` attribute set to the new value of the current `notification-id`.
- The `events` element MUST contain one `resource` element for each URI in the list, and the value of the `rref` attribute of each element MUST be the corresponding URI.

If the list is empty then the server SHOULD NOT respond to the request immediately, but SHOULD keep the connection open. We refer to requests whose connections are left open in this manner as "waiting" requests.

Upon a notifiable change occurring to any notifiable resource implemented on the server, the server MUST immediately add one to the current `notification-id` and assemble lists of resources for all waiting GET requests as if those requests had just arrived. Immediately afterwards, the server MUST respond to all the waiting requests with a 200 response code and a representation generated exactly as described above, in which a string representation of the newly-incremented current `notification-id` is used as the value of the `notification-id` attribute of the `events` element. Any further notifiable events which occur during this processing MUST be considered by the server to have occurred immediately after the incrementation of the current `notification-id`.

Upon receiving a GET request to this resource which does not have a query parameter named `since`, or which has a query parameter named `since` which is not identifiable by the server as a `notification-id`, or which is greater than the current `notification-id`, the server MUST respond by returning a response with a 200 response code containing a representation such that:

- The response element MUST contain one `events` element.
- The `events` element MUST have its `notification-id` attribute set to the current `notification-id`.
- The `events` element MUST NOT contain any `resource` elements.

The value of the current `notification-id` SHOULD NOT be altered as a result of processing a request of this kind.

The server MAY discard its record of notifiable changes (whatever form this takes) when it is powered off or enters standby mode. When the box is in standby mode and the server implementation is still functioning the server SHOULD respond to GET requests to this resource as described above except that the server SHOULD NOT include any resources other than "uc" and "uc/power" in the list of resources which have undergone notifiable changes.

The "uc/events" resource itself MUST NOT undergo notifiable changes.

An example XML document that fulfils the requirements for a representation of the "uc/events" resource is as follows:

```
<response resource="uc/events">
  <events notification-id="3004">
    <resource rref="uc/power"/>
    <resource rref="uc/outputs"/>
  </events>
</response>
```

4.5.4 Implementation on Servers with No Persistent Storage

If a server cannot reserve persistent storage for the current `notification-id`, or the server implementer does not wish to store the `notification-id` persistently for whatever reason, then the server SHOULD initialise it each time the box is switched on (or whenever the value of the current `notification-id` is otherwise forgotten by the server) to a value that is extremely unlikely to be lower than the last `notification-id` conveyed to clients. One way to do this is using a timestamp, as follows:

Let t be the number of seconds that have elapsed since a date and time known to be earlier than the box's date of manufacture.

Let the initial value of the "current" `notification-id` i be the string representation of the integer generated by the formula $i = t \times n$, where n is the maximum number of notifications expected between box power cycles. It is RECOMMENDED that $n \geq 1,000,000,000$. Even if an average of 100 client notifications occur every second, which is highly unlikely, this value of n would allow the box to stay powered on for more than seven years before there was any possibility that the server's inability to store the "current" `notification-id` persistently could lead to undesirable client behaviour.

4.6 The uc/power Resource (Recommended)

This resource SHOULD be implemented by all servers. Representations of this resource MUST comply with the schema for a response element containing a "power" element defined in Appendix D. It is NOT REQUIRED to implement any other resources in order to implement this one.

This resource is used to represent the power state of the box. This section defines behaviour for the "uc/power" resource when the box is in "standby" mode that overrides the general behaviour mandated in section 4.

When the box's power state is "on", the server MUST respond to a GET request made to this resource by returning a representation such that:

- The representation MUST contain a single `power` element.

- The `state` attribute of the `power` element MUST have the value "on".
- If the box is in the process of transitioning to the "standby" or "off" power states, the `transitioning-to` attribute of the `power` element MAY have the value "standby" or "off" respectively.

When the box's power state is "standby", and if the server is capable of responding to client requests when the box is in that state, the server MUST respond to a GET request made to this resource by returning a representation such that:

- The representation MUST contain a single `power` element.
- The `state` attribute of the `power` element MUST have the value "standby".
- If the box is in the process of transitioning to the "on" or "off" power states, the `transitioning-to` attribute of the `power` element MAY have the value "on" or "off" respectively.

If the server receives a PUT request whose body contains a representation that meets one of the two sets of requirements defined above whilst the box is "on" or in "standby" if the server is still capable of responding, then the behaviour of the box depends on the `state` attribute of the `power` element in the request:

- If it is equal to the current power state of the box, the server MUST return a 204 response with no body.
- If it is equal to "on", the server SHOULD begin switching the box's power state to "on". If the box does so then the server MUST return a 204 response with no body, and if the box does not then the server MUST return a 500 error.
- If it is equal to "standby" the server SHOULD begin switching the box's power state to "standby". If the box does so then the server MUST return a 204 response with no body, and if the box does not then the server MUST return a 500 error.

The server MUST ignore the value of any `transitioning-to` value of the `power` element in the client request.

This server MUST respond to any other PUT request by returning a 400 error.

If the server does not implement the "uc/events" resource then this resource MUST be implemented in such a way as to never undergo notifiable changes. If the server also implements the "uc/events" resource then this resource MUST undergo a notifiable change whenever the box transitions between power states. If a power state change will render the server unable to respond to further requests, then the notifiable change SHOULD occur before this happens.

An example XML document that fulfils the requirements for a representation of the "uc/power" resource is as follows:

```
<response resource="uc/power">
  <power state="on" transitioning-to="standby"/>
</response>
```

4.7 The uc/time Resource

This resource SHOULD be implemented by all servers. Representations of this resource MUST comply with the schema for a response element containing a "time" element defined in Appendix D. It is NOT REQUIRED to implement any other resource in order to implement this one.

The purpose of this resource is to allow accurate time synchronisation between clients and servers.

The server MUST respond to a GET request to this resource by returning a representation such that:

- The representation MUST contain a single `time` element.
- The `time` element MUST have a `rcvdtime` attribute which MUST be set to as close to the time the server received the GET request as the server implementation allows, in the Internet timestamp format described in section 1.3.
- The `time` element MUST have a `replytime` attribute which MUST be set to as close to the time the response is sent back as the server implementation allows, in the Internet timestamp format described in section 1.3.

If the box is in standby mode and the server is still responding to requests then the server MUST respond to requests to this resource normally.

This resource MUST NOT undergo a notifiable change simply because the time has advanced as it normally does. If the "uc/events" resource is not implemented then this resource MUST NOT undergo notifiable changes. If the "uc/events" resource is implemented then this resource SHOULD undergo a notifiable change if the time or date is changed in any way other than as a result of the normal passage of time (for example due to user action or a transition between winter and summer time).

An example XML document that fulfils the requirements for a representation of the "uc/time" resource is as follows:

```
<response resource="uc/time">
  <time rcvdtime="2001-01-01T00:00:00.000000Z"
        replytime="2001-01-01T00:00:01.000000Z"/>
</response>
```

4.8 The uc/outputs Resource

Implementing this resource is OPTIONAL, but is REQUIRED if any of the resources which lie below it in the path hierarchy are implemented. The rest of this section assumes that this resource is being implemented. Representations of this resource MUST comply with the schema for a response element containing an "outputs" element defined in Appendix D. If this resource is implemented then the "uc/events" resource MUST be implemented.

4.8.1 Introduction

This section is purely informative.

This resource provides an enumeration of the available outputs of the box to which media can be directed. Depending on the nature of the box, these may include (but are not limited to) television screens, external loudspeakers and HiFis, teletype printers, and "picture-in-picture" [17] displays on other outputs. An output is any independent device or component under the control of the box, either internal or external, with the ability to display media to the user. Multiple physical connectors on a box are considered to be the same output if they cannot be controlled independently: for example, the HDMI, SCART and audio line-out sockets on a DVD player would commonly be modelled as a single output for this reason. Internal recording capabilities of a box should not be considered as outputs. We define an output to be "accessible" if it is being made available to Universal Control clients for information or control purposes.

Servers must identify one of their outputs as the "main" output. This output should be the one that users would expect media to be rendered on by default: the television connected to a simple STB, for example, or the internal loudspeakers of an Internet radio. The box may re-designate which output is to be considered "main" at any time, if doing so would match user expectations. The main output should not be confused with an "active" output. Multiple outputs may be active simultaneously, and from the perspective of the API, any output connected to a source is considered to be active.

Some outputs may be identified as "children" of other outputs. For example, if a video output supports a "picture-in-picture" feature, the latter may be modelled as a child output of the former. This relationship is purely indicative, and does not affect any normative requirement imposed by this API: in particular, the `output-id`s (see section 4.8.2 of outputs with such a relationship must still be unique.

4.8.2 Output IDs

The box MUST maintain a list of outputs connected to the box, one of which MUST be considered the "main output" for the box. Each output MUST be assigned an `output-id` that is unique on a given server instance. An `output-id` takes the form:

```
output-id        = id-element
```

The box MUST NOT identify two outputs with the same `output-id` at the same time. The box MUST NOT assign two `output-id`s to the same output. The box MUST NOT change the `output-id` of an output while an output is being made available to clients for information or control purposes. The box MUST NOT have more than one main output. A box MAY change which of its outputs is the main output at any time. The server MUST NOT assign the string "main" as an `output-id`, since this is used as an alias for the main output in the UC resource hierarchy, as described in section 4.9.

4.8.3 Implementation

The server MUST respond to a GET request to this resource by returning a representation such that:

- The response MUST contain a single `outputs` element.
- The `outputs` element SHOULD contain one `output` element for each output currently accessible on the box which is not considered a sub-output of another output.
- The first `output` element in the representation MUST be associated to the main output of the box, and MUST have a `main` attribute set to "true".
- Each `output` element MUST have one `output` element contained within it for each output which is considered a sub-output of that output.
- Each `output` element except for the first one MUST NOT have a `main` attribute.
- Each `output` element MUST have its `oid` attribute set to the `output-id` of the output.
- Each `output` element MUST have its `name` attribute set to a human readable name which can be used to identify the output.

Any change to the box's state that would alter the response that would be returned by this resource to a GET request SHOULD trigger a notifiable change on this resource. If the box is implemented in such a way that these data will never change then this resource will never undergo a notifiable change, but its child resources likely will.

An example XML document that fulfils the requirements for a representation of the "uc/outputs" resource is as follows:

```
<response resource="uc/outputs">
  <outputs>
    <output name="Main Screen" oid="0" main="true">
      <output name="Picture-in-Picture" oid="pip"/>
    </output>
    <output name="Other Screen" oid="1"/>
  </outputs>
</response>
```

4.9 The uc/outputs/main Resource

This resource MUST be implemented if the box has implemented the "uc/outputs" resource, and MUST NOT be implemented otherwise.

This resource is simply an alternative URI for accessing the "uc/outputs/{id}" resource for the output which has been designated as the main output. This resource should respond precisely like that resource, and all representations returned by this resource MUST give the URI of the main output resource in the "uc/outputs/{id}" form.

This resource MUST NOT undergo notifiable changes. Changes to the output it represents will be notified as having occurred to the equivalent "uc/outputs/{id}" resource.

4.10 The uc/outputs/{id} Resources

4.10.1 Introduction

This section is purely informative.

The resources provide information regarding the content that is currently being presented on the box's output(s) and how it is being presented. These are also the resources to which POST requests are made to change the content currently being presented. The server indicates the content that is currently being presented using a `source-id` and a `content-id`. It is possible for an output to be presenting no content. it is only possible for a particular output to present a single piece of AV content at a time, and it is only possible for a single output to present a single piece of interactive content at a time. In some cases it may be possible for a single output to present one piece of interactive and one piece of AV content simultaneously, but this capability is not required and even on boxes which have it, it may only be possible with certain combinations of content.

In particular the behaviour when the presentation of a piece of content is requested on an output which is already presenting a piece of content of the other type is left entirely up to the individual implementation – and may vary depending upon the specific piece of content in question. For example a request to activate an MHEG application which can run as an overlay over video on an output currently presenting a television programme may cause the application to be overlaid on the video, whilst a request to present a locally stored video-file on an output currently presenting a DVD menu may well cause the new content to replace the old.

If the output is currently presenting AV content then it will have one or more media components associated with it. For example, a piece of content that represents the current programme on a TV channel might have a video component, two audio components in different languages and a subtitle component. As described in section 2.4, the server or box will consider some of these components to be "default", and and will not provide any further information that it is presenting them. If the box is presenting non-default components as part of the piece of content then it will indicate this by means of a component "override". Components that are explicitly listed in a representation of a "uc/outputs/{id}" resource "override" the default components of the AV content from which the output is presenting media. Default components can also be overridden with "nothing" to (for example) indicate that a default subtitle track has been turned off. The mechanism for indicating this is to include a media component of the appropriate type with a `media-component-id` of the empty string.

If the output is currently presenting interactive content then it will list the remote-control profiles which that content responds to.

If an output is presenting content from a source with the "follow-on" property (ie each piece of content from the source is automatically followed by another piece when it finishes) then whether or not component overrides persist across the content boundary is implementation-specific. It is recommended that the box and the server do whatever is most likely to meet the expectations of the user. For example, if two adjacent pieces of content offer a choice of audio tracks in different languages, and the same two languages are available for both pieces, it is likely that the user will expect the language presented on the output(s) of the box not to change across the content boundary.

In this and subsequent sections, the concept of a "decimal" number is used. A decimal number is one that meets the requirements for decimal numbers set out in the XML Schema 1.0 specification [18], which defines them to be fixed-precision representations of real numbers. "Minimally-conforming" XML Schema processors are required to support at least 18 decimal digits of precision (ie. they must be able to handle numbers with up to 18 decimal digits, regardless of where the decimal point falls within a given number).

POST requests to this resource can be made to alter the content which is currently being presented. There are several different ways of making such a request, including a method with only query variables and a method using request bodies which allows extra precision in the control it provides.

4.10.2 Implementation

One resource of this form MUST be implemented for each output if the box has implemented the "uc/outputs" resource, and MUST NOT be implemented otherwise. The rest of this section assumes that these resources are being implemented. Representations of this resource MUST comply with the schema for a response element containing an "output" element defined in Appendix D. If these resources are implemented then the "uc/events" resource MUST also be implemented, and the "uc/sources" resource SHOULD also be implemented.

Each resource of this type has a different URI in which the placeholder "{id}" is replaced by the `output-id` of the output (see section 4.8).

If the server receives a GET request to a resource of this form with an "{id}" not known to the box then it MUST return a 404 error.

The server MUST respond to any other GET request to a resource of this form by returning a representation such that:

- The representation contains a single `output` element.
- The `output` element MUST have a `name` attribute which MUST be set to a human-readable name which can identify the output to the user.
- The `output` element MUST contain a `settings` element.
- The `settings` element MAY have `volume` and `mute` attributes which MAY be set to indicate the volume and muted-status of the output. The `volume` attribute of the `settings` element is a decimal number that takes a value between `0` and `1`, with `0` representing no sound, and `1` representing full volume. The volume scale SHOULD be *perceptually* linear: ie a value of `0.5` SHOULD correspond to audio that sounds half as loud as for a value of `1`, and a value of `0.1` SHOULD result correspond to audio a quarter as loud as would correspond to a value of `0.4`, etc.
- The `settings` element MAY have an `aspect` attribute that indicates the aspect ratio to which the output is currently set. The values listed in section 2.4 are valid, along with the

special "source" value that indicates that the aspect ratio of the output is set to match that of the source-material.

- If the server also implements the "uc/sources" resource and the output is currently presenting AV content then the `output` element MUST contain a `programme` element. If the server does not implement this resource or if the output is not presenting AV content then the `output` element SHOULD NOT contain a `programme` element.
- If present the `programme` element MUST have `sid` and `cid` attributes. The `sid` attribute MUST be set to the `source-id` of the source of the AV content currently being presented on the output and the `cid` attribute SHOULD be set to the `content-id` of that content (see section 2), if not set to the `content-id` then the `cid` attribute MUST be set to the empty string.
- If a `programme` element in present and the default media components associated with the content being presented are being overridden with alternative components of the same type then the `programme` element MUST contain one `component-override` element for each overridden component. If a media component not present in the set of default media components is being added to the media presented by the output to the user, this is to be considered as overriding the absence of that component.
- If present each `component-override` element MUST have `type` and `mcid` attributes which MUST be set respectively to a valid component type and to the `media-component-id` for a valid component of the non-interactive media-component currently being presented by the output. If a default media component is being disabled in the presentation of the media to the user (eg turning off default subtitles) then a `component-override` element MUST be included to represent this, with its `mcid` attribute set to the empty string.
- If the server also implements the "uc/sources" resource and the output is currently presenting interactive content then the `output` element MUST contain an `app` element. If the server does not implement this resource or if the output is not presenting any interactive content then the `output` element SHOULD NOT contain an `app` element.
- If present the `app` element MUST have `sid` and `cid` attributes. The `sid` attribute MUST be set to the `source-id` of the source of the interactive content currently being presented on the output and the `cid` attribute SHOULD be set to the `content-id` of that interactive content (see section 2), if not set to the `content-id` then the `cid` attribute MUST be set to the empty string.
- If present the `app` element MAY contain any number of `controls` elements.
- Each `controls` element MUST have a `profile` attribute set to the `profile-id` of a remote-control profile to which the currently presented interactive content will respond (see section 7).
- If the output is currently presenting AV content then the `output` element MAY contain a `playback` element, otherwise it MUST NOT.
- If present the `playback` element MUST have a `speed` attribute, which, if present, MUST be set to a decimal value representing the speed of playback, with "0.0" representing a still frame, "1.0" representing normal play speed forwards, "2.0" representing double normal play speed forwards, "-1.0" representing normal play speed backwards, etc ...

Upon receiving a POST request to this resource with a message body such that:

- The representation contains a single `programme` element.
- The `programme` element has a `sid` attribute with a recognised source-id for its value.

- The `programme` element has an `cid` element which is a recognised content-id of a piece of AV content within the specified source.
- The `programme` element contains no elements other than `media-component` elements which conform to the restrictions for `media-component` elements described above and identify media-components found in the specified content.

then the box MUST immediately attempt to change its internal state such that the supplied `programme` element is a valid representation which could be returned as part of a GET request to this resource. If this action is attempted but fails immediately then the server MUST return a 500 error, otherwise it MUST return a 204 response with no response body.

Upon receiving a POST request as described above except that the `cid` attribute is blank then the server MUST respond exactly as if the request body had specified the current `default-content-id` for the specified source (automatically failing with a 500 error if the default piece of content for that source is interactive).

Upon receiving a POST request with no query parameters to this resource with a message body such that:

- The representation contains a single `app` element.
- The `app` element has a `sid` attribute with a recognised source-id for its value.
- The `app` element has an `cid` element which is a recognised content-id of a piece of interactive content within the specified source.
- The `app` element contains no elements.

then the box MUST immediately attempt to change its internal state such that the supplied `app` element is a valid representation which could be returned as part of a GET request to this resource (except possibly for missing some `controls` elements. If this action is attempted but fails immediately then the server MUST return a 500 error, otherwise it MUST return a 204 response with no response body.

Upon receiving a POST request as described above except that the `cid` attribute is blank then the server MUST respond exactly as if the request body had specified the current `default-content-id` for the specified source (automatically failing with a 500 error if the default piece of content for that source is non-interactive).

Upon receiving a POST request with no body but with a single query parameter `sid` which has a source-id of a source known to the box as its value and no query parameter `cid` the server MUST immediately attempt to alter the box's state such that the default piece of content of the specified source is being presented on the output. If unable to immediately do so the server MUST return a 500 error, otherwise it MUST return a 204 response with no response body.

Upon receiving a POST request with no body but with a single query parameter `sid` which has a `source-id` of a source known to the box as its value and a single query parameter `cid` which has the `content-id` of a piece of content contained in that source as its value the server MUST immediately attempt to alter the box's state such that the specified item of the specified source is being presented on the output. If unable to immediately do so the server MUST return a 500 error, otherwise it MUST return a 204 response with no response body.

Upon receiving a POST request that meets any of the aforementioned requirements, but which additionally has a single query parameter `seek` which has a decimal number as its value, the server SHOULD attempt to shift the box's current playback position within the current AV content item (or, if a new AV content item is being requested as part of the POST request, the new content item) to be the value of the `seek` parameter interpreted as a decimal number of seconds since the start of the piece of content. If the value of the `seek` parameter is a decimal number that does not correspond to a valid point to seek to within the piece of content, the server SHOULD instead

attempt to seek to the closest valid point to the value requested. If the value of the `seek` parameter is not a decimal number, if the server does not have information regarding the start time of the piece of content in which seeking is desired, or if the server cannot fulfil the request for some other reason, the parameter MUST be ignored.

Upon receiving any other POST request to this resource the server MUST immediately return a 400 error.

Any change to the information used in forming the representation which would be returned by a GET request made to one of these resources MUST trigger a notifiable change in that particular resource. Whilst it is conceivable that a box might be implemented in such a way that a particular resource of this form will never undergo notifiable changes this is not likely.

An example XML document that fulfils the requirements for a representation of the "uc/outputs/{id}" resource is as follows, for the case where {id}=0:

```
<response resource="uc/outputs/0">
  <output name="Main Screen">
    <settings volume="0.5" mute="false" aspect="16:9"/>
    <programme sid="BBCOne" cid="2009-12-14T22%3a00%3a00Z">
      <component-override type="audio" mcid="audio2" lang="UK English"/>
    </programme>
    <app sid="BBCOne" cid="BBC_Bridge">
      <controls profile="ft.example:ft_remote"/>
    </app>
    <playback speed="1.0"/>
  </output>
</response>
```

4.11 The uc/outputs/{id}/settings Resources

4.11.1 Introduction

This section is purely informative.

This resource is the resource to which clients will send representations in order to change the way in which content is presented by the output in question. For example a PUT request to this resource is used to change the volume or aspect-ratio of a television.

4.11.2 Implementation

One resource of this form MUST be implemented for each output if the box has implemented the "uc/outputs" resource, and MUST NOT be implemented otherwise. The rest of this section assumes that these resources are being implemented. Representations of this resource MUST comply with the schema for a response element containing an "settings" element defined in Appendix D. If these resources are implemented then the "uc/events" resource MUST also be implemented.

Each resource of this type has a different URI in which the placeholder "{id}" is replaced by the `output-id` of the output (see section 4.8).

If the server receives a GET request to a resource of this form with an "{id}" not known to the box then it MUST return a 404 error.

The server MUST respond to any other GET request to a resource of this form by returning a representation containing a `settings` element identical to that which would be contained in the `output` element of a response returned by a GET request to the parent resource.

In this section, we define a "valid" PUT request to be one whose message body contains a representation containing a `settings` element which conforms to the requirements placed upon a `settings` element contained in a representation returned by a GET request to the parent resource.

Upon receiving a PUT request to one of these resources that is not valid the server MUST return a 400 error.

Whenever the server receives a valid PUT request, the box SHOULD change its internal state so that the representation provided by the client becomes an accurate representation of the output. Any data specified in the request to have the same value as it does already SHOULD be left unchanged. If after the body is processed the state of the output matches that given by the representation then the server MUST return a 204 response with no message body. Otherwise the server MUST return a 500 error. A box SHOULD NOT change its internal state in any way as a result of a request which returned an error.

This resource is never notifiable, but a PUT request to this resource which results in changes to the state of the output will result in a notifiable change in the parent resource.

An example XML document that fulfils the requirements for a representation of the "uc/outputs/{id}/settings" resource is as follows, for the case where {id}=0:

```
<response resource="uc/outputs/0/settings">
  <settings volume="0.5" mute="false" aspect="16:9"/>
</response>
```

4.12 The uc/outputs/{id}/playhead Resources

4.12.1 Timing Information

This resource is used to represent the current playback position within the media stream of the non-interactive content being presented on the output at a given point in time.

To correctly process requests to this resource the server MUST itself have access to either or both of the following:

- A decimal number representing how far behind or in advance of the current stream position the current playback position is in seconds. This is called *relative* positioning information.
- A decimal number representing how far from the start of the stream the current playback position is in seconds. This is called *absolute* positioning information.

When including the positioning information in a representation, the server SHOULD report it to the greatest degree of accuracy possible, and use a level of precision appropriate to that accuracy. (For example, if the server can determine the positioning information to an accuracy of 0.01s, the corresponding value in the XML sent to the client should have two digits after the decimal point.

In this section, we define a "valid playhead resource" to be a resource of this form that meets the following criteria:

1. It has a value for {id} that is a valid `output-id` as described in section 4.8, and corresponds to the value of {id} in a "uc/outputs/{id}" resource implemented by the server.
2. It has a value for {id} which identifies an output which is currently presenting AV content.
3. It has a value for {id} which identifies an output for which the server currently has access to relative and/or absolute positioning information.

4.12.2 Implementation

One resource of this form MAY be implemented for each output if the box has implemented the "uc/outputs" resource, and MUST NOT be implemented otherwise. The rest of this section assumes that these resources are being implemented. Representations of this resource MUST comply with the schema for a response element containing a "playhead" element defined in Appendix D. If this resource is implemented then the "uc/events" resource MUST also be implemented.

The server MUST respond to any GET or PUT request to a resource of this form for which the value of {id} is not recognised by the server with a 404 error.

The server MUST respond to any GET or PUT request to a resource of this form with a recognised value of {id} but is not a "valid playhead resource" as defined in the previous section, with a 400 error.

The server MUST respond to a GET request to a valid playhead resource as follows:

- The `response` element MUST contain a single `playhead` element.
- The `playhead` element MUST contain a `timestamp` attribute which MUST be set to the point in time when the `aposition` and/or `rposition` data was measured, and be formatted according to the Internet timestamp format described in section 1.3.
- The `playhead` element MUST contain a `length` attribute set to the length of the content in seconds if the server has access to that information, and MUST NOT contain such an attribute if it does not.
- The `playhead` element MUST contain either a single `aposition` element or a single `rposition` element, or one of each.
- If present, the `aposition` element MUST have its `position` attribute set to a decimal value, which is the absolute positioning information for the source.
- if present the `aposition` element MUST contain a `seek-start` attribute set to the absolute position in the source which is the earliest to which seeking can currently happen if the server has access to such information, and MUST NOT contain such an attribute otherwise.
- If present the `aposition` element MUST contain a `seek-end` attribute set to the absolute position in the source which is the latest to which seeking can currently happen if the server has access to such information, and MUST NOT contain such an attribute otherwise.
- If present, the `rposition` element MUST have its `position` attribute set to a decimal value, which is the relative positioning information for the source.
- if present the `rposition` element MUST contain a `seek-start` attribute set to the relative position in the source which is the earliest to which seeking can currently happen if the server has access to such information, and MUST NOT contain such an attribute otherwise.
- If present the `rposition` element MUST contain a `seek-end` attribute set to the relative position in the source which is the latest to which seeking can currently happen if the server has access to such information, and MUST NOT contain such an attribute otherwise.
- The `playhead` element MUST contain a `playback` element if the representation returned by a GET request to the parent resource would contain one, and MUST NOT contain a `playback` element otherwise.
- If present the `playback` element MUST be identical to that which would be contained in the representation returned by a GET request to the parent resource.

The server MAY respond to all PUT requests to valid playhead resources by returning a 405 error, if the server does not support client-modifiable playhead resources.

If the server does not respond to all PUT requests to valid playhead resources with a 405 error then the server MUST respond to a PUT request to a valid playhead resource that has a message body which does not comply with the restrictions specified above for a response by returning a 400 error.

If the server does not respond to all PUT requests to valid playhead resources with a 405 error then upon receiving a PUT request to to a valid playhead resource with a body which does meet the

above requirement the box SHOULD change its internal state such that the representation given is a valid representation of the position of playback in the presented stream for this output. If after the processing of this request the internal state of the box matches the representation in the client request then the server MUST return a 204 response with no message body, otherwise it MUST return a 500 error. The attributes `length`, `seek-start`, and `seek-end` are NOT REQUIRED in the body of such a PUT request, and the server SHOULD ignore their value if they are present. If both an `aposition` and an `rposition` element are present in the request body, the server SHOULD ignore the first-listed of the two in the request.

When processing a PUT request as described above which contains a `timestamp` attribute for its `playhead` element that is a valid date and time in the Internet timestamp format, the server MAY assume that the timestamp represents the point in time at which the user made the request, and use the difference between the timestamp in the request and the box's internal clock to adjust the values of any relative position changes to account for network latency.

All timestamps provided by this resource MUST be in the same timeframe as those provided by the "uc/time" resource, and all timestamps received in requests to this resource MUST be interpreted as being in that timeframe as well.

Each resource of this form SHOULD undergo a notifiable change whenever the currently presented AV content for the associated output changes, and also whenever the current play position is caused to "jump" from one point in the stream to another or a change occurs to the rate of playback (such as pausing). Each resource of this form MUST NOT undergo a notifiable change if the play position has merely changed due to movement of the playhead at the current rate. A PUT request which results in no changes to the internal state of the box SHOULD NOT cause the resource to undergo a notifiable change.

An example XML document that fulfils the requirements for a representation of the "uc/outputs/{id}/playhead" resource is as follows, for the case where {id}=0:

```
<response resource="uc/outputs/0/playhead">
  <playhead length="2700" timestamp="2001-01-01T00:00:01.000000">
    <aposition position="257.3" seek-start="0" seek-end="2700"/>
    <rposition position="-10.0" seek-start="-267.3" seek-end="2432.7"/>
    <playback speed="1.0"/>
  </playhead>
</response>
```

4.13 The uc/remote Resource

Implementing this resource is OPTIONAL. The rest of this section assumes that this resource is being implemented. Representations of this resource MUST comply with the schema for a response element containing a "remote" element defined in Appendix D. It is NOT REQUIRED to implement any other resource in order to implement this one.

The purpose of this resource is to allow the direct simulation of infra-red remote control keypresses on the box by the client. It is assumed that all boxes will either support global keypresses, or that all keypresses must be associated to an individual output.

To properly respond to requests to this resource the server MUST have access to a list of remote control profiles (see section 7) to which the box will respond.

The server MUST respond to any GET request to this resource by returning a representation such that:

- The representation MUST contain a single `remote` element.
- The `remote` element SHOULD contain one `controls` element for each profile the box supports.

- Each `controls` element MUST have a `profile` attribute, which MUST be set to the profile-id for the profile in question..

The server MUST respond to any POST request to this resource which does not have a query parameter named `button`, which has more than one such parameter, which has more than one query parameter named `output`, which has a query parameter named `button` the value of which is not equal to any keycode in a profile supported by the box, or which has a query parameter named `output` the value of which is not equal to any known output on the box by returning a 400 error.

Upon receiving a POST request to this resource which does have a single query parameter named `button` which does have a value equal to a fully qualified keycode from one of the profiles supported by the box the server SHOULD return a 204 response with no message body. If the box supports keypresses globally but not for individual outputs then the box SHOULD then behave as if it had received the keypress that keycode represents through its normal input mechanisms, such as a button press on an infra-red remote control, or a keypress on an attached USB keyboard. If the box architecture requires keypresses to be directed to individual outputs and the request had a single `output` query parameter with a value equal to a known output on the box then the box SHOULD instead behave exactly as if the keypress represented by the keycode had been received by that output. If the box requires per-output routing of key-presses but no `output` query parameter is provided then the box SHOULD behave as if the main output had been specified.

If the "uc/events" resource is also implemented then this resource MUST undergo a notifiable change if the profiles supported by the box change. If the "uc/events" resource is not implemented or if the profiles are fixed then this resource will never undergo notifiable changes.

An example XML document that fulfils the requirements for a representation of the "uc/remote" resource is as follows:

```
<response resource="uc/remote">
  <remote>
    <controls profile="ft.example:ft_remote" />
    <controls profile=":uk_keyboard"/>
  </remote>
</response>
```

4.14 The uc/feedback Resource

Implementing this resource is OPTIONAL. The rest of this section assumes that this resource is being implemented. Representations of this resource MUST comply with the schema for a response element containing a "feedback" element defined in Appendix D. If this resource is implemented then the server MUST also implement the "uc/events" resource.

The purpose of this resource is to provide textual feedback from the box to the client device when the state of the box's built-in interface changes. This is expected to be of use for accessibility purposes, allowing users who cannot sense the built-in interface to know, for example, what menu item might currently be selected.

The server MUST respond to any GET request to this resource by returning a representation containing a single `feedback` element. The content of this feedback element MUST be a string (which MAY be empty) which is intended for presentation directly to a user and will give information about the current state of the user-interface. For example if the box is currently displaying a menu then this text could contain the textual title of the currently selected menu item. The `feedback` element MUST have a `time` attribute which MUST be set to the time at which the box notified the server that clients should present this text to the user.

Any change to the text which will be presented in this resource MUST trigger a notifiable change in this resource. A notifiable change MAY also be triggered when the text which will be

presented in this resource has not changed, if the box indicates to the server that clients should present the same textual information to the user again.

An example XML document that fulfils the requirements for a representation of the "uc/feedback" resource is as follows:

```
<response resource="uc/feedback">
  <feedback time="2001-01-01T00:00:00.000000Z">Press the red button for further information.</
</response>
```

4.15 The uc/source-lists Resource

4.15.1 Introduction

This section is purely informative.

The purpose of this resource is to provide "source lists" defined by the server implementer, which allow sources to be grouped together in various ways that help the user locate them. For example, sources that correspond to DVB services and VOD services on an STB could be included in two different lists, allowing clients to present them separately.

See section 2 for more information about sources and source-lists, and how they fit into the UC data model.

4.15.2 Implementation

This resource MUST be implemented if the "uc/sources" resource is implemented (see section 4.17), and MUST NOT be implemented otherwise. The remainder of this section assumes that it is being implemented. Representations of this resource MUST comply with the schema for a response element containing a "source-lists" element, defined in Appendix D. Other than the "uc/sources" resource no other resources are required for this resource to be implemented in general, although some implementations MAY be such that the "uc/events" resource is required.

If this resource is implemented, the box MUST maintain a list of "source lists". Each "source list" MUST be a list of `source-id`s.

The server MUST respond to a GET request to this resource by returning a representation such that:

- The representation MUST contain a single `source-lists` element.
- The `source-lists` element MUST contain one `list` element for each "source list" the box maintains.
- Each `list` element MUST have an `list-id` attribute which MUST be set to the `list-id` of the list,
- Each `list` element MUST have a `name` attribute which MUST be set to a short human readable name for the list.
- Each `list` element MAY have a `description` attribute containing a textual description of the list.
- Each `list` element MAY have a `logo-href` attribute, which, if present, MUST be set to the URL of an image which can be used to represent the list (which SHOULD have a scheme of either `http` or `https`).
- Each `list` element MAY contain any number of `link` elements identifying external links associated with the list. These links could be websites associated with the given list, for example. The `href` attribute of each `link` element MUST be a single correctly-formatted URL. The `description` attribute of each link MUST be a human-readable string describing

the link. A link associated with a source list in this manner MUST be intended for resolution by the client, and SHOULD only be included if resolving the link will result in a useful experience to the user. For example, if the link has an "http" or "https" scheme, then resolving the link in a web browser SHOULD result in the presentation of information to the user that is related to the relevant source list.

- If present each `link` element MUST have an `href` attribute, which MUST be set to a URI identifying a related piece of content.
- If present each `link` element MUST have a `description` attribute, which MUST be set to a human-readable string describing the related content.

If any of the data used to generate a response to a GET request to this resource changes then this resource MUST undergo a notifiable change. If this resource is implemented in such a way that the data will never change then it will never undergo a notifiable change and the "uc/events" resource is NOT REQUIRED. Simply changing which sources are on a list MUST NOT trigger a notifiable change in this resource.

An example XML document that fulfils the requirements for a representation of the "uc/source-lists" resource is as follows:

```
<response resource="uc/source-lists">
  <source-lists>
    <list list-id="uc_default" name="Default"/>
    <list list-id="box_dvb" name="TV"
          description="FooView(tm) TV channels"
          logo-href="http://stb.example/images/ondemand.jpg">
      <link href="http://dvbnetwork.example.com/ondemand/default.php?on=demand"
            description="FooView(tm)'s web site"/>
    </list>
    <list list-id="uc_storage" name="Recordings"
          description="Programmes recorded from FooView(tm)"
          logo-href="http://stb.example/images/storagedrive.jpg"/>
  </source-lists>
  </response>
```

4.16 The uc/source-lists/{id} Resource

Resources of this form MUST be implemented if the "uc/source-lists" resource is implemented, and MUST NOT be implemented otherwise. The remainder of this section assumes that these resources are being implemented. Representations of these resources MUST comply with the schema for a response element containing a "sources" element defined in Appendix D. If this resource is implemented then the "uc/sources" resource MUST be implemented. Depending upon the implementation the "uc/events" resource may also be REQUIRED.

The server MUST respond to any GET request made to a resource of this form with a value for "{id}" which is not equal to any `list-id` known by the box by returning a 404 error.

The server MUST respond to a GET request to this resource with a value for "{id}" which is equal to a `list-id` known by the box by returning a representation such that:

- The representation MUST contain a single `sources` element.
- The `sources` element which MUST contain one `source` element for each source on the requested list.
- Each `source` element MUST have a single `name` attribute set to a human readable description of the source.

- Each `source` element MUST have a single `sid` attribute set to the `source-id` for this source.
- Each `source` element SHOULD have an `sref` attribute set to a URI which identifies the origin of the source's content, as detailed in section 2.1.2, if such information is available for the source in question.
- Each `source` element MAY have a `live` attribute which MUST be `true` if the source has a concept of a current playback position that exists whether or not media from the source is being presented to the user and MUST be false otherwise.
- Each `source` element MAY have a `linear` attribute which MUST be `true` if all the pieces of AV content associated with a source have both start and end times, and the start and end times of different pieces do not overlap and MUST be false otherwise.
- Each `source` element MAY have a `follow-on` attribute which MUST be `true` if at the end of the presentation of one piece of content from the source to the user, presentation of another piece of content begins immediately, without interrupting the flow of media and MUST be false otherwise.
- Each `source` element MAY have a `audio-only` attribute which MUST be `true` if the source will never contain video and MUST be false otherwise.
- Each `source` element MAY have an `owner` attribute which gives a short human-readable identifier of the owner of this source. If this attribute is used, then sources from the same owner SHOULD have identical values of this attribute.
- Each `source` element MAY have an `lcn` attribute which contains an integer which corresponds to a logical channel number for this source.
- Each `source` element MAY have its `logo-href` attribute set to the URL of an image which can be used to represent the source (which SHOULD have a scheme of either `http` or `https`).
- Each `source` element MAY have its `owner-logo-href` attribute set to a URL of an image (with the same restrictions) which identifies the owner of the source.
- Each `source` element MAY have a `default-content-id` attribute. If present, the value of this attribute MUST be set to the `content-id` of the content that would be automatically selected if this source was selected to provide media to an output with no `content-id` explicitly specified.
- Each `source` element MAY contain any number of `link` elements identifying external links associated with the source. These links could be websites associated with the given source, for example. The `href` attribute of each `link` element MUST be a single correctly-formatted URL. The `description` attribute of each link MUST be a human-readable string describing the link. A link associated with a source in this manner MUST be intended for resolution by the client, and SHOULD only be included if resolving the link will result in a useful experience to the user. For example, if the link has an "http" or "https" scheme, then resolving the link in a web browser SHOULD result in the presentation of information to the user that is related to the relevant source.

If two `source` elements in different source lists share the same `source-id`, they MUST represent the same source, and the same values must be presented in each case for the elements and attributes listed above.

Items in the list SHOULD be returned in the same order each time a representation of the resource is returned to a client GET request. The order of items in the list SHOULD correspond to the order of the corresponding list in the box's built-in interface, if one exists.

A resource of this form MUST undergo a notifiable change any time sources are added to, removed from or replaced in its associated source list. Resources of this form MUST NOT undergo a notifiable change simply because a property of a source (eg its name `sref` or `default-content-id`) has changed. If the sources on a list will never change then this resource will never undergo a notifiable change and the "uc/events" resource is NOT REQUIRED.

An example XML document that fulfils the requirements for a representation of the "uc/source-lists/{id}" resource is as follows, for the case where {id}=`uc_default`. Note that in this example, a string representing the scheduled start time of a television programme has been used to identify the current default content within the "BBCOne" source. This is just one of many options for choosing `content-id`s, and is unlikely to be valid for all kinds of source.

```
<response resource="uc/source-lists/uc_default">
  <sources>
    <source name="BBC One"
        sid="BBCOne"
        default-content-id="2009-12-14T22%3a00%3a00Z"
        live="true"
        linear="true"
        follow-on="true"
        sref="dvb://233a.1048.1048"
        owner="BBC"
        owner-logo-href="http://example.com/bbc.png"
        logo-href="http://example.example/BBCOne.png"
        lcn="1">
      <link href="http://www.bbc.co.uk/bbcone/" description="BBC One Homepage"/>
    </source>
    <source name="Another TV channel"
        sid="%21%2a%5e%40%20TV%3b"
        live="true"
        linear="true"
        follow-on="true"
        sref="dvb://1234.5678.90ab"
        owner="example"
        owner-logo-href="http://example2.com/mylogo.png"
        logo-href="http://example2.com/another_channel_logo.png"
        lcn="2"/>
    <source name="On Demand Service"
        sid="OnDemandService"
        live="false"
        linear="false"
        follow-on="false"
        sref="http://example2.com/ondemandservice.php"
        owner="example"
        owner-logo-href="http://example2.com/mylogo.png"
        logo-href="http://example2.com/another_channel_logo.png"/>
  </sources>
</response>
```

4.17 The uc/sources Resource

Implementing this resource is OPTIONAL. The remainder of this section assumes that it is being implemented. This resource has no representation format. If this resource is implemented then the "uc/source-lists" resource MUST also be implemented.

Sub-resources of this resource are used to convey information about particular sources of media to the client. The "uc/sources" resource itself does not represent any aspect of the box's state.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.18 The uc/sources/{id} Resources

Resources of this form MUST be implemented if the "uc/sources" resource is implemented, and MUST NOT be implemented otherwise. The remainder of this section assumes that these resources are being implemented. Representations of these resources MUST comply with the schema for a response element containing a "source" element defined in Appendix D. If these resources are implemented then the "uc/source-lists" resource MUST also be implemented. Depending upon the implementation details the "uc/events" resource may also be REQUIRED.

Every `source-id` that is valid for use within this API MUST have an associated source that is represented by a resource of this form.

The server MUST respond to any GET request made to a resource of this form with a value for "{id}" which is not equal to a `source-id` in at least one of the "source lists" represented by the "uc/source-lists" resource (see section 4.15) by returning a 404 error.

The server MUST otherwise respond to each GET request made to a resource of this form by returning a representation containing a single `source` element representing this source with attributes and child-elements exactly as described for representations the "uc/source-lists/{id}" resource (see section 4.16). The `source` element in the representation returned by this request MUST be identical to one which would be contained in the response to a GET request to any "uc/source-lists/{id}" resource representing a source list containing the source in question.

A resource of this form MUST undergo a notifiable change whenever a change occurs that would alter the representation returned by a GET request to it. It is possible to implement these resources in a manner which does not generate notifiable changes, in which case the "uc/events" resource is NOT REQUIRED.

An example XML document that fulfils the requirements for a representation of the "uc/sources/{id}" resource is as follows, for an example case where {id}=`BBCOne`:

```
<response resource="uc/sources/BBCOne">
  <source name="BBC One"
      sid="BBCOne"
      default-cid="2009-12-14T22%3a00%3a00Z"
      live="true"
      linear="true"
      follow-on="true"
      owner="BBC"
      owner-logo-href="http://example.com/bbc.png"
      logo-href="http://example.example/BBCOne.png"
      lcn="1"/>
</response>
```

4.19 The uc/search Resource

Implementing this resource is REQUIRED if the server implements one or more of its sub-resources, it MUST NOT be implemented otherwise. Implementing the "uc/time" resource is RECOMMENDED if this resource is implemented. The remainder of this section assumes that the "uc/search" resource is being implemented.

4.19.1 Introduction

This section is entirely informative.

The purpose of this resource is to provide a base for the sub-resources which are listed in the following sections: "uc/search/outputs", "uc/search/sources", "uc/search/source-lists", "uc/search/text", "uc/search/categories", "uc/search/global-content-id", "uc/search/global-series-id", and "uc/search/global-app-id". The "uc/search" resource itself is not a functional part of the UC API.

Each of the aforementioned sub-resources corresponds to a particular type of "top-level" search: for example, a request to the "uc/search/categories resource will return results that fall into the categories specified in the query parameters of the request. Together, the different sub-resources offer client-developers a range of different ways to search for content and its metadata, which all follow roughly the same format: a request to the URI "uc/search/{type}/{term}" will return results for the search term "term" of type "type", where "type" is one of "outputs", "sources", "source-lists", "text", "categories", "global-content-id", "global-series-id", or "global-app-id", and the term is correspondingly an `output-id`, `source-id`, etc.... In each case the search can be refined and controlled by specifying a series of query parameters with the request, which act as "filters" on the results which would otherwise be returned by the search. For example the request:

```
uc/search/sources/BBC_One;BBC_Two?results=10&start=2011-01-25T13:50:00Z
```

would return lists of content from each of the sources `BBC_One` and `BBC_Two`, filtered such that each list contains the first ten items of content known to the box whose start times are at or after 13:50:00 UTC on Tuesday 25th January 2011.

The process by which a server responds to a request to any of these sub-resources is essentially the same:

1. Parse the `{term}` path-segment of the URI in a manner dependent upon the specific sub-resource in question. If this results in an invalid response then the server will return a 404 error, otherwise it will continue.

2. Parse the query parameters of the resource as described in Section 4.19.3. The specific query parameters that are valid for each sub-resource will depend upon the sub-resource in question.

3. Assemble one or more content lists. This step is dependent upon the sub-resource in question, but in all cases a mechanism is described for identifying a number of pieces of content and assembling them into a numerically indexed list, starting from index 0. Since such a list may be infinite in length the server should only identify the content in the list as it is required to do so for the process of assembling the actual return XML. All sub-resources except for `uc/search/sources/{sid}` will produce a single list in response to a single request, whilst requests to that sub-resource may produce multiple lists.

4. Filter the lists based on the values of the query parameters. This process is common for all resources and is described in Section 4.19.4. Since the lists will generally be of infinite length this filtering should be performed dynamically as required for the assembling of the return XML.

5. Return the response to the client, as described in Section 4.19.5.

4.19.2 Implementation of the uc/search Resource

All sub-resources of this resource which have a representation format MUST conform to the schema for a representation containing one or more "results" elements defined in Appendix D.

The server MUST respond to a GET request to the "uc/search" resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.19.3 Parsing the Query Parameters for a Sub-Resource

Upon receiving a GET request to a sub-resource of the form `uc/search/{type}/{term}`, after parsing the `{term}` path segment as described for the individual resource, the server MUST parse the query parameters received with this request as follows:

Query parameters are separated by the character `&` and are of the form `{key}={value}`. The table below lists each allowed query parameter, whether it may appear more than once, the default value for it (or `None` if there is none) and the description of what form its values must take. Not all query parameters are valid for all such sub-resources; each individual sub-resource lists which ones are valid for requests to it.

Depending on whether or not a query parameter may appear more than once in a request to a sub-resource, the values of each parameter are stored either as a single value or as a list, in a local variable with the same name as the query parameter. If a query parameter is not present in a request, the corresponding local variable is filled with the default value if one exists, and left unset otherwise. If a variable is interpreted as a percent-encoded string then the values are un-percent-encoded before being written to the local variables. If a parameter is interpreted as one or more `id-element`s, then the values of the parameter are to be interpreted as strings, and a parsing error MUST occur if they contain any characters forbidden in the definition of an `id-element`.

Key	List?	Default	Interpretation
results	No	1	An integer greater than or equal to 1.
offset	No	0	An integer.
sid	Yes	None	`id-element`s.
cid	Yes	None	`id-element`s.
series-id	Yes	None	`id-element`s.
gcid	Yes	None	Percent-encoded strings.
gsid	Yes	None	Percent-encoded strings.
gaid	Yes	None	Percent-encoded strings.
category	Yes	None	`id-element`s that MUST be known `category-id`s.
text	Yes	None	Percent-encoded strings.
field	Yes	None	Must be "title" or "synopsis".
interactive	No	"true"	A boolean type, either "true" or "false".
AV	No	"true"	A boolean type, either "true" or "false".
start	No	current time	A timestamp in internet standard timestamp format.
end	No	None	A timestamp in internet standard timestamp format.
days	No	None	An integer greater than or equal to 1.

If there is a parsing error, including the circumstance where a parameter allowed only once is included multiple times, or if the value of a parameter does not match the format required for it, then the server MUST immediately return a 400 error. If both the `days` and `end` parameters are included in the same request then the server MUST immediately return a 400 error, since these are mutually exclusive.

Furthermore if the request is not to a resource of the form `uc/search/outputs/{id}` then the value of `offset` must be greater than or equal to 0, if it is not then the server MUST return a 400 error.

Finally, if the `field` internal variable is not set then the server MUST set the `field` internal variable to be equal to the list containing the strings "title" and "synopsis".

The server MUST then proceed with assembling the content lists, as described in the following section.

4.19.4 Filtering the Content List on a Request to a Subresource

The lists assembled in response to a request to a sub-resource of the form `uc/search/{type}/{term}` should be filtered as follows. When elements are removed from the lists the remaining items are reindexed so that they move towards 0, ie. if the item at index 2 was removed then the item previously at index 3 would now be at index 2, and if the item at index -3 were removed then the item previously at -4 would now be at -3. Since the lists will in many cases be of infinite length this filtering SHOULD only be performed as required to assemble the XML returned from the request.

For each list, the list elements MUST be filtered as follows:

- If the `sid` local variable is set and not empty then the server MUST remove all content from the list with a source-id not equal to one of the values in the list.
- If the `cid` local variable is set and not empty then the server MUST remove all content from the list with a content-id not equal to one of the values in the list.
- If the `series-id` local variable is set and not empty then the server MUST remove all content from the list with a series-id not equal to one of the values in the list.
- If the `gcid` local variable is set and not empty then the server MUST remove all content from the list with a global-content-id not equal to one of the values in the list.
- If the `gsid` local variable is set and not empty then the server MUST remove all content from the list with a global-series-id not equal to one of the values in the list.
- If the `gaid` local variable is set and not empty then the server MUST remove all content from the list with a global-app-id not equal to one of the values in the list.
- If the `category` local variable is set and not empty then the server MUST remove all content from the list which does not belong to one of the categories identified by the ids in the list.
- If the `interactive` local variable is `False` then the server MUST remove all interactive content from the list.
- If the `AV` local variable is `False` then the server MUST remove all AV content from the list.
- The server MUST remove from the list any content with a last-time of presentability (or last-time of acquirability if there is no last time of presentability) earlier than the `start` internal variable.
- If the `end` internal variable is set then the server MUST remove any content with a start-time, first time of presentability, or first time of acquirability (if there is no first time of presentability) later than the `end` internal variable.
- If the `text` internal variable is set and the `fields` internal variable contains both "title" and "synopsis" then the server MUST remove any content which has a title or a synopsis and does not match a case-insensitive search in either its title or synopsis for all of the string values found in the `text` variable.
- If the `text` internal variable is set and the `fields` internal variable contains "title" but not "synopsis" then the server MUST remove any content which has a title and does not match a case-insensitive search in its title for all of the string values found in the `text` variable.

- If the `text` internal variable is set and the `fields` internal variable contains "synopsis" but not "title" then the server MUST remove any content which has a synopsis and does not match a case-insensitive search in its synopsis for all of the string values found in the `text` variable.

4.19.5 Returning a Response to a Query

After assembling and filtering the lists in response to a request to a sub-resource of the form `uc/search/{type}/{term}` the server MUST immediately start from the element in each list at index 0 and step forwards (or, if the `offset` variable is negative then backwards) through the list until either there are no more elements in the list, or it reaches the element with index equal to the value of `offset`. The selected piece of content is called the "first piece" for that list. If a list runs out of content before reaching the offset then the server MUST proceed as if that list were empty.

The server MUST then return a representation such that:

- The representation MUST contain one `results` element for each list.
- If a list is empty then the associated `results` element MUST contain no child elements.
- Otherwise each `results` element MUST contain one `content` element for each piece of content starting from the "first piece" and moving forwards through the list until either there are no more pieces of content in the list or the element at index `results` + `offset` - 1 is reached.
- If there is no item in a list with index `results` + `offset` then the appropriate `results` element MUST contain a `more` attribute with value `false`.
- If there is an item in a list with index `results` + `offset` then the appropriate `results` element MUST contain a `more` attribute with value `true`.
- Each `content` element MUST have an `sid` attribute containing a `source-id` for the content within the source.
- Each `content` element MUST have an `cid` attribute containing a `content-id` for the content within the source.
- Each `content` element MAY have a `cref` attribute containing a URI that identifies the origin of this content. See section 2.2.2 for details.
- The `logo-href` attribute MAY be used to specify a URL from which an image identifying the content can be retrieved (the scheme of thus URL MUST be `http` or `https`).
- The `interactive` attribute MUST be "true" if the content is interactive, and false otherwise.
- All other attributes of each `content` element MUST be included if the box has the data to fill them, and MUST NOT be included otherwise:
  - The `global-content-id` attribute carries a global content identifier for the content.
  - The `series-id` attribute carries a content identifier for a series which the content belongs to.
  - The `global-series-id` attribute carries a global series identifier for the content.
  - The `global-app-id` attribute carries the property of that name for a piece of interactive content that uses the API extension mechanism defined in section 4.46.

- The `title` attribute carries a human readable title for the content[4].
- The `start` attribute carries the date and time (in the Internet standard timestamp format described in section 1.3) at which a piece of AV content on a "live" source will begin.
- The `duration` attribute carries the length of a piece of AV content as a decimal number of seconds.
- The `presentable` attribute is "true" if the content is currently available for presentation on an output, and "false" otherwise.
- The `acquirable` attribute is "true" if the content is currently available to acquire (via the "uc/acquisitions" resource) and "false" otherwise.
- The `extension` attribute is "true" if the content is interactive and capable of making use of the extension mechanism described in (Section 4.44)
- The `presentable-from` attribute carries the date and time (in the Internet standard timestamp format) at which the content will first become (or first became) available for presentation.
- The `presentable-until` attribute carries the date and time (in the Internet standard timestamp format) at which the content will stop (or stopped) being available for presentation.
- The `acquirable-from` attribute carries the date and time (in the Internet standard timestamp format) at which the content will first become (or first became) available for acquisition.
- The `acquirable-until` attribute carries the date and time (in the Internet standard timestamp format) after which the content will stop (or stopped) being available for acquisition.
- The `presentation-count` attribute indicates the number of times a user has initiated the presentation of any part of the content on an output.
- The `last-presented` attribute indicates the most recent date and time (in the Internet standard timestamp format) at which the content stopped being presented by on output.
- The `last-position` attribute indicates the last position the playback of a piece of AV content reached (as a decimal number of seconds from the start of the content) before being stopped.
- The `associated-sid` attribute contains the `source-id` in which the associated content for this content is located.
- The `associated-id` attribute contains the `content-id` of the associated content for this content.

- If the box has a textual synopsis for the content then that MUST be contained in a `synopsis` element inside the `content` element.
- If the box has category information for the content and the server implements the "uc/categories" resource then a `category-id` for each category the content falls into MUST be specified as the `category-id` attribute of a `category` element inside the `content` element. The server SHOULD NOT include any category in this list that is a parent (or grandparent, etc) of another category being included in the list.

[4]Some metadata sources have separate concepts of "series name" and "episode name". The "title" attribute in a "content" element in a representation of this resource should contain all the information necessary to fully identify the content. It should not be just a series name, and it should not be just an episode name if a series name is also required to identify the content fully. For example, the episode "The Ninky Nonk Wants a Kiss" of the series "In The Night Garden..." requires both pieces of information to fully identify the content.

- If the media components that make up a piece of AV content are known, then the `content` element MUST contain a `media-component` element for each known component, each with an `mcid` attribute giving a `media-component-id` identifying that component and a `type` element identifying that component's type. If a media component forms part of the default experience presented to the user, the `default` attribute of that component's associated `media-component` element MUST be set to `true`.

- If the remote-control profiles which a piece of interactive content will respond to are know then the `content` element MUST contain a `controls` element for each known profile, each with a `profile` attribute set to the `profile-id` of the profile in question.

- Each `content` element MAY contain any number of `link` elements identifying external links associated with the content. These links could be websites associated with the given content, for example. The `href` attribute of each `link` element MUST be a single correctly-formatted URL. The `description` attribute of each link MUST be a human-readable string describing the link. A link associated with the content in this manner MUST be intended for resolution by the client, and SHOULD only be included if resolving the link will result in a useful experience to the user. For example, if the link has an "http" or "https" scheme, then resolving the link in a web browser SHOULD result in the presentation of information to the user that is related to the relevant content.

An example XML document which fulfils the requirements for a representation of a sub-resource of the form `uc/search/{type}/{term}` is as follows, for the case where `{type}` is "outputs" and `{term}` is "0".

```
<response resource="uc/search/outputs/0?results=2">
  <results more="true">
    <content cid="2009-12-14T22%3a00%3a00Z"
        sid="BBC_One"
        global-content-id="crid://bbc.co.uk/272950470"
        series-id="crid%3A%2F%2Fbbc.co.uk%2F__SERBBCNewsatTen"
        global-series-id="crid://bbc.co.uk/__SERBBCNewsatTen"
        title="BBC News at Ten"
        start="2009-12-14T22:00:00Z"
        duration="1500.0"
        interactive="false"
        acquirable="false"
        presentable="true"
        presentable-from="2009-12-14T22:00:00Z"
        presentable-until="2009-12-14T22:25:00Z"
        acquirable-from="2009-12-14T22:00:00Z"
        logo-href="http://example.com/BBCNews.png">
      <synopsis>The latest national and international
                news, with reports from BBC correspondents
                worldwide. [S]</synopsis>
      <category category-id="nati"/>
      <category category-id="weat"/>
      <media-component type="video" aspect="16:9"
                       mcid="video" name="Main Video"
                       vidformat="SD" default="true"/>
      <media-component type="audio" name="Main Audio"
                       mcid="audio1" lang="UK English" audformat="stereo"
                       default="true"/>
```

```
      <media-component type="audio" name="Audio Description"
                       mcid="audio2" lang="UK English" intent="admix"
                       name="Audio Description"  audformat="stereo"
                       default="false"/>
      <media-component type="subtitles" lang="UK English"
                       mcid="subtitles1" intent="hhsubs"
                       default="false"/>
      <link href="http://www.bbc.co.uk/programmes/b007mplc"
            description="BBC News website"/>
    </content>
    <content cid="2009-12-14T22%3a25%3a00Z"
        sid="BBC_One"
        global-content-id="crid://bbc.co.uk/272951840"
        series-id="crid%3A%2F%2Fbbc.co.uk%2F__SERBBCLondonNews"
        global-series-id="crid://bbc.co.uk/__SERBBCLondonNews"
        title="BBC London News"
        start="2009-12-14T22:25:00Z"
        duration="600.0"
        interactive="false"
        acquirable="true"
        presentable="false"
        presentable-from="2009-12-14T22:25:00Z"
        presentable-until="2009-12-14T22:35:00Z"
        acquirable-from="2009-12-14T22:25:00Z"
        logo-href="http://example.com/BBCLondonNews.png">
      <synopsis>The latest stories from the BBC London newsroom.
                [S]</synopsis>
      <category category-id="loca"/>
      <category category-id="weat"/>
      <media-component type="video" aspect="16:9"
                       mcid="video" name="Main Video"
                       vidformat="SD" default="true"/>
      <media-component type="audio" name="Main Audio"
                       mcid="audio1" lang="UK English"  audformat="stereo"
                       default="true"/>
      <media-component type="audio" name="Audio Description"
                       mcid="audio2" lang="UK English" intent="admix"
                       name="Audio Description" audformat="stereo"
                       default="false"/>
      <media-component type="subtitles" lang="UK English"
                       mcid="subtitles1" intent="hhsubs"
                       default="false"/>
      <link href="http://www.bbc.co.uk/programmes/b006mj67"
                       description="BBC London News website"/>
    </content>
  </results>
</response>
```

4.20 The uc/search/outputs Resource

Implementing this resource is REQUIRED if the server implements the "uc/outputs" resource and any other "uc/search" sub-resource. This resource MUST NOT be implemented if the server does

not implement the "uc/outputs" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/outputs" resource is being implemented.

The purpose of resources of the form "uc/search/outputs/{term}" is to allow clients to determine the content currently being presented on an output and, if applicable and if known, the pieces of content which preceded it and will follow it.

The server MUST respond to a GET request to the "uc/search/outputs" resource itself by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.21 The uc/search/outputs/{output-id} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/outputs/{output-id}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.21.1 Introduction

The purpose of these resources is to allow the client to make requests to determine what content is currently, will soon be, or recently has been presented on a specific output. For this reason this resource is the only one of the search sub-resources which allows the `offset` query parameter to take a negative value.

4.21.2 Parsing the search term

Upon receiving a GET request to a resource of this form with a value for the path-segment `output-id` which is not an output-id used in the "uc/outputs/{output-id}" resources or the string "main", the server MUST immediately return a 404 error.

Upon receiving a GET request to a resource of this form which is such an id the server MUST immediately set the local `term` variable to be equal to this id (or, if the path-segment is equal to the string "main". the server MUST immediately set the local variable to be equal to the `output-id` of the output that would be identified as the main output in a simultaneous request to the "uc/outputs" resource) and then proceed to parsing the query parameters as described in Sections 4.19.3 and 4.21.3.

4.21.3 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. The query parameters which are valid for requests to this resource are: `results`, `offset`, `interactive`, `AV`, `start`, `end`, and `days`. For requests to resources of this form the value of `offset` MAY be negative.

After parsing the query parameters the server MUST assemble the content list as described in the following section.

4.21.4 Assembling the Content List

This section describes how the server will assemble a list of content items which correspond to the terms of the search. This list may potentially be infinitely long, so it is necessary for server implementations to assemble this list only as it is required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST immediately be set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an additional number of days equal to (`days` − 1).

The output identified by the `output-id` stored in the `term` variable will be referred to as the "selected output" in this section.

If the selected output is not currently presenting any content then the server MUST immediately proceed to Section 4.19.4 with an empty list of content.

The next step depends upon what is being presented on the selected output:

- If no content is currently being presented on the output then the process continues as described in the following section, but with an empty list.
- If there is currently a piece of interactive content and no AV content being presented on the output then the process continues as described in the following section with a list containing only the currently-presenting piece of interactive content at list index 0.
- If there is currently a piece of interactive content and a piece of AV content presenting on the output then the interactive content is placed at list index 0 and the AV content at list index 1, and if further elements from the list are required then the rest of this section supplies the process for generating them.
- If there is currently no piece of interactive content being presented, but a piece of AV content is being presenting on the output then the AV content is placed at list index 0, and if further elements from the list are required then the rest of this section supplies the process for generating them.

To generate the piece of content with index n on the list, where $n > 0$, and the piece of content $n - 1$ is already known consider the source from which content $n - 1$ originates. If this source has the follow-on property then piece of content n is the next piece of content which will follow piece of content $n - 1$ when it finishes presenting if this is known. If the source does not have the follow on property or the next piece of content is not known to the box then the list has no piece of content at index n or any larger index.

To generate the piece of content with index -1 on the list consider the source for the piece of AV content at index 0 on the list, or if the piece of content at index 0 is interactive then instead consider the source for the piece of content at index 1. If this source has the follow-on property then the piece of content at index -1 is the piece of content which would have been presented before the piece of content at index 0 (or index 1 if the piece of content at index 0 is interactive). If this piece of content is not known or the source does not have the follow-on property then the list does not have any content at negative indices.

To generate the piece of content with index m on the list, where $m < -1$, and the piece of content $m + 1$ is already known consider the source from which content $m + 1$ originates. If this source has the follow-on property then piece of content m is the previous piece of content which would have been presented before piece of content $m + 1$ if this is known. If the source does not have the follow on property or the previous piece of content is not known to the box then the list has no piece of content at index m or any smaller index.

Upon generating this list the server MUST filter it as described in Section 4.19.4, and then respond to the request as described therein.

4.22 The uc/search/sources Resource

Implementing this resource is REQUIRED if the server implements the "uc/sources" resource and any other "uc/search" sub-resource. This resource MUST NOT be implemented if the server does not implement the "uc/sources" resource. In all other circumstances, implementing this resource

is OPTIONAL. The remainder of this section will assume that the "uc/search/sources" resource is being implemented.

The purpose of resources of the form "uc/search/sources/{term}" is to allow clients to determine the content items that are available from a given source, or list of sources.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.23 The uc/search/sources/{source-ids} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/sources/{source-ids}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.23.1 Introduction

Because resources of this form may be used to fetch information about content from multiple sources of content, they may assemble more than one content list, and hence may have more than once `results` element in their response.

4.23.2 Parsing the search term

Upon receiving a GET request to a resource of this form the server MUST split the `source-ids` component of the path into a series of values separated by the literal character `;` and assign these values to the local list variable `term`. Ie. if the request was made to the resource `uc/search/sources/BBC_One;BBC_Two` then the value of the local variable would be the list `['BBC_One','BBC_Two']`.

If any of the values in this list is not a valid source-id contained in some source-list then the server MUST immediately return a 404 error.

Otherwise the server MUST proceed to parsing the query parameters as described in Sections 4.19.3 and 4.23.3.

4.23.3 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. All query parameters described in that section are valid for these resources except for `sid`. For requests to resources of this form the value of `offset` must not be negative.

After parsing the query parameters the server MUST proceed to the next section.

4.23.4 Assembling the Content Lists

In this section the server will assemble lists of content items which correspond to the terms of the search. These lists may potentially be infinitely long, so it is necessary for server implementations to assemble the lists only as they are required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST be immediately set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an additional number of days equal to (`days` − 1).

The server MUST follow the mechanisms described in the rest of this section (and the next one) once for each source identified by a source-id in the local variable list `term`. In the rest of this section the source currently being worked with is called the "selected source".

If the selected source has the linear property then the list MUST be assembled which contains all content available from that source; Any pieces of interactive content from the source which are currently presentable MUST appear in the list starting at index 0, any AV content currently presentable MUST appear immediately following the interactive content (or at index 0 if there is no currently presentable interactive content), all other AV content MUST be arranged in order of start time with later content having a larger index. Other interactive content MAY be arranged in any desired way. The server MUST then proceed with the next section.

If the selected source does not have the linear property then the list MUST be assembled such that the content identified by the current default-content-id for the source appears at index 0 on the list if such a piece of content exists. Other content may appear on the list in any order. It is RECOMMENDED that content be ordered in the way which will best match user expectations. The server MUST then proceed with the next section.

Each list MUST be assembled in such a way that the same request will always result in the same ordering of the list.

Upon generating these lists the server MUST filter them as described in Section 4.19.4, and then respond to the request as described therein.

4.24 The uc/search/source-lists Resource

Implementing this resource is REQUIRED if the server implements the "uc/sources" resource and any other "uc/sources" sub-resource. This resource MUST NOT be implemented if the server does not implement the "uc/sources" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/source-lists" resource is being implemented.

The purpose of resources of the form "uc/search/source-lists/{term}" is to allow clients to determine the content items available in the sources that constitute the given source-list. As such, a request to this resource is always equivalent to making a request to a "uc/search/sources/{term}" resource (see section 4.22) whose {term} path-component is a list of all the sources in the given source-list.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.25 The uc/search/source-lists/{source-list-ids} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/source-lists/{source-list-ids}" resources are being implemented.

4.25.1 Parsing the search term

Upon receiving a GET request to a resource of this form the server MUST treat the `source-list-ids` component of the path as a series of values separated by the literal character ";", and assign these values to the local list variable `source-list-id`. For example, if the request was made to the resource `uc/search/sources/uc_default;uc_storage` then the value of the local variable would be the list `['uc_default','uc_storage']`.

If any of the values in this list is not a valid source-list-id then the server MUST immediately return a 404 error.

Next, the server MUST populate the internal list variable `term` with the concatenation of the `source-id`s contained in the `source-lists` identified by the `list-id`s in the `source-list-id` variable. The resulting list should be trimmed to remove any duplicate `source-id`s.

Next, the server MUST respond to the request exactly as described in the `uc/search/sources/{source-ids}` resource starting with Section 4.23.3 and proceeding from there as for that resource.

4.26 The uc/search/text Resource

Implementing this resource is REQUIRED if the server implements the "uc/sources" resource and any other "uc/search" sub-resource. This resource MUST NOT be implemented if the server does not implement the "uc/sources" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/text" resource is being implemented.

The purpose of resources of the form "uc/search/text/{term}" is to allow clients to determine the content known to the server that matches a specified text string in its title and/or synopsis. In resources of this form, multiple text strings may be included in the final path element, all percent-encoded, and all separated by a literal "`+`". The search treats the relationship between these strings as an "and" type relationship (hence the use of "`+`" rather than "`;`", which is used elsewhere to distinguish multiple independent searches).

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.27 The uc/search/text/{search-terms} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/text/{search-terms}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.27.1 Parsing the search term

Upon receiving a GET request to a resource of this form the server MUST split the `search-terms` component of the path into a series of values separated by the literal character `+`, unpercent-encode these values and assign them to the local list variable `term`. Ie. if the request was made to the resource `uc/search/text/foo%20bar+baz` then the value of the local variable would be the list `['foo bar','baz']`.

Then the server MUST proceed to parsing the Query Parameters as described in Sections 4.19.3 and 4.27.2.

4.27.2 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. All query parameters described in that section are valid for these resources except for `text`. For requests to resources of this form the value of `offset` must not be negative.

After parsing the query parameters the server MUST proceed to the next section.

4.27.3 Assembling the Content Lists

In this section the server will assemble a list of content items which corresponds to the terms of the search. This list may potentially be infinitely long, so it is necessary for server implementations to assemble the list only as it is required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST be immediately set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an

additional number of days equal to (`days` − 1).

If the local variable `field` contains both the strings "title" and "synopsis" then the server MUST proceed with a list containing every piece of content known to it which has a case insensitive substring match for all of the values in the `term` local variable in its synopsis or title.

If the local variable `field` contains the string "title" but not "synopsis" then the server MUST proceed with a list containing every piece of content known to it which has a case insensitive substring match for all of the values in the `term` local variable in its title.

If the local variable `field` contains the string "synopsis" but not "title" then the server MUST proceed with a list containing every piece of content known to it which has a case insensitive substring match for all of the values in the `term` local variable in its synopsis.

It is REQUIRED that if pieces of content in the list are currently available for presentation then one of the pieces which is currently available should be at list index 0 and all other pieces currently available for presentation SHOULD follow it before any pieces of content which are not currently available for presentation.

Otherwise the list MAY be ordered in any order which the server implementer wishes, but it is RECOMMENDED that all AV content from sources with the linear property should be arranged in order of start time.

It is REQUIRED that repeated requests to this resource with the same parameters will produce the same ordering.

Upon generating these lists the server MUST filter them as described in Section 4.19.4, and then respond to the request as described therein.

4.28 The uc/search/categories Resource

Implementing this resource is REQUIRED if the server implements the "uc/sources" resource, the "uc/categories" resource and any other "uc/search" sub-resource. This resource MUST NOT be implemented if the server does not implement both the "uc/sources" resource and the "uc/categories" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/categories" resource is being implemented.

The purpose of resources of the form "uc/search/categories/{term}" is to allow clients to determine the content known to the server that is associated with the category specified in the {term} path component.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.29 The uc/search/categories/{category-id} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and they MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/categories/{category-id}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.29.1 Parsing the search term

Upon receiving a GET request to a resource of this form the server where the `category-id` component of the path is not a valid category-id used in the "uc/categories" resource then the server MUST immediately return a 404 error.

Otherwise the server MUST set the local variable `term` to be the value of the `category-id` segment of the path.

Then the server MUST proceed to parsing the Query Parameters as described in Sections 4.19.3 and 4.29.2.

4.29.2 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. All query parameters described in that section are valid for these resources except for `category`. For requests to resources of this form the value of `offset` must not be negative.

After parsing the query parameters the server MUST proceed to the next section.

4.29.3 Assembling the Content Lists

In this section the server will assemble a list of content items which corresponds to the terms of the search. This list may potentially be infinitely long, so it is necessary for server implementations to assemble the list only as it is required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST be immediately set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an additional number of days equal to (`days` − 1).

The server MUST assemble a list of content, all pieces of content in the list MUST be in the category identified by the category-id stored in the local `term` variable, and all pieces of content in that category SHOULD be in the list.

It is REQUIRED that if pieces of content in the list are currently available for presentation then one of the pieces which is currently available should be at list index 0 and all other pieces currently available for presentation SHOULD follow it before any pieces of content which are not currently available for presentation.

Otherwise the list MAY be ordered in any order which the server implementer wishes, but it is RECOMMENDED that all AV content from sources with the linear property should be arranged in order of start time.

It is REQUIRED that repeated requests to this resource with the same parameters will produce the same ordering.

Upon generating this list the server MUST filter it as described in Section 4.19.4, and then respond to the request as described therein.

4.30 The uc/search/global-content-id Resource

This resource MUST NOT be implemented if the server does not implement the "uc/sources" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/global-content-id" resource is being implemented.

The purpose of resources of the form "uc/search/global-content-id/{term}" is to allow clients to determine the content known to the server that is associated with the specified `global-content-id`.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.31 The uc/search/global-content-id/{gcid} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/global-content-id/{gcid}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.31.1 Parsing the search term

Upon receiving a GET request to a resource of this form the server MUST immediately unpercent-encode the component `gcid` of the path and store the result in the local variable `term`.

If the value of the local variable `term` is not a valid `global-content-id` (as determined by whatever criteria the server uses for such ids) the server MUST immediately return a 404 error.

Otherwise the server MUST proceed to parsing the Query Parameters as described in Sections 4.19.3 and 4.31.2.

4.31.2 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. The query parameters which are valid for requests to this resource are: `results`, `offset`, `sid`, `start`, `end`, and `days`. For requests to resources of this form the value of `offset` must not be negative.

After parsing the query parameters the server MUST proceed to the next section.

4.31.3 Assembling the Content Lists

In this section the server will assemble a list of content items which corresponds to the terms of the search. This list may potentially be infinitely long, so it is necessary for server implementations to assemble the list only as it is required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST be immediately set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an additional number of days equal to (`days` − 1).

The server MUST assemble a list of content, all pieces of content in the list MUST have the global-content-id which is stored in the local variable `term`, and all pieces of content with that global-content-id SHOULD be in the list.

It is REQUIRED that if pieces of content in the list are currently available for presentation then one of the pieces which is currently available should be at list index 0 and all other pieces currently available for presentation SHOULD follow it before any pieces of content which are not currently available for presentation.

Otherwise the list MAY be ordered in any order which the server implementer wishes, but it is RECOMMENDED that all AV content from sources with the linear property should be arranged in order of start time.

It is REQUIRED that repeated requests to this resource with the same parameters will produce the same ordering.

Upon generating this list the server MUST filter it as described in Section 4.19.4, and then respond to the request as described therein.

4.32 The uc/search/global-series-id Resource

This resource MUST NOT be implemented if the server does not implement the "uc/sources" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/global-series-id" resource is being implemented.

The purpose of resources of the form "uc/search/global-series-id/{term}" is to allow clients to determine the content known to the server that is associated with the specified `global-series-id`.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.33 The uc/search/global-series-id/{gsid} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/global-series-id/{gsid}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.33.1 Parsing the search term

Upon receiving a GET request to a resource of this form the server MUST immediately unpercent-encode the component `gsid` of the path and store the result in the local variable `term`.

If the value of the local variable `term` is not a valid global-series-id (as determined by whatever criteria the server uses for such ids) the server MUST immediately return a 404 error.

Otherwise the server MUST proceed to parsing the Query Parameters as described in Sections 4.19.3 and 4.33.2.

4.33.2 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. The query parameters which are valid for requests to this resource are: `results`, `offset`, `sid`, `start`, `end`, and `days`. For requests to resources of this form the value of `offset` must not be negative.

After parsing the query parameters the server MUST proceed to the next section.

4.33.3 Assembling the Content Lists

In this section the server will assemble a list of content items which corresponds to the terms of the search. This list may potentially be infinitely long, so it is necessary for server implementations to assemble the list only as it is required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST be immediately set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an additional number of days equal to (`days` − 1).

The server MUST assemble a list of content, all pieces of content in the list MUST have the global-series-id which is stored in the local variable `term`, and all pieces of content with that global-series-id SHOULD be in the list.

It is REQUIRED that if pieces of content in the list are currently available for presentation then one of the pieces which is currently available should be at list index 0 and all other pieces currently available for presentation SHOULD follow it before any pieces of content which are not currently available for presentation.

Otherwise the list MAY be ordered in any order which the server implementer wishes, but it is RECOMMENDED that all AV content from sources with the linear property should be arranged in order of start time.

It is REQUIRED that repeated requests to this resource with the same parameters will produce the same ordering.

Upon generating this list the server MUST filter it as described in Section 4.19.4, and then respond to the request as described therein.

4.34 The uc/search/global-app-id Resource

This resource MUST NOT be implemented if the server does not implement the "uc/sources" resource and the "uc/apps" resource. In all other circumstances, implementing this resource is OPTIONAL. The remainder of this section will assume that the "uc/search/global-app-id" resource is being implemented.

The purpose of resources of the form "uc/search/global-app-id/{term}" is to allow clients to determine the interactive content known to the server that is associated with the specified `global-app-id`.

The server MUST respond to a GET request to this resource by returning a 204 response with no message body.

This resource does not undergo notifiable changes.

4.35 The uc/search/global-app-id/{gaid} Resources

Implementing resources of this form is REQUIRED if the parent resource is implemented and it MUST NOT be implemented otherwise. The remainder of this section will assume the the "uc/search/global-app-id/{gaid}" resources are being implemented.

Resources of this form MUST NOT undergo notifiable changes.

4.35.1 Parsing the search term

Upon receiving a GET request to a resource of this form the server MUST immediately store the component `gaid` of the path in the local variable `term`.

If the value of the local variable `term` is not a valid global-app-id the server MUST immediately return a 404 error.

Otherwise the server MUST proceed to parsing the Query Parameters as described in Sections 4.19.3 and 4.35.2.

4.35.2 Parsing the Query Parameters

The server MUST parse the query parameters on a request as described in Section 4.19.3. The query parameters which are valid for requests to this resource are: `results`, `offset`, `sid`, `start`, `end`, and `days`. For requests to resources of this form the value of `offset` must not be negative.

After parsing the query parameters the server MUST proceed to the next section.

4.35.3 Assembling the Content Lists

In this section the server will assemble a list of content items which corresponds to the terms of the search. This list may potentially be infinitely long, so it is necessary for server implementations to assemble the list only as it is required for the behaviour described in later subsections.

If the local variable `days` is set then the local variable `end` MUST be immediately set to be equal to the next midnight UTC which follows the date and time of the `start` variable, plus an additional number of days equal to (`days` − 1).

The server MUST assemble a list of content, all pieces of content in the list MUST be interactive, and MUST have the `global-app-id` which is stored in the local variable `term`, and all pieces of content with that `global-app-id` SHOULD be in the list.

It is REQUIRED that if pieces of content in the list are currently available for presentation then one of the pieces which is currently available should be at list index 0 and all other pieces currently available for presentation SHOULD follow it before any pieces of content which are not currently available for presentation.

Otherwise the list MAY be ordered in any order which the server implementer wishes.

It is REQUIRED that repeated requests to this resource with the same parameters will produce the same ordering.

Upon generating this list the server MUST filter it as described in Section 4.19.4, and then respond to the request as described therein.

4.36 The uc/categories Resource

4.36.1 Introduction

This section is purely informative.

The "uc/categories" resource exists to convey a server's category hierarchy to clients. See section 2.3 for information on categories and normative requirements regarding the server's category hierarchy. The implementation section below contains normative requirements for the presentation of the hierarchy to clients via this resource.

4.36.2 Implementation

Implementing this resource is OPTIONAL if the server also implements the "uc/search/categories" resource, it MUST NOT be implemented otherwise. The remainder of this section assumes that the resource is being implemented. Representations of this resource MUST comply with the schema for a response element containing a "categories" element defined in Appendix D. If this resource is implemented then the server MUST also implement the "uc/sources", "uc/source-lists", and "uc/search/categories" resources. Depending upon implementation details the "uc/events" resource may also be REQUIRED.

The server MUST respond to any GET request made to this resource by returning a representation such that:

- The representation MUST contain a single `categories` element representing the root of the server's category hierarchy.
- The `categories` element MUST contain one or more `category` elements that represent the children of the hierarchy root.
- In turn, each `category` element MUST contain one or more `category` child elements representing the associated category's own children if it has any, and MUST NOT contain child `category` elements otherwise.
- Every `category` element is subject to the following requirements:
  - The `category` element MUST have a `name` attribute whose value is the human-readable name of the category.
  - If the category associated with the `category` element has a `category-id`, the element MUST have a `category-id` attribute whose value is that `category-id`.
  - The `category` element MAY have a `logo-href` attribute whose value is a URL that points to an image that the client can use to represent the category to the user.

A resource of this form MUST undergo a notifiable change whenever the category hierarchy itself changes. This resource MUST NOT undergo a notifiable change simply because a change occurs to the categories associated with one or more pieces of content. It is possible to implement this resource in such a way that it will never undergo notifiable changes, in which case the "uc/events" resource is NOT REQUIRED.

An example XML document that fulfils the requirements for a representation of the "uc/categories" resource is as follows:

```
<response resource="uc/categories">
  <categories>
    <category name="Genres">
      <category name="Children's" category-id="child">
        <category name="Factual" category-id=fact"/>
      </category>
```

```
      <category name="Comedy" category-id="come"/>
      <category name="Drama" category-id="dram"/>
      <category name="Entertainment" category-id="ente"/>
      <category name="Factual" category-id="fact1">
        <category name="Arts, Culture &amp; the Media" category-id="arts"/>
        <category name="Cars &amp; Motors" category-id="cars"/>
        <category name="Consumer" category-id="cons"/>
      </category>
      <category name="Films" category-id="film"/>
      <category name="Learning" category-id="lear"/>
      <category name="Lifestyle &amp; Leisure" category-id="life"/>
      <category name="News" category-id="news">
        <category name="National News" category-id="nati"/>
        <category name="Local/Regional News" category-id="loca"/>
        <category name="Weather" category-id="weat"/>
      </category>
    </category>
    <category name="Featured" category-id="feat"/>
    <category name="Signed" category-id="sign"/>
    <category name="Audio Described" category-id="audi"/>
  </categories>
</response>
```

4.37 The uc/acquisitions Resource

4.37.1 Introduction

This section is purely informative.

The purpose of this resource is to provide access to a list of content which is not currently accessible for presenting on the box's outputs, but is expected to become so in the future, and is expected to remain accessible indefinitely after that time in storage media under the control of the box (represented by the "uc/storage" resource, if implemented - see section 4.39).

If the box supports the concept of user-initiated acquisitions, server implementors should use this resource to make information about acquisitions available to the client. Not all forms of acquisition are appropriate for representation by this resource. For example, "push-VOD" functionality in a box, whereby a box uses idle tuners to record digital TV content selected by broadcasters may or may not be suited to representation by this resource. As a general guideline, server implementors should use this resource to represent acquisitions that would be *individually* represented to and modifiable by users via the box's built-in interface, and should not use this resource to represent acquisitions that are completely hidden from the user.

4.37.2 Implementation

Implementing this resource is OPTIONAL if the server also implements the "uc/sources" resource, and it MUST NOT be implemented otherwise. The remainder of this section assumes that it is being implemented. Representations of this resource MUST comply with the schema for a response element containing an "acquisitions" element defined in Appendix D. If this resource is implemented then the "uc/sources", "uc/source-lists", "uc/search/sources", and "uc/events" resources MUST also be implemented.

The server MUST respond to any GET request to this resource by returning a representation such that:

- The representation MUST contain a single `acquisitions` element.

- The `acquisitions` element SHOULD contain one `content-acquisition` element for each "content acquisition" in the maintained list, and one `series-acquisition` element for each "series acquisition" in the maintained list.
- Each `content-acquisition` element MUST give its `acquisition-id` in the `acquisition-id` attribute.
- Each `content-acquisition` element MUST contain a `sid` and a `cid` attribute which give a `source-id` and `content-id` from which the content is expected to be acquired.
- Each `content-acquisition` element MUST contain a `interactive` attribute which MUST be "true" if the content is interactive, and "false" otherwise.
- Each `content-acquisition` element SHOULD contain a `global-content-id` attribute which indicates a global content identifier (such as a CRID) if one is known for this content.
- Each `content-acquisition` element SHOULD contain a `series-id` attribute which indicates a series identifier if one is known for this programme.
- Each `content-acquisition` element SHOULD contain a `series-linked` attribute which SHOULD be set to `true` if the booking is "series-linked" and `false` otherwise. If this attribute is missing a client MAY assume that it is `false`.
- Each `content-acquisition` element SHOULD contain a `priority` attribute which SHOULD be used to indicate whether it is a "priority" acquisition, if the box supports a concept of priority acquisitions. If this attribute is missing then a client MAY assume that it is `false`.
- Each `content-acquisition` element MAY contain a `start` attribute which if present SHOULD be used to indicate the expected time when the acquisition will begin.
- Each `content-acquisition` element MAY contain an `end` attribute which if present SHOULD be used to indicate the expected time when the acquisition will become available.
- Each `content-acquisition` element SHOULD contain a `speculative` attribute which if present MUST be set to `true` if the acquisition is "speculative" and `false` otherwise. If not present a client MAY assume that this attribute is `false`.
- Each `content-acquisition` element MAY contain a `active` attribute which if included SHOULD be `true` if the content is currently in the process of being acquired and MUST be `false` otherwise.
- Each `series-acquisition` element MUST contain an `acquisition-id` and `series-id` attribute exactly as described for the `content-acquisition` element, the `acquisition-id` used in the `acquisition-id` attribute MUST be unique, and not shared with any other element.
- Each `series-acquisition` element SHOULD contain a `speculative` attribute which if present MUST be set to `true` if the acquisition is "speculative" and `false` otherwise. If not present a client MAY assume that this attribute is `false`.

Items in the list SHOULD be returned in the same order each time a representation of the resource is returned to a client GET request. The order of items in the list SHOULD correspond to the order of the corresponding list(s) in the box's built-in interface, if such a list (or lists) exists.

An example XML document that fulfils the requirements for a representation of the "uc/acquisitions" resource generated in response to a GET request as described above is as follows:

```
<response resource="uc/acquisitions">
  <acquisitions>
    <series-acquisition
        acquisition-id="s0"
        series-id="crid%3A%2F%2Fbbc.co.uk%2F__SERBBCLondonNews"/>
    <content-acquisition
        acquisition-id="p0"
        sid="BBCOne"
        cid="2009-12-14T22%3a25%3a00Z"
        interactive="false"
        global-content-id="crid://bbc.co.uk/272951840"
        series-id="crid%3A%2F%2Fbbc.co.uk%2F__SERBBCLondonNews"
        series-linked="true"
        priority="false"
        start="2009-12-14T22:25:00Z"
        end="2009-12-14T22:35:00Z"
        active="false"/>
  </acquisitions>
</response>
```

Upon receiving a POST request to this resource the server MUST examine the query parameters given and respond as follows:

- If the query parameters `sid` and `content-id` are set and the query parameters `global-content-id` and `series-id` are not set, the box SHOULD attempt to book a new "content acquisition" for the content identified by the given `content-id` from the source identified by `sid`. If the box is unable to identify content with the supplied `content-id` in the source with the supplied `source-id` then the server MUST return a 400 error; if such a piece of content can be identified but booking the acquisition fails then the server MUST immediately return a 500 error. The box MAY use any mechanism at all to locate and acquire the content. If the `priority` query parameter was also present and set to "true" then the box SHOULD record that the new acquisition is a "priority" acquisition. If booking the acquisition is successful then the box MUST return a response identical to that which would be returned if the client had made a GET request to the "uc/acquisitions/{id}/" resource corresponding to the newly created "content acquisition".

- If the query parameter `global-content-id` is set, and the query parameters `sid`, `content-id`, and `series-id` are not set the box SHOULD attempt to book a new "content acquisition" for the content specified by the `global-content-id` (after it has been un-percentage encoded). If the box is unable to identify a piece of content with the supplied `global-content-id` then the server MUST return a 400 error; if such content can be identified but booking the acquisition fails then the server MUST immediately return a 500 error. The box MAY use any mechanism at all to locate and acquire the content. If the `priority` query parameter was also present and set to "true" and if the box supports the concept of priority acquisitions, then the box SHOULD record that the new acquisition is a "priority" acquisition. If booking the acquisition is successful then the box MUST return a response identical to that which would be returned if the client had made a GET request to the "uc/acquisitions/{id}/" resource corresponding to the newly created "content acquisition".

- If the query parameter `series-id` is set, and the query-parameters `sid`, `content-id`, and `global-content-id` are not set the box SHOULD attempt to book a new "series acquisition" for the series specified by the `series-id`. If the box is unable to identify a series with the supplied `series-id` then the server MUST return a 400 error; if such a series can be identified

but booking the acquisition fails then the server MUST immediately return a 500 error. The box MAY use any mechanism at all to locate the series. If booking the acquisition is successful then the box MUST return a response identical to that which would be returned if the client had made a GET request to the "uc/acquisitions/{id}/" resource corresponding to the newly created "series acquisition".

- In all other cases the box MUST immediately return a 400 error.

The booking of a new acquisition or the cancellation of a booked acquisition MUST trigger a notifiable change in this resource. The booking of a speculative or series-linked acquisition automatically by the box SHOULD also trigger a notifiable change in this resource.

4.38 The uc/acquisitions/{id} Resources

One resource of this form MUST be implemented for each acquisition listed in the response from the "uc/acquisitions" resource if that resource is implemented and MUST NOT be implemented otherwise. The rest of this section assumes that these resources are being implemented. Representations of this resource MUST comply with the schema for a response element containing an "acquisition" element defined in Appendix D. If these resources are implemented then the "uc/sources", "uc/source-lists", "uc/search/sources", and "uc/events" resources MUST also be implemented.

The purpose of this resource is to identify and allow interaction with a particular booked acquisition.

The server MUST respond to any GET request to a resource of this form with a value for "{id}" which does not not correspond to any `acquisition-id` maintained for the parent resource by returning a 404 error.

The server MUST respond to any GET request to a resource of this form for which the value of "{id}" is equal to an `acquisition-id` corresponding to a "content acquisition" by returning a representation containing a single `content-acquisition` element identical to the one with that `acquisition-id` which would be returned in the response to a GET request to the parent resource.

The server MUST respond to any GET request to a resource of this form for which the value of "{id}" is equal to an `acquisition-id` corresponding to a "series acquisition" by returning a representation containing a single `series-acquisition` element identical to the one with that `acquisition-id` which would be returned in the response to a GET request to the parent resource.

Upon receiving a DELETE request to a resource of this form for which the value of "{id}" is equal to a known `acquisition-id` the server SHOULD attempt to cancel the content or series acquisition identified by the `acquisition-id` given in the path of the URI. If no such booking exists after the request has been processed then the server MUST return a 204 response with no response body regardless of whether or not an acquisition has been cancelled, otherwise it MUST return a 500 error.

Changes to this resource are never directly notifiable, but cancelling an acquisition SHOULD trigger a notifiable change in the "uc/acquisitions" resource.

An example XML document that fulfils the requirements for a representation of the "uc/acquisitions/{id}" resource is as follows, for the case where {id}=0:

```
<response resource="uc/acquisitions/0">
  <content-acquisition
      acquisition-id="0"
      sid="BBCOne"
      cid="2009-12-14T22%3a25%3a00Z"
      interactive="false"
      global-content-id="crid://bbc.co.uk/272951840"
      series-id="crid%3A%2F%2Fbbc.co.uk%2F__SERBBCLondonNews"
```

```
      series-linked="true"
      priority="false"
      start="2009-12-14T22:25:00Z"
      end="2009-12-14T22:35:00Z"
      active="false"/>
</response>
```

4.39 The uc/storage Resource

4.39.1 Introduction

This section is purely informative.

The purpose of this resource is to provide an interface to a list of stored content (see Section 2.2). For example, if the box has an internal hard-drive which is used to store content recorded from DVB transmissions or downloaded from the Internet then content stored on that hard-drive could be listed here, with a "stored-content" element in representations of this resource representing each locally stored piece of content.

The intention of this resource is to support "storage management" use cases, particularly the freeing up of storage space by deleting unwanted stored content. "Media playback" use-cases were not considered in the design of this resource, and servers that want to allow remote clients to initiate the presentation of stored content should do so by listing them in a source in the source-list with the reserved `list-id` "uc_storage" defined in section 2, representing each piece of stored content as a piece of content available from that source in all the other parts of the API that the server implements.

In order to be able to respond to requests to this resource the server needs to have access to details of stored content, a record of how much storage capacity the storage media have, and a record of how much capacity each piece on the list of stored content uses. There is no requirement that this list be complete. This interface does not show separate storage media separately. A box with access to multiple storage media should manage the distribution of content between them internally and conceal this detail from the client.

There is no requirement that server implementors use this resource to represent all forms of storage media connected to the device. Use cases such as simple playback of media files from a USB device connected to the STB can and should be implemented without listing the files as `storage` elements in this resource at all (see section 2.7 for an example of how to model this functionality within the UC API). As a guideline to implementors, storage media should be represented by this resource if the user is also given the ability to record (or otherwise acquire) content onto it and delete content from it using the box's built-in interface.

4.39.2 Implementation

Implementing this resource is OPTIONAL. Implementing it is RECOMMENDED if the server also implements the "uc/sources" resource and one or more sources in the source-list with the reserved `list-id` "uc_storage" defined in section 2. The remainder of this section assumes that it is being implemented. Representations of this resource MUST comply with the schema for a response element containing a "storage" element defined in Appendix D. If this resource is implemented then the "uc/sources", "uc/source-lists" and "uc/search/sources" resources MUST also be implemented.

The server MUST respond to a GET request made to this resource by returning a representation such that:

- The representation MUST contain a single `storage` element.
- The `storage` element MAY have a `size` attribute containing an integer representing the size of the total storage space in bytes.

- The `storage` element MAY have a `free` attribute containing an integer representing the size of the storage space available for new entities, in bytes. Servers need not take filesystem block sizes or the division of storage between multiple volumes into account when calculating this value.
- The `storage` element SHOULD contain one `stored-content` element for each piece of stored content managed by the box.
- If the server implements the "uc/sources" resource then each `stored-content` element MUST have a `sid` attribute that MUST correspond to the `source-id` of a source in the source-list with the reserved `list-id` "uc_storage" (see section 2).
- Each `stored-content` element MUST have a `cid` attribute set to a `content-id` which uniquely identifies the content within the source identified by the `sid` attribute if one is provided, and uniquely identifies the content within this list otherwise.
- Each `stored-content` element MAY contain a `created-time` attribute, used to indicate the time at which the entity was created (in the Internet standard timestamp format described in section 1.3).
- Each `stored-content` element MAY have a `size` attribute set to an integer describing the size of the entity in bytes.
- Each `stored-content` element MAY have a `global-content-id` attribute

Items in the list SHOULD be returned in the same order each time a representation of the resource is returned to a client GET request. The order of items in the list SHOULD correspond to the order of the corresponding list in the box's built-in interface, if one exists.

The addition or removal of stored content from the local storage MUST cause this resource to undergo a notifiable change. Changes in the data conveyed by any of the attributes of the `stored-content` elements MUST also cause this resource to undergo a notifiable change. It is possible that this resource could be implemented in such a way that it would never generate notifiable changes, though such an implementation is unlikely to be very useful.

An example XML document that fulfils the requirements for a representation of the "uc/storage" resource is as follows:

```
<response resource="uc/storage">
  <storage size="6442450944" free="5368709120">
    <stored-content sid="hdd" cid="file0"
        created-time="2009-12-14T22:00:00Z"
        size="1073741824"/>
  </storage>
</response>
```

4.40 The uc/storage/{sid} Resources

One resource of this form MUST be implemented for each source in the "uc_storage" list if "uc/sources" and the parent resource are implemented. If the "uc/storage" resource is not implemented then these resources MUST NOT be implemented. The remainder of this section assumes that they are being implemented. Representations of this resource MUST comply with the schema for a response element containing a "storage" element defined in Appendix D. If this resource is implemented then the "uc/sources", "uc/source-lists" and "uc/search/sources" resources MUST also be implemented.

The purpose of this resource is to provide a list of those pieces of stored-content which fall into a particular source.

The server MUST respond to any GET request to a resource of this form with a value for "{sid}" which is not equal to a `source-id` that appears in the "uc_sources" source-list by returning a 404 error.

The server MUST respond to any other GET request to a resource of this form exactly as it would a request to "uc/storage" except that the the `size` and `free` attributes of the `storage` element MUST either not be included or represent only the total and available storage space for the source identified by the "{sid}" path-component, and the returned representation MUST contain one `stored-content` element for each piece of stored content in the source identified by the "{sid}" path-segment, and MUST NOT contain any other `stored-content` elements.

The addition or removal of stored content from the local storage source identified by "{sid}" MUST cause a resource of this form to undergo a notifiable change. Changes in the data conveyed by any of the attributes of the `stored-content` elements MUST also cause these resources to undergo notifiable changes. It is possible that these resources could be implemented in such a way that they would never generate notifiable changes, though such an implementation is unlikely to be very useful.

An example XML document that fulfils the requirements for a representation of the "uc/storage/hdd" resource is as follows:

```
<response resource="uc/storage/hdd">
  <storage size="6442450944" free="5368709120">
    <stored-content sid="hdd" cid="file0"
        created-time="2009-12-14T22:00:00Z"
        size="1073741824"/>
  </storage>
</response>
```

4.41 The uc/storage/{sid}/{cid} Resources

One resource of this form MAY be implemented for each entity in the local storage of the box if "uc/storage" is implemented. If the "uc/storage" resource is not implemented then these resources MUST NOT be implemented. The remainder of this section assumes that they are being implemented. Representations of this resource MUST comply with the schema for a response element containing a "stored-content" element defined in Appendix D. If these resources are implemented then the "uc/sources", "uc/source-lists", and "uc/search/sources" resources MUST also be implemented.

The purpose of this resource is to convey information about and allow interaction with a specific stored entity.

The server MUST respond to any GET request to a resource of this form with a value for "{sid}" which is not equal to a `source-id` that appears in the "uc_sources" source-list by returning a 404 error.

The server MUST respond to any GET request to a resource of this form with a value for "{cid}" which is not equal to a `content-id` that appears as the `cid` attribute of a `stored-content` element in a representation of the "uc/storage/{sid}" resource by returning a 404 error.

The server MUST respond to any other GET request to a resource of this form by returning a representation containing a single `stored-content` element that is identical to the one that would be returned as part of a representation of the "uc/storage/{sid}" resource. The `stored-content` element returned in representations of this resource should represent the stored programme with a `content-id` equal to the value of "{cid}".

Upon receiving a DELETE request to a resource of this form, the server MAY immediately return a 405 error (for example if it does not support the remote deletion of stored content). If the server does not immediately return a 405 error then the box MUST attempt to delete the stored content identified by "{cid}" in the source identified by "{sid}" and free up all the associated

resources. If there is no stored content with a `content-id` equal to "{cid}" in the source identified by "{sid}" after the request has been processed then the server SHOULD return a 204 response with no response body regardless of whether or not a resource was actually deleted. If the attempt to delete the stored content fails, the server MUST return a 500 error.

Resources of this form SHOULD NOT undergo notifiable changes, but when pieces of stored content are deleted the "uc/storage" and "uc/storage/{sid}" resources MUST undergo notifiable changes.

An example XML document that fulfils the requirements for a representation of the "uc/storage/{sid}/{cid}" resource is as follows, for the case where {sid}=`hdd` and {cid}=`file0`:

```
<response resource="uc/storage/hdd/file0">
  <stored-content sid="hdd" cid="file0"
      created-time="2009-12-14T22:00:00Z"
      size="1073741824"/>
</response>
```

4.42 The uc/credentials Resource

Implementing this resource is OPTIONAL if the box supports the security scheme and it MUST NOT be implemented otherwise. The remainder of this section assumes that it is being implemented. Representations of this resource MUST comply with the schema for a response element containing a "credentials" element defined in Appendix D. If this resource is implemented then the "uc/events" resource MUST also be implemented.

The server MUST respond to a GET request to this resource by returning a representation such that:

- The representation MUST contain a single `credentials` element.
- The `credentials` element MUST contain one `client` element for each `client-id` with authentication credentials and a client name stored persistantly by the box. This does not include pending `client-id`s which have been POSTed to "uc/security", but have not yet been used to authenticate a request.
- Each `client` element MUST have a single `client-id` attribute, which must be equal to the `client-id` the element represents represented as the string representation of a UUID.
- Each `client` element MUST have a single `name` attribute set to the client name associated with the relevant `client-id`.

Items in the list SHOULD be returned in the same order each time a representation of the resource is returned to a client GET request. The order of items in the list SHOULD correspond to the order of the corresponding list in the box's built-in interface, if one exists.

This resource MUST undergo a notifiable change every time a new `client-id` is used for authentication.

An example XML document that fulfils the requirements for a representation of the "uc/credentials" resource is as follows:

```
<response resource="uc/credentials">
  <credentials>
    <client client-id="550e8400-e29b-41d4-a716-446655440000" name="Bob's Generic Phone"/>
  </credentials>
</response>
```

4.43 The uc/credentials/{id} Resources

One resource of this form MUST be implemented for each `client-id` currently stored by the box if the "uc/credentials" resource is implemented and MUST NOT be implemented otherwise. This resource has no representation format. If these resources are implemented then the "uc/events" resource MUST also be implemented.

Different resources of this form MAY require authentication with different `client-id`/LSGS pairs.

Upon receiving a DELETE request to a resource of this form, the server MUST attempt to de-authorise the `client-id` which, when represented as a string representation of a UUID is equal to the "{id}" in the resource's URI. If no such `client-id` is authorised to connect to resources on the server after the request has been processed, the server MUST return a 204 response with no response body, regardless of whether or not a `client-id` was de-authorised while processing the response, otherwise it MUST return a 500 error.

DELETE requests to this resource are very likely to be protected and may well require that the client be authenticated in order to make them, perhaps even requiring that the client be authenticated with a particular client-id. This is, however, left up to the individual implementer to decide.

Resources of this form SHOULD NOT generate notifiable changes.

4.44 The uc/apps Resource

4.44.1 Introduction

This section is entirely informative

The purpose of this resource is to provide information about the interactive content currently active on the box and optionally to allow clients to request that interactive content be activated without presenting them on an output (this may not make sense in all cases).

This resource is provided for two reasons:

- To allow remote clients to request that interactive content be activated and deactivated without also requesting that they be presented on an output.
- To allow remote clients to identify when individual pieces of interactive content are able to directly interact with them using the API extension mechanism defined in section 4.46

It is assumed that both the interactive content and the clients that communicate with it are provided by the same organisation, or that communication between them is standardised elsewhere. That being the case, no aspect of the communication between interactive content and remote clients is standardised in this document.

4.44.2 Definitions

This section is normative if the "uc/apps" resource is being implemented.

We define "remote-enabled" interactive content to be that content which has the capability to extend the UC API in the manner described in section 4.46 AND which is able to respond to requests to its dynamic API extensions at a given point in time. Not all applications with the capability to extend the API may be permitted to do so. For example, a box or server MAY implement a mandatory internal registration mechanism for interactive content that wish to extend the API. A server or box MAY implement any other such restriction mechanisms as are considered appropriate. A piece of interactive content that is *capable* of extending the server API may thus be temporarily or permanently unable to do so on a specific box at a specific point in time.

For the purposes of this resource (and its sub-resources) interactive content which is currently being presented on an output, capable of interacting with a client via the extension mechanism, or otherwise "running" is considered to be "active", whilst content which is not is called "inactive".

4.44.3 Implementation

Implementing this resource is OPTIONAL. The remainder of this section assumes that it is being implemented. Representations of this resource MUST comply with the schema for a response element containing an "apps" element defined in Appendix D. If this resource is implemented then the "uc/events" resource MUST also be implemented.

The server MUST respond to a GET request to this resource by returning a representation such that:

- The representation MUST contain a single `apps` element.
- The `apps` element MUST contain one `app` element for each piece of interactive content which is currently active.
- Each `app` element MUST have a single `sid` attribute, which MUST contain the `source-id` of the content in question.
- Each `app` element MUST have a single `cid` attribute, which MUST contain the `content-id` of the content in question.
- Each `app` element MUST have a single `global-app-id` attribute, which MUST contain the `global-app-id` of the content in question. (See section 2.2 for a definition of this identifier.)
- Each `app` element that represents a piece of content that, at the point in time when the response is being prepared, is remote-enabled according to the definition above, MUST have a single `remote-enabled` attribute which MUST take the value "true". If the content is not remote-enabled, this attribute MUST either be absent or take the value "false".

Items in the list SHOULD be returned in the same order each time a representation of the resource is returned to a client GET request. The order of items in the list SHOULD correspond to the order of the corresponding list in the box's built-in interface, if one exists.

If the server does not allow clients to request the activation of interactive content, upon receiving a POST request to this resource, the server MUST return a 405 error. The rest of this section assumes that the server allows clients to request the activation of interactive content.

Upon receiving a POST request to this resource with no `sid` query parameter in its URL, with more than one `sid` or `cid` parameter, for which the value of the single `sid` query parameter is not a valid `source-id`, or for which the value of the single `cid` parameter is not the `content-id` of a piece of interactive content in the source identified by the value of the `sid` parameter the server MUST return a 400 error.

Upon receiving any other POST request to this resource with a valid `source-id` as the value of the single `sid` query parameter, the server MUST take one of the following actions:

1. If the request also has a single `cid` parameter which is a valid `content-id` for a piece of interactive content within the source then the server MUST immediately take steps to activate the identified content. This MAY result in changes to the state of an output, and if so MUST trigger a notifiable change in that output.
2. If the request has no `cid` parameter then the box MUST attempt to determine a default piece of interactive content for the identified source to attempt to activate, and proceed as if this item had been specified. If it is unable to find such a piece of content then it MUST immediately return a 404 error.

If an attempt to start a piece of interactive content immediately fails the box MUST immediately return a 500 error. Otherwise it MUST return the same result as would be returned by a GET

request to the sub-resource "uc/apps/{global-app-id}", where in this case "{global-app-id}" is replaced with the global-app-id of the piece of content which has now been activated.

This resource MUST undergo a notifiable change whenever an event occurs that would result in a change to the representation returned by a GET request to this resource.

Servers SHOULD take steps to prevent malicious interactive content from masquerading as trusted content, perhaps by only allowing interactive content from trusted sources to register for API extension. Servers implementors should bear in mind that malicious clients may masquerade as trusted ones. It may be appropriate to place restrictions on the behaviour of interactive apps that extend the API: preventing them from accessing personally-identifying information about the users of the box, for example.

An example XML document that fulfils the requirements for a representation of the "uc/apps" resource is as follows:

```
<response resource="uc/apps">
  <apps>
    <app sid="BBCOne"
         cid="dvb%3A%2F%2F3b1a.321.4362%2F0"/>
    <app sid="apps_on_disk"
         cid="file76"
         global-app-id="bbc.co.uk:CLAS010739-V-U-1.2.1"
         remote-enabled="true">
  </apps>
</response>
```

4.45 The uc/apps/{global-app-id} Resources

One resource of this form MUST be implemented for each piece of interactive content listed in the response from the "uc/apps" resource if that resource is implemented and MUST NOT be implemented otherwise. The rest of this section assumes that these resources are being implemented. Representations of this resource MUST comply with the schema for a response element containing an "app" element defined in Appendix D. If these resources are implemented then the "uc/events" resource MUST also be implemented.

The purpose of this resource is to allow clients to request modifications to the state of particular pieces of interactive content, and to request their deactivation.

The server MUST respond to any GET request to a resource of this form for which the path segment placeholder "{global-app-id}" is replaced with the `global-app-id` of a piece of interactive content that is currently active by returning a representation containing a single `app` element that is equivalent to the `app` element with the same values of `sid` and `cid` that would be returned in a GET request to the parent resource.

The server MUST respond to any other GET request to a resource of this form by returning a 404 error.

If a server does not permit remote clients to request the deactivation of interactive content, then upon receiving a DELETE request to a resource of this form, the server MUST return a 405 error.

Otherwise, upon receiving a DELETE request to a resource of this form the server SHOULD attempt to deactivate any interactive content whose `global-app-id` is as specified in the path. The meaning of "deactivate" is implementation-specific, but MUST include include removal from the list of interactive content reported by a GET request to the parent resource. If no such interactive content is currently active then the server MUST return a 204 response with no response body. If such interactive content is successfully deactivated then the server MUST return a 204 response with no response body. If an error occurs while deactivating the content, the server MUST return a 500 error.

At the point in time when a piece of interactive content is deactivated, the server SHOULD interrupt any pending requests to the API extension resource(s) implemented by the content by returning a 410 error to each one. This is the case regardless of whether the interactive content is deactivated due to a DELETE request to a resource of this form or for some other reason.

Changes to this resource are never directly notifiable.

An example XML document that fulfils the requirements for a representation of the "uc/apps/{global-app-id}" resource is as follows, for the case where {global-app-id}=`bbc.co.uk:f0sb5qa1`

```
<response resource="uc/apps/bbc.co.uk:f0sb5qa1">
  <app sid="apps_on_disk"
       cid="file76"
       global-app-id="bbc.co.uk:f0sb5qa1"
       remote-enabled="true" />
</response>
```

4.46 The uc/apps/{global-app-id}/ext Resource Hierarchies

In this section, when we refer to a "resource hierarchy", we mean the set of resources that share the URL path component prefix "uc/apps/{global-app-id}/ext", where the path segment placeholder {global-app-id} is replaced with the `global-app-id` of a piece of interactive content. For example, the resource "uc/apps/example.com:foo/ext/bar" is a member of the "uc/apps/example.com:foo/ext" resource hierarchy for the piece of interactive content with the `global-app-id` "example.com:foo".

These resource hierarchies are intended to allow interactive content running on the same box as the server to communicate with remote devices by defining an "API extension" that exists only when the interactive content in question is both "active" and "remote-enabled"[5]. This API extension takes the form of a resource hierarchy associated with the interactive content as described above. It is up to the interactive content author to define and describe the API extension implemented by their own content, although some guidance on the subject is given in section 6 of this document.

One resource hierarchy of this form MUST be implemented for each active and remote-enabled interactive content if the box has implemented the "uc/apps" resource, and MUST NOT be implemented otherwise. The rest of this section assumes that one or more of these resource hierarchies are being implemented. If these resources are implemented then the "uc/events" resource MUST also be implemented.

Each resource hierarchy of this type has a different URL path component prefix in which the placeholder "{global-app-id}" in "uc/apps/{global-app-id}/ext" is replaced with the `global-app-id` of a piece of active and remote-enabled interactive content.

If the server receives an HTTP request to a resource with this URL path component prefix that has a value of "{global-app-id}" that does not correspond to a piece of active and remote-enabled interactive content then it MUST return a 404 error.

The server MUST respond to any other HTTP request to a resource of this form by passing the following components of the request to the interactive content associated with the `global-app-id` in the URL path component of the request:

- The HTTP verb.
- The relative URI generated by stripping the "uc/apps/{global-app-id}/ext" prefix from the URL path component of the request. For example, a request to "uc/apps/example.com:foo/ext/bar" would correspond to a relative URI of "bar". A request to "uc/apps/example.com:foo/ext" would correspond to a subpath of the empty string: "".

[5] See section 4.44 for the definitions of "active" and "remote-enabled".

- Any query string from the request. For example, a request to "uc/apps/example.com:foo/ext/bar?baz=flange&qux=flob" would correspond to a query string of "baz=flange&qux=flob".
- The headers from the HTTP request, except any `X-UCClientAuthorisation` and `X-UCRestriction-Credentials` headers, which MUST NOT be passed to the interactive content.
- A boolean value indicating whether or not authentication (with a valid `client-id` and LSGS) was used in the request.
- The body of the HTTP request, if any.

If the client request includes a "`method_`" query parameter as described in section 4.1 then this parameter MUST be removed from the query string passed to the interactive content, and the HTTP verb passed to the interactive content MUST be that specified in the query parameter.

The server MUST implement a mechanism for the interactive content to provide an HTTP response to client requests, including an HTTP response code, headers and a response body. The server MUST NOT permit interactive content to return a 401 response code to the client. The server MUST NOT allow responses to clients to include a "WWW-Authenticate" header generated by the interactive content.

If the server is unable to obtain a response to a client request from the interactive content because the interactive content is no longer active, it MUST return a 404 error.

If the server is unable to obtain a response to a client request from the interactive content for some other reason, perhaps because the interactive content has crashed, it MUST return a 500 error.

The server MUST implement a mechanism for the interactive content to indicate that resources in its resource hierarchy have undergone a notifiable change, and MUST trigger notifiable changes in these resources when this is indicated. The server MUST NOT allow an interactive content to notify clients of changes to resources other than those in the "uc/apps/{global-app-id}/ext" resource hierarchy that is associated with that application.

4.47 The Minimal Server Implementation

This section is purely informative.

One of the objectives in designing this specification has been to ensure that the API is as modular as possible, and hence implementable by a wide range of devices. A consequence of this is that the minimal server implementation does not have to implement very many technologies in order to comply.

The simplest implementation which fully complies with this specification must contain an HTTP 1.1 server operating on port 48875 (see section 3.1), and must also be capable of advertising a DNS-SD [11] record using multicast DNS [12] (see section 3.2.2), a very simple standard using only a small subset of the full DNS specification. The implementation's HTTP server is required to respond to HTTP GET requests to the resource "uc" with a representation (which may be a static XML file) describing the server, and is required to respond to properly authenticated GET requests to the "uc/security" resource with a 204 response code and no message body (see section 4.3 and section 4.4). It must also respond correctly to an OPTIONS request made to either of these resources with the correct headers for a CORS [9] preflight request (see section 3.1), and to a GET request for "/crossdomain.xml", in response to which the server is required to respond with a message body containing a valid cross-domain policy file [10] (see also section 3.1), which may also be a static XML file.

The box on which this minimal server is running is also required to be able to display a pairing code to the user. If the security scheme is not implemented then the code on this screen does not need to change except when the box's IP-address changes. See section 3.2 for details.

This minimal server will be of no practical use to a client device since it does not implement any functionality through its resources, but by adding even one of the optional elements of this specification to the server a useful remote control or remote interface capability can be provided.

5 Client Implementation Considerations

This section is normative for client implementations. It contains the RECOMMENDED and REQUIRED behaviours for Universal Control clients. In general a client implementation MAY assume that servers will implement all of the REQUIRED server behaviour described in the rest of this document. A client SHOULD NOT assume that the server will implement RECOMMENDED and OPTIONAL behaviours described elsewhere in this document, and SHOULD check that particular OPTIONAL behaviours have been implemented before making use of them.

5.1 Client Identification

Each UC client MUST retain two pieces of information for use by servers in identifying it: a human-readable client name and a globally-unique client ID.

The client name MUST conform to the following requirements (expressed here in ABNF):

```
client-name = 1*( unreserved / pct-encoded )
```

Since percent-encoding is permissible the client name MAY be formed from any human-readable string, and SHOULD be chosen to make it as easy as possible for the user to identify the client in question from a list of all the UC clients that are authorised to connect to it. It SHOULD be un-percent-encoded before being presented to the client. The client name SHOULD identify both the specific device on which the client software is running as well as the client software itself if the device in question is capable of running more than one kind of UC client application. For example, "Bob's TV Client running on Alice's iPhone" contains all of these advisory features: the name of the software application ("Bob's TV Client") and the specific device on which the client is running ("Alice's iPhone") rather than just the name of the client platform (eg "Apple iPhone"). The client name SHOULD NOT include a representation of the client ID. The client name MUST be a between 0 and 63 characters long.

The client ID MUST be a string representation of an 128-bit integer in the format described for UUID's in RFC 4122[14]. Two UC client instances MUST NOT have the same client ID, regardless of whether or not they are running on different devices or at different times. The client ID MUST be generated when the client is first run, before any server connections are attempted. Once generated, it MUST be stored persistently, and always used to identify that particular client software on that particular device. It is RECOMMENDED that the client ID be a UUID as specified in RFC 4122[14], and the guarantees of uniqueness contained therein are considered sufficient.

The usage of these identifiers is described in sections 5.5,

5.2 Server Discovery

A UC client MAY discover the existence of the root URI of a UC server by any means that it is capable of employing.

A UC client MAY discover UC servers on the local network by using DNS Service Discovery carried via unicast or Multicast DNS to search for servers with the protocol name `_universalctrl._tcp` in the `.local` domain. The DNS record will contain a name for the server and its `server-id` in the `server_name` and `server_id` fields, and will contain the path, ip-address, and port-number of the server as normal. The `server-id` will be a 128-bit number in the standard string form described in RFC 4122 [14].

A UC client MUST implement a mechanism for a user to enter a "pairing code" (see section 3.2.2) as a string consisting only of the characters listed in Appendix B. A pairing code which

contains other characters MUST NOT be processed by the client. A valid pairing code is used in two different way. First it should be interpreted as a base-32 representation of an unsigned integer; The value of the IP address and port number can be decoded from this integer code using the algorithm given in Appendix A. Secondly it can be used to decode the SSS which can be used to pair with servers which implement the security scheme. The client behaviour when given a malformed code is up to the client implementer.

Once the client has decoded the information from a well-formed pairing code it MUST construct the URI of the "uc" resource for that server as `http://{ip address}:{port number}/uc`, where `{ip address}` and `{port number}` are the IP address and port number decoded from the code. The client MUST also decode the SSS if the security scheme is to be used (see Section 3.3). The pairing mechanism for a secure server is fully described in Section 5.5.

The client MAY choose a server to connect to from the list of discovered servers by any means. If providing the user with a list of servers to choose between then the client SHOULD identify each server discovered using the server's human-readable name if that is known. The client MAY use a stored name for a server it has already connected to. The client SHOULD replace any such stored name with a more up-to-date name if it has one. Any server with no known name which was discovered via a user-entered pairing-code MAY be referred to as "User Specified Server" or some description conveying similar information until a name for it is obtained.

A client MAY retain information about a server between sessions so that it can automatically connect to the same server(s) again when restarted. If it does so then it MAY store the IP address and `server-id` of servers between sessions and SHOULD take steps when reconnecting to ensure that the IP address has not been reassigned to a different server, for example by inspecting the `server-id` returned by the server to an initial request to the server's "uc" resource (see 5.4 below).

5.3 Client Requests and Responses

Upon making a request to a server the client SHOULD check the HTTP response code it receives. If the response code is 404 then the server does not implement the resource to which the response was made and the client SHOULD NOT make any further requests to that resource. If the response code is 405, then the resource does not respond to the verb used in the client request, and the client SHOULD NOT make further requests to that resource with that verb. If the response code is 204 then the request was successful but the server has not returned a representation. If the response code is 200 then the client MAY interpret the body of the response as an XML file conforming to the Universal Control XML schema (see Appendix D). Client behaviour when receiving a body which does not conform to the schema is left up to the individual implementation. If the code is 401 then the client has not correctly authenticated for the request it has made (see section 5.5), if the client is able to then it MAY repeat the request with correct authentication. Other codes have specific meanings depending upon the resource in question as explained in section 4.

If the response code to any request is in the range 500-599 (other than 504) then the client SHOULD NOT make any assumptions about whether the box has changed its internal state as a result of the request.

The client MAY assume that the server will not change any of the resource's state information in response to a GET, HEAD, or OPTIONS request which did not return a response in the range 500-599. The client MAY assume that the server will not change any of the resource's state information in response to a PUT, POST, or DELETE request which did not return a response in the ranges 200-299 or 500-599.

A client with no access to the HTTP status codes returned in response to its requests should still be able to act as a fully-functional UC client: if a request is successful, an XML document with a `response` root element will be returned, or a response with no body will be returned. If a request results in an error, an XML document with an `error` root element will be returned, with the HTTP status code duplicated in that document as the `code` attribute of the `error` element.

If a client platform does not permit HTTP requests with the full range of HTTP verbs used by

the Universal Control API ("GET", "POST", "PUT" and "DELETE"), the verb may be overridden by means of a query parameter `method_`. For example, to make a DELETE request to the resource `http://stb.example:48875/uc/foo/bar` from a client running on a platform that does not permit DELETE requests, a POST or GET request to `http://stb.example:48875/uc/foo/bar?method_=DELETE` could be made instead. It is RECOMMENDED that when the underlying HTTP verb is being overridden in this fashion that POST be used as the underlying verb. See section 4.1 for more information regarding how servers handle this query parameter.

All permitted HTTP verbs on resources defined in the UC API (with the exception of POST requests to "uc/remote") are guaranteed to be "idempotent" in compliant server implementations. This means that multiple consecutive identical client requests with the same HTTP verb, headers and message body will have the same result: they will return the same values, and any consequent changes to the server state will only be made once no matter how many times they are requested. This gives clients a clear correct course of action to take if a request fails: they MAY simply repeat the request. This guarantee can not be made for resources added to the API by interactive content as part of the API extension mechanism (see section 4.44): the idempotency or otherwise of these resources is determined by the interactive content developer. See section 5.8 for details of how client developers should deal with the non-idempotency of POST requests to "uc/remote".

The following restrictions on XML generated by the client are RECOMMENDED in order to ensure that the XML sent over the network is small, reducing bandwidth, CPU and power utilisation. The client SHOULD NOT produce XML with any non-required white-space characters. The client SHOULD NOT include namespace information in the XML it generates. The client SHOULD NOT include an `<?xml ... ?>` tag at the start of the XML it generates. When whitespace is required it SHOULD consist of a single character. Clients SHOULD NOT include elements and attributes in their resource representations that MUST be ignored by servers according to this document. To improve readability, example XML in this specification does not follow these guidelines.

When making requests with query-parameters clients MUST ensure that their values do not contain any characters which are not permitted in URI query-strings, and in addition do not contain any of the characters '&', ';', or '=' which are used by the server to separate keys from values and from each other. The technique of "percentage encoding" may be used to avoid this. In addition the client MUST ensure that the character '%' occurs only when followed by two hexadecimal digits (which MAY be upper or lower case). All locally scoped ids used in the API are already free from these non-permitted characters, and so may be passed without modification. The globally scoped identifiers (`global-content-id` and `global-series-id`) SHOULD be percent-encoded before inclusion in a URI query parameter, as should user entered free-text (such as the `text` parameter which can be passed to several of the "uc/search/{type}/{term}" resources). In all cases the code used to replace the character ' ' should be "%20" and not '+'.

When plain-text is included in a path segment (such as for requests to the "uc/search/text/{term}" resources) it MUST not contain any characters forbidden in a path-segment, and MUST also not contain any of the characters '?', ';', or '+'. The character '%' MUST only be present if followed by two hexadecimal digits. Percentage-encoding as described above is a legitimate way of ensuring these restrictions and all such strings will be un-percentage-encoded by the server before being used.

5.4 Initial Behaviour

When first connecting to a server, the client SHOULD make a GET request to the "uc" resource. If this request produces any response code except 200 then the attempt to connect has failed.

Upon receiving a response with a status code of 200, the client SHOULD then parse the response body to determine whether or not the server supports the UC security scheme (see section 3.2), to determine what version of the UC API the server implements, to acquire a server name and `server-id` for the server if it does not already have them, and to determine which optional resources

the server implements. The client MAY use the `logo-href` attribute of the `ucserver` element in the response (if present) to acquire an image to represent the server.

The version number reported by a compliant server will take the form of three parts separated by the character '.', each consisting only of decimal digits. If the first two parts of the API version number reported by the server do not match a version of the API supported by the client, the client SHOULD stop communicating with the server and report an incompatibility to the user.

If the server supports the UC security scheme, the client SHOULD prompt the user for a pairing code if one has not been retained for that server from a previous connection. The client SHOULD then immediately attempt to trigger an authenticated HTTP request by making a request to the server's "uc/security" resource. This will minimise the possibility of the pairing code expiring before its first use confirms the client as authorised to connect to the server.

5.5 Authentication

In order to be able to connect to a secured server the client MUST store a unique "client-id", which takes the form of an 128-bit number which is represented as a string according to the mechanism described for representing UUIDs as described in RFC4122[14]. When connecting to a secured server the client MUST follow the following process:

1. If the client already has access to a "Large Server Generated Secret" (LSGS) for this server then skip to step 7.

2. If the client already has access to a "Short Shared Secret" (SSS) for this server then skip to step 5.

3. The client prompts the user to enter a Pairing Code.

4. The client decodes the SSS from the entered pairing code.

5. The client makes a POST request to "uc/security" with the query parameters "client-id" and "client-name" providing the client's client-id and a human readable name for the client.

6. The client extracts the 256-character "key" from the server's response, interprets it as a lowercase hexadecimal representation of a sequence of 128 bytes, and then performs a bitwise xor of each byte with the 8-bit SSS. The resulting 128-byte sequence is the LSGS.

7. The client MUST make a GET request to "uc/security", and then repeats the request with the authentication headers described below. If the result is not a 204 response then the LSGS is invalid, and the client SHOULD inform the user that the credentials are invalid and return to step 3. Otherwise continue.

8. The client MUST store the LSGS persistently, future requests to this server SHOULD provide authentication headers as described below.

The mechanism for providing Authentication is described from the server's perspective in Section 3.3.3. From the client's perspective it is as follows:

- Any request made to a server resource may result in a 402 error with an `X-UCClientAuthenticate` header. Upon receiving such a response the client SHOULD parse it and extract the nonce and iteration value. Each nonce received this way SHOULD be stored semi-persistently by the client, and an associated unsigned integer nonce-count should be kept (which starts at 0). A request made with a previously valid, but no longer valid, nonce (see below) may indicate that this is so in this header.

- Any request made to a server resource SHOULD have a `X-UCClientAuthorisation` header, which MUST contain the nonce and iteration value obtained from the challenge, the URI of the request, the nonce-count, the client-id of the client and two pieces of information calculated by the client, the cnonce and the digest (see below). Each time such a request is made with a particular nonce the nonce-count for that nonce MUST be incremented by the client before the request is made (and the incremented value MUST be used).

The `cnonce` is a string of 40 lowercase hexadecimal digits which may be generated by the client in any way the client implementer wishes, and SHOULD be relatively non-predictable. It MAY change with each request made, or stay the same; frequent unpredictable changes provide better security.

The `digest` is calculated as described in Section 3.3.4 using the LSGS the client has stored, and the method, URI, nonce, message-body, nonce-count, and cnonce being used for the request. Libraries for the algorithm PBKDF2-HMAC-SHA1 are available for all major client platforms, and if a client developer wishes to write their own then the algorithm is described in detail in Appendix C.

5.6 Listening for Events

This specification defines a "notification mechanism" whereby servers can notify clients when certain key resources have changed. See section 4.5 for information about how servers implement this mechanism. If client implementors wish to update their user interfaces or perform other actions as soon as possible after changes to the server's state, it is RECOMMENDED that they use this mechanism if the server implements it. Clients MAY repeatedly poll server resources for changes instead, but this is likely to substantially increase the utilisation of CPU time, network bandwidth and battery life compared to using the notification mechanism, under almost all circumstances.

The client MAY listen for event notifications from the server if the server implements the optional "uc/events" resource. To do this the client MUST store a `notification-id` string between requests to this resource that identifies the last set of event notifications that the client received from the server. The client SHOULD NOT attempt to parse this string, or make any assumptions about the way in which it is generated.

Each time a client begins listening for events it does so by making a GET request to the "uc/events" resource with a single query parameter `since` set to the `notification-id` string returned by the server as the value of the `notification-id` attribute of the `events` element in the previous response to a client request to this resource. If the client does not have a cached `notification-id` string, it may obtain one by making a GET request to the "uc/events" resource with no query parameters. The response to such a request will be a representation of the "uc/events" resource (see section 4.5) containing a valid `notification-id`, but no event notifications. In subsequent requests, the client will never be informed of events that took place prior to the point in time when it obtained a `notification-id` from the server.

The client SHOULD NOT make multiple simultaneous requests to the "uc/events" resource.

When a request of this form returns with a 200 response the response body will contain a representation which can be parsed to acquire a list of URIs of resources which have undergone notifiable changes since the given time. The `notification-id` attribute of the `response` element in the representation will contain an updated `notification-id`, and the client SHOULD replace its stored `notification-id` value with that one. The client MAY make a new request to the "uc/events" resource immediately, or at any other time. If a response from the server is malformed in any way then the client SHOULD ignore it completely and MAY simply repeat the request.

If the response indicates that the "uc/power" resource has changed state, the client SHOULD assume that an indeterminate number of notifiable changes were lost at the point when the power state changed.

5.7 Clock Synchronisation

Implementing this section is OPTIONAL, but becomes RECOMMENDED if the client wishes to make use of time-related information about individual pieces of content, such as when they start and end. The rest of this section assumes that this is being implemented.

The client SHOULD maintain a clock offset value for each server – an offset which can be added to the current time according to the client's clock to give an approximation to the current time according to the server clock. A client MAY at any time attempt to synchronise it's clock with that of the server. A client SHOULD NOT do so more often than is necessary, but actual frequency is left up to the implementation (and will vary depending upon the purpose to which the synchronised timing data is being put). A client SHOULD do this at least once before attempting to use any of the "uc/search" sub-resources to obtain metadata. The client SHOULD perform the following steps in order:

1. The client SHOULD store the current date and time according to its own clock as s_1.
2. The client SHOULD make a GET request to the "uc/time" resource.
3. The client SHOULD store the current date and time according to its own clock as soon as it receives a response from the server as r_2.
4. If the response code is not 200 or there is an error parsing the representation returned then the client SHOULD abort the process.
5. The client SHOULD store the value of the `replytime` attribute of the `time` element as s_2 and the value of the `rcvdtime` attribute of the `time` element as r_1.
6. The client SHOULD calculate a new approximation for the clock offset as

$$\frac{(r_2 - s_2) - (r_1 - s_1)}{2}.$$

The client MAY combine this new estimate of the true clock offset with one or more stored offset values in any way in order to obtain a new approximation. For example, a mean of several values may be taken to reduce the effects of variable latency across the network.

5.8 Simple and Accessible Remote Control Behaviour

If a server implements the "uc/remote" resource (see section 4.13), a client MAY simulate a simple user input device such as an infra-red remote or a keyboard, by making POST requests to that resource with a single query parameter "button" whose value is a valid button code for one of the profiles which the server supports (see section 7). The client MAY determine which profiles the server supports by making a GET request to the same resource with no query parameters.

Since the "uc/remote" resource is not idempotent, the state of the box cannot be predicted if a network error interrupts a POST request to this resource, or a timeout occurs. To minimise the possibility that network errors may cause unintended side-effects, it is RECOMMENDED that clients make only one POST request to this resource at a time. If the client buffers keypresses made by the user in order to feed them to the server one at a time, the client SHOULD clear this buffer whenever a network error is detected. It is further RECOMMENDED that clients only make POST requests to this resource in direct response to requests from the user to simulate individual button presses. In this case, no further action (other than clearing any buffered button presses) is required by the client in the event of a request to this resource failing.

The client MAY receive textual feedback from the server regarding the current state of the box's built-in interface by making GET requests to the "uc/feedback" resource. This textual

feedback can be presented to the user in an accessible manner, enabling users who would not otherwise be able to to interact with the box's built-in interface.

Since the box's built-in interface is unlikely to have been designed with the accessibility needs of the individual user in mind, this kind of remote control mechanism should be regarded as a secondary method of providing accessible interfaces: a fall-back option for box-specific functionality. It is suggested that client developers wishing to produce accessible remote interface clients should instead make use of the EPG data, Output Control, and Seeking features of the Universal Control API (see section 5.9, section 5.10, and section 5.11) controlled from a custom interface on the client device, as this is likely to result in a better experience for users with (and without) accessibility requirements.

5.9 Identifying and Changing the Media Presented to the User

A client MAY determine the content that is currently being presented on an output, and what the current output settings are (such as audio volume, video aspect ratio, etc...) by making a GET request to the "uc/outputs/{output_id}" resource (see section 4.10), provided that the server implements the "uc/outputs" optional resource, and may alter what the output settings are by making a PUT request to the associated "uc/outputs/{output_id}/settings" resource, again assuming that the server permits this. A list of valid `output-id`s can be obtained by making a GET request to the resource "uc/outputs". The `output-id` "main" is always valid.

If the server implements the optional "uc/sources" and "uc/search/sources" or "uc/search/categories" resources (see sections 4.17 and 4.19 respectively), then the client MAY also alter what content (and which media components of the AV content) is being presented on an output by making a POST request to the "uc/outputs/{output_id}" resource. A list of valid sources can be obtained by making a GET request to the "uc/source-lists/{list_id}" resource for some valid `list-id`. A list of valid `list-id`s can be acquired by making a GET request to the "uc/source-lists" resource (see section 4.15). The `list-id` "uc_default" is always valid. If the source selected has the "linear" property then specifying a `source-id` is sufficient when making such a request, provided that only AV content is desired, when it does not, or when interactive content is desired, the client SHOULD discover what content is available from the source and provide the content-id for an appropriate piece of content as well as the source-id.

Details of the content available for a given source can be obtained by querying the "uc/search/sources/{source-id}" resource (see section 4.22), passing the `source-id` of the source for which content is desired. The default number of pieces of content returned by this resource is one (or none), unless the optional `results` parameter is specified. Many other parameters may be passed to the "uc/search" sub-resources.

The "uc/search/outputs/{output-id}" resource may also be useful: it will return information about the content which that output is currently presenting (if any), and the content which it is due to present (if any). One advantage of making this query is that a query to the "uc/search/outputs/{id}" resource will return information about the content currently being presented by the output even if that content is no longer available from the source (for example, if the user has paused live television for longer than the current piece of content's remaining length). Another advantage is that the content being presented on an output can be determined without having to query that output's associated resource to get the `source-id` and `content-id` of the content which it is currently presenting, and then make a second request to the "uc/search/sources/{sid}" resource for details of that content.

Each piece of AV content is comprised of one or more "media components" (see section 2.4), such as video, audio and subtitle tracks. If the user wishes to override the set of components selected automatically by the server or box, clients can request this by adding or modifying `component-override` elements in the representation they POST to a "uc/outputs/{output_id}" resource. See section 4.19 for information about how clients can determine what components a

piece of content has, and which components the server will select by default. Note that for some sources, details of a piece of content's media components may not be known by the server until that playback of that content starts. See section 4.8 for information about how the server will interpret `component-override` elements in a client POST request to "uc/outputs/{output_id}".

5.10 Seeking in a Stream

The current play position in a stream being presented on an output MAY be determined by making a GET request to the "uc/outputs/{output_id}/playhead" resource (see section 4.12), provided that the server implements that optional resource. The play position MAY be changed by making a PUT request to the same resource. It is RECOMMENDED that when making a change to the play position, only one `aposition` or `rposition` element is included in the request, and not one of each. If both are included, the server will ignore the one that appears first in the request.

A piece of AV content newly presented on an output is not guaranteed to start at any specific point. If a specific start point is desired, clients MAY request one by adding a `seek` query parameter to the request to change the AV content on an output. Using this query parameter as a way to seek within the current programme is NOT RECOMMENDED: using the appropriate "uc/outputs/{output_id}/playhead" resource (if available) will be more flexible and reliable. If this resource is not implemented by the server, the client SHOULD assume that the server does not permit seeking within the current piece of AV content from Universal Control clients by any means.

5.11 Content and Schedule Metadata (EPGs)

A client MAY obtain metadata about sources and content by making GET requests to the "uc/source-lists/{id}" (section 4.16) and "uc/search/sources/{sources-ids}" (section 4.22) resources, provided those resources are implemented. This metadata includes information about the content itself, as well as information regarding when the content will be broadcast or otherwise made available, if appropriate. Clients can use this information for many purposes, including the generation of user interfaces that offer similar functionality to the electronic programme guides (EPGs) conventionally included in the built-in interfaces of set-top boxes.

The "uc/search/{type}/{term}" resources are also the means by which a client can determine what media components are associated with a piece of AV content – see section 5.9. It is RECOMMENDED that the client synchronise its clock to the server (see section 5.7) before making use of this data.

Each piece of content may also be associated with one or more "categories". Each category is identified by a `category-id`. `category-id`s for each category with which a piece of content is associated may be returned by the server in responses to queries to the "uc/search/{type}/{term}" resources, and especially the "uc/search/categories/{category-id}" resources. Servers receiving a request for content that fall into a particular category will automatically include the content that fall into all the categories that are children, grandchildren etc of that category.

To obtain a list of the server's complete category hierarchy and to obtain human-readable names for the categories (and, potentially, the URLs of images that can represent them), clients can make a request to the "uc/categories" resource (see section 4.36). When presenting categories to the user, client developers should bear in mind that categories are designed to make sense to the user in the context of their ancestors in the tree. For example, the hierarchy depicted in Figure 1 (see section 2.3) has two "Factual" categories, one of which is a sub-category of "Children's". Two straightforward ways to avoid confusing the user are to present the entire hierarchy as a navigable user interface component (perhaps as a tree diagram), or to include the names of a category's ancestors when presenting its name to the user (for example, "Genres / Children's / Factual").

The UC metadata model incorporates the concept of global identifiers for pieces of content, series, and interactive pieces of content that dynamically extend the UC API when they are ac-

tive. The "uc/search/global-content-id/{global-content-id}, "uc/search/global-series-id/{global-series-id}" and "uc/search/global-app-id/{global-app-id}" resources can be used to determine the content known to the box that matches a global identifier obtained by the client by some other means (from the website of a broadcaster, for example, or from a QR code). Note that the default behaviour of the "uc/search" resources is not to return information regarding pieces of content that are no longer available: this can be overridden by specifying a value for the "start" query parameter that is a suitable distance in the past.

5.12 Managing the Acquisition and Storage of Content

"Acquisitions" are pieces of content that are not currently available, but that the user wants to become so. See section 4.37 for a more detailed explanation of what acquisitions are, and what they may represent.

A client MAY obtain information regarding the acquisitions that are currently scheduled by making a GET request to the "uc/acquisitions" resource (also described in section 4.37, if that optional resource is implemented, and MAY request the acquisition of individual pieces of content or entire series specified either via `content-id`s or `global-content-id`s by making POST requests to the same resource. Acquisitions can be cancelled (if the server permits) by making DELETE requests to the relevant "uc/acquisitions/{acquisition_id}" resources. When making POST requests to this resource with the query parameter `global-content-id` the client SHOULD use "percent-encoding" to ensure that the passed in query parameter complies with the requirements:

```
global-content-id-query-parameter  = id-element
```

in particular the query parameter MUST NOT contain the characters '&', '=', '?', or ';', as these are used by the server to separate parts of the URI, and MUST NOT contain any characters not permitted in a URI query-string (such as ' ', or '/').

A box may have dedicated storage assets, such as an internal hard disk or Internet-based storage area. A client MAY receive information regarding the content that is currently stored by these assets by making GET requests to the "uc/storage" resource, if that optional resource is implemented by the server, and MAY request the permanent deletion of such content by making DELETE requests to the "uc/storage/{source_id}" resources. See section 4.39 for more information about this resource, and the storage assets it represents.

5.13 Coupled Resources

Some resources are "coupled" to one another, by which we mean that a change to one resource can result in a simultaneous change to another. For example, the "uc/outputs/{output_id}/playhead" resource is coupled to the "uc/outputs/{output_id} resource with the same {output_id}, since a change to the current source of video for the latter will virtually always result in a change to the playhead position represented by the former. In general, when coupled resources undergo simultaneous notifiable changes, clients SHOULD NOT make assumptions about the order in which multiple related notifications are sent or received, nor about the timeliness of such notifications, which may be delayed due to circumstances beyond the control of either the server or the client.

5.14 Client Implementation in Web Browsers

This section is purely informative.

The same-origin policy employed by most web browsers at time of writing prevents clients loaded from a web server from accessing a Universal Control server running on the local network. Several browsers have begun to support the Cross Origin Resource Sharing recommendation[9], which permits AJAX clients to be written, served from web-servers other than the UC server. However, at the time of writing, all implementations of this recommendation that we are aware

of are limited. Browser compliance with W3C recommendations on cross-origin resource access is expected to improve over time.

Cross-origin access to resources from Flash applications is permitted by the cross-domain policy file described in section 3.1. Support for policy files of this kind is present in the versions of Flash installed on the overwhelming majority of client devices.

It is not common for web applications to be capable of making Multicast DNS requests, and hence it is likely that such clients would have to make use of pairing codes to discover the server. (If the server implements the security scheme then a pairing code is required anyway.)

5.15 Changes to the Server's IP Address

This section is purely informative.

On a typical home network, IP Addresses are allocated by DHCP from a pool. Although DHCP servers and clients attempt to minimise the frequency with which a dynamically-allocated IP address changes, clients cannot assume that such changes never happen. If a server's IP address changes, the most likely result is that connections from clients that have cached the old address will stop working. A less likely result is that the server's old IP address is assigned to a different UC server. In this case, the `server-id` in the server's base resource will appear to change. In both cases, the same symptoms can be caused by the client device being connected to a different network than the desired server.

If a client is unable to connect to a UC server using an IP address and port with which connections were previously possible, the following sequence of steps MAY be taken.

1. The client MAY retry the connection using the same IP address and port,

2. If the client has access to information about the network it is connected to, it MAY attempt to determine if the network has changed. For example, it MAY determine that the SSID of the wireless network to which it is connected has changed, or that the IP subnet or the default gateway have changed. If there is reason to believe that the client is no longer connected to the same network as the box then the client SHOULD prompt the user to remedy the situation before giving them the option to retry the connection.

3. The client MAY use DNS-SD carried over Multicast DNS to attempt to obtain a new IP address for the server, comparing the `server-id` of each server discovered by this means to the cached `server-id` for the sought server to identify the correct server on a network with multiple UC servers. If a new IP address is found, the client SHOULD retry the connection with the new IP address.

4. The client MAY prompt the user for a new pairing code. If the IP address in the pairing code is different to the client's cached value for that server, the client SHOULD assume that the IP address has changed, and retry the connection using the new IP address. If the IP address has not changed, the client MAY assume that the box was switched off, or in a standby mode, and retry the connection using the same IP address.

5. The client MAY inform the user that an undiagnosable network error has occurred, or that the original server cannot be found.

Whenever a client retrieves the base resource for a server, it MAY compare the `server-id` returned therein to the cached `server-id` for that server. If the two `server-id`s do not match, the client MAY take the following steps:

1. If the client has access to information about the network it is connected to, it MAY attempt to determine if the network has changed. For example, it MAY determine that the SSID of the wireless network to which it is connected has changed, or that the IP subnet or the default gateway have changed. If there is reason to believe that the client is no longer connected to

the same network as the box then the client SHOULD cease communicating with the new server and MAY make attempts to locate the old server on the new network as described in the following steps. Alternatively, the client MAY prompt the user to change back to the original network before giving them the option to retry the connection.

2. The client MAY use DNS-SD carried over Multicast DNS to attempt to obtain a new IP address for the original server, comparing each discovered server's `server-id` to the cached `server-id` for the original server. If a new IP address is found, the client SHOULD retry the connection with the new IP address.

3. The client MAY prompt the user for a new pairing code. If the IP address in the pairing code is different to the client's cached value for that server, the client SHOULD assume that the IP address has changed, and retry the connection using the new IP address. If the IP address has not changed, the client SHOULD assume that an undiagnosable network error has occurred.

4. The client MAY inform the user that the original server cannot be found.

6 Interactive Content Implementation Considerations

The "uc/apps" resource and its sub-resources define a means by which extensions to this API can be implemented at runtime by interactive content running on the same hardware as a UC server. It is the responsibility of interactive content authors to design these extensions. This section is intended to provide some basic guidance on the subject.

The "uc/apps" mechanism has been deliberately designed to be as flexible as possible. The mechanism allows for the API extension provided by the interactive content to be either RESTful or RPC-based. This specification mandates (see section 4.45) that virtually all parts of the HTTP request from the client be passed through to the interactive content. However, the interactive content is not required to use all parts of the request in the API extension it implements. To give a simple example, an API extension could be implemented in which the path, query string and headers in the client's HTTP request were all ignored by the interactive content, creating an API extension that is simply a mechanism for exchanging arbitrary strings of text (perhaps in XML format, perhaps not) between client and server.

Despite this extreme degree of flexibility, there are some restrictions on the kinds of API extension which it is advisable to implement:

- Web browsers traditionally limit the number of simultaneous connections that they can make, often to two per server. Since in many cases clients will be reserving one of these two connections for the "uc/events" resource (see sections 4.5 and 5.6), interactive content SHOULD NOT keep connections to their resources open for any longer than is necessary. In particular, if the information requested by a client is not available immediately, an asynchronous mechanism SHOULD be implemented in the API extension to prevent the client from having to wait for a response. For example, the interactive content could respond immediately without providing the information, thus closing the connection, and then use the "uc/events" notification mechanism to inform clients when the information has become available.

- An underlying principle of the UC API is that communication is initiated by clients, which can choose either to poll for updates or to use the "uc/events" notification mechanism (if implemented) to discover when server state changes. If interactive content developers wish for their content to initiate communication with a client, they should do so by using the "uc/events" mechanism to instruct the client to fetch a new copy of a resource that contains the information the server wishes to send.

- If interactive content authors want to communicate with some clients or users but not others (ie they wish to implement sessions, or client or user authentication), they must implement independent support for this as part of their API extension. There are a number of ways to accomplish this, including the generation of session IDs or persistent client IDs that clients then include in the URLs, headers or bodies of their requests. If the server requires clients to authenticate, then a boolean value is provided to the interactive content that indicates whether or not a request originated from a client authorised by a user of the box to connect to its UC server, but this is unlikely to be sufficient for all applications.

- The representation format (XML, JSON, CSV, plain text) used by API extensions is not specified. However, it is very likely that clients will have access to XML parsing libraries, since the rest of the UC API uses this standard. interactive content authors should bear in mind that the resource overhead associated with supporting an additional representation format may be significant on resource-limited platforms such as mobile phones.

- The XML schema used in the UC API has been designed to be simple to process using event-driven "SAX" XML parsers, which are more common than tree-based "DOM" parsers on resource-limited platforms. If it is desired that API extensions follow the same principles, then XML documents should be designed that minimise the use of "mixed content" (elements that contain a mixture of text and other elements) and nested elements. The use of attributes in preference to nested elements is preferred when that is possible.

- This specification document recommends, but does not specify or require, that server implementors provide a security mechanism that ensures that a piece of interactive content is from a trusted source. Depending on the nature of the platform on which a UC server is implemented, it may be possible for malicious applications to masquerade as trusted ones. interactive content authors should bear this in mind when considering what kinds of data it is appropriate to convey using this API extension mechanism.

- Similarly, interactive content authors should bear in mind the possibility that their API extensions may be accessed by untrusted clients. This specification document does not specify or require a security mechanism that allows interactive content to identify the source of a client application reliably, and implementing such a mechanism, if required, is left to the designer of the API extension. interactive content MUST NOT use the HTTP authentication mechanism as specified in RFC 2616 [8] (namely, the use of "WWW-Authenticate" and "Authorization" headers and the 401 response code), but MAY use a similar mechanism using different headers and an alternative response code.

- The HTTP response codes 404, 410 and 500 can be generated by the UC server directly in response to client requests in various situations (see section 4.46). If the interactive content author wants to return these codes to a client, he or she should consider whether or not it is necessary to implement some way to distinguish server-originated responses with these codes from application-originated ones.

In addition, designers of API extensions are advised to read the Universal Control Overview document [1], to gain a fuller understanding of the design process behind the rest of the UC API, and the decisions made.

In section 2.2.2, the `global-app-id` is introduced. Clients use this ID to determine whether or not a given piece of interactive content extends the UC API in a way that they can communicate with. For this reason, every distinct piece of interactive content that is made available on boxes that implement the UC API and that extends the UC API using the API extension mechanism MUST have a *globally unique* `global-app-id` formatted according to the requirements in the aforementioned section. A piece of interactive content that does not make use of the UC API extension mechanism MAY have an `global-app-id` associated with it, but whether or not a

given server platform makes use of it is implementation-specific. This specification mandates (see section 2.2.2) that when a piece of content implements the extension mechanism, servers MUST obtain a `global-app-id` directly from the content itself, or from the content's metadata, but the details of this process are specific to a given server implementation, and not described here.

It is anticipated that many interactive content developers will be providing both the client and the server implementations for a given multi-device user experience implemented via the UC API extension mechanism, in which case each individual experience is likely to be given a unique `global-app-id`. More complex use cases are also possible however, including:

- An `global-app-id` could contain a version number, allowing clients to be written that can communicate with multiple versions of a piece of interactive content.

- A common series of characters in two or more `global-app-id`s could be used to signal that multiple pieces of interactive content implement the same API, or part of an API.

- An `global-app-id` could contain several such "API identifiers", allowing multiple applications to implement multiple APIs (either standardised within the organisation that produced the content, or subject to wider standardisation).

7 Button Codes

This section describes the model used for identifying input-device key-codes as used by the server's "uc/remote" resource.

Each keycode represents a single button or key on an input device. Different input devices support different key-codes. The server MAY simulate the inputs from a particular input device, and when doing so SHOULD accept requests to perform key-presses using key-codes associated to that device.

The various forms of input device which the server MAY emulate are identified by "profiles", each of which is identified by a profile-id which is described in ABNF below:

```
profile-id            = [ domain ] ":" profile-specific-part

profile-specific-part = id-element
```

where `domain` SHOULD be included in all profiles except for the reserved ones described below. Where included the `domain` component SHOULD be a fully-qualified domain-name registered to the profile owner in the global DNS, eg "bbc.co.uk". As an example, the profile for the dbook remote control profile could be "dtg.org.uk:dbook_remote".

Each key in a profile MUST be associated with at least one keycode (it is permissible but NOT RECOMMENDED for more than one code in the same profile to identify the same key), which is as specified below:

```
keycode           = ( profile-id / ":" ) ":" key-specific-part
key-specific-part = id-element
```

where the `profile-id` MUST be the profile-id of the profile in which the code is located, and the alternative form without a profile-id MUST NOT be used except for the reserved key-codes listed below.

The `profile-ids` `:uk_keyboard` and `mheg5bp` (see Appendices E and F respectively) are reserved.

7.1 Common Remote Control Button Codes

Any profile for simulating conventional remote controls and similar devices which has the following types of button SHOULD identify them with the following key-codes:

- The first power button on a remote control SHOULD be given the code `::POWER`.
- The first set of numeric keys on a device SHOULD be identified by the codes `::0` to `::9`.
- Any input device with coloured red, green, yellow, and blue buttons for interactive application control SHOULD identify them with the codes `::RED`, `::GREEN`, `::YELLOW`, and `::BLUE`.
- A button designed to bring up an EPG SHOULD be designated by `::EPG`.
- A button designed to bring up the main menu SHOULD be designated by `::MENU`.
- A button designed to activate teletext SHOULD be designated by `::TEXT`.
- The first set of cursor keys on a device SHOULD be designated by `::CURSOR_UP`, `::CURSOR_DOWN`, `::CURSOR_LEFT`, and `::CURSOR_RIGHT`.
- A button designed to make a selection which has been indicated using cursor keys SHOULD be designated `::OK`.
- A button designed to take the user back to the previous screen SHOULD be designated `::BACK`.
- Buttons for raising and lowering volume SHOULD be designated `::VOLUME_UP` and `::VOLUME_DOWN`.
- Buttons for incrementing and decrementing the channel SHOULD be designated `::CHANNEL_UP` and `::CHANNEL_DOWN`.
- A button designed to mute the audio output SHOULD be designated `::MUTE`.
- PVR control buttons SHOULD be designated `::PLAY`, `::PAUSE`, `::PLAY_PAUSE`, `::STOP`, `::REWIND`, `::FASTFORWARD`, `::SKIPBACK`, `::SKIPFORWARD` and `::RECORD`.
- Any button designed to switch audio descriptions on and off SHOULD be designated `::AUDIO`.
- Any button designed to switch subtitles on and off SHOULD be designated `::SUBTITLE`.

Other buttons MAY be identified as the designer of the profile desires within the above constraints. Profiles for user input devices that are dissimilar to conventional remote controls (for example, computer keyboards) MAY use some or all of the common remote control button codes defined above, if they wish servers to make no distinction between button presses on the user input device being profiled and conventional remote controls. An example of a button profile for keyboards can be found in Appendix E. This example does not incorporate any of the common remote control button codes defined above.

References

[1] J. Barrett and S. Jolly. The Universal Control Project. White Paper 193, BBC R&D, 2011.

[2] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, BCP, 1997.

[3] Ed. Klyne, G. and C. Newman. Date and Time on the Internet: Timestamps. RFC 3339, IETF, 2002.

[4] Ed. Crocker, D. and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234, IETF, 2008.

[5] T. Berners-Lee, R. Fielding, , and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, IETF, 2005.

[6] Digital Video Broadcasting (DVB); Uniform Resource Identifiers (URI) for DVB Systems. Technical Report A142, DVB, 2009. `http://www.dvb.org/technology/standards/a142_UniformResourceIdentifiers.pdf`.

[7] R. Moats. URN Syntax. RFC 2141, IETF, 1997.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, IETF, 1999.

[9] A. van Kesteren. Cross Origin Resource Sharing. Technical report, W3C, 2009. http://www.w3.org/TR/access-control/.

[10] Adobe. Cross-domain policy file specification. Technical report, 2010. http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html.

[11] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. Technical report, Apple Inc., 2008. http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt.

[12] B. Woodcock and B. Manning. Multicast Domain Name Service. Technical report, watersprings.org, 2000. http://www.watersprings.org/pub/id/draft-manning-dnsext-mdns-00.txt.

[13] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewert. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, IETF, 1999.

[14] P. Leach, M. Mealling, and R. Salz. A Universal Unique Identifier (UUID) URN Namespace. RFC 4122, IETF, 2005.

[15] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, IETF, 2000.

[16] Comet_(programming). Technical report, wikipedia.org, 2010. http://en.wikipedia.org/wiki/Comet_

[17] Picture-in-picture. Technical report, wikipedia.org, 2010. http://en.wikipedia.org/wiki/Picture-in-picture.

[18] W3C XML Schema Working Group. XML Schema 1.0. Technical report, W3C, 2004. `http://www.w3.org/XML/Schema.html#dev`.

[19] BSI. *BS 4822:1994: Keyboard allocation of graphic characters for data processing.* British Standards Institute, 1994.

[20] DirectFB community. `http://git.directfb.org/?p=core/DirectFB.git;a=blob;f=include/directfb%_keyboard.h`. Retrieved 3rd November 2010.

[21] ETSI. MHEG-5 Broadcast Profile. Technical Report ES202184 v2.1.1, ETSI, 2010. `http://www.etsi.org/deliver/etsi_es/202100_202199/202184/02.01.01_50/es_202184v020101m.pdf`.

Appendix A Generation and Parsing of Valid Pairing Codes

To generate the binary code, the box MUST have access to its own IP-address, and the port number upon which the HTTP server operates, if the security scheme is in operation then it MUST also generate a random 8-bit Simple Shared Secret ("SSS") before generating the code. The box MUST generate the binary code by a mechanism which produces a value decodable by the algorithm described in python-like pseudo-code below. Digits shown wrapped in `=` signs are literal binary digits with the LSB on the left, and the `input` command reads a specified number of bits from the bitstream starting with the LSB. Each successive read operation reads from where the previous one left off, and reading beyond the end of the specified bits always returns zeros:

```
signal = input 1 bit
if signal is =1=:
  The binary code is in an unknown format.
  Stop processing and report an error to the user.
else:
  Continue processing the binary code

signal = input 1 bit
if signal is =1=:
  SSS = input 8 bits
else:
  SSS is undefined: the server does not implement the security scheme.

signal = input 2 bits
if signal is =00=:
  A = 192
  B = 168
  signal = input 2 bits
  if signal is =00=:
    C = 0
  elif signal is =10=:
    C = 1
  elif signal is =01=:
    C = 2
  else:
    C = input 8 bits

  D = input 8 bits

elif signal is =10=:
  D = input 8 bits
  C = input 8 bits
  B = (input 4 bits)+16
  A = 172
elif signal is =01=:
  D = input 8 bits
  C = input 8 bits
  B = input 8 bits
  A = 10
else:
  D = input 8 bits
```

```
  C = input 8 bits
  B = input 8 bits
  A = input 8 bits

signal = input 1 bit
if signal is =1=:
  P = input 16 bits
else:
  P = 48875

let A.B.C.D denote the IP address
let P denote the port number
let CP denote the CP
```

One possible algorithm for generating such a code is specified in the same kind of pseudo-code below. The `output` commands append their parameter to the outputted code starting with the LSB.

```
let A.B.C.D denote the IP address
let P denote the port number

output =0=

if the server supports the security scheme:
  output =1=
  output SSS in 8 bits
else:
  output =0=

if IP is of form 192.168.*.*:
  output =00=

  if IP is of form 192.168.0.*:
    output =00=
  elif IP is of form 192.168.1.*:
    output =10=
  elif IP is of form 192.168.2.*:
    output =01=
  else:
    output =11=
    output C in 8 bits

  output D in 8 bits

elif IP is of form 172.16-31.*.*:
  output =10=
  output D in 8 bits
  output C in 8 bits
  output B-16 in 4 bits

elif IP is of form 10.*.*.*:
  output =01=
```

```
  output D in 8 bits
  output C in 8 bits
  output B in 8 bits

else:
  output =11=
  output D in 8 bits
  output C in 8 bits
  output B in 8 bits
  output A in 8 bits

if P != 48875:
  output =1=
  output P in 16 bits
```

As a non-normative note we suggest that some client devices will also function as telephones, and hence it may be preferable if the representation of the pairing code presented by the box is not a valid telephone number in the country in which the box is located. Including otherwise unnecessary leading `0`s may be a good way of ensuring that this is the case.

The length of this code will vary between 14 and 44 bits in length provided that the port-number is 48875. With a non-standard port-number the code can rise as high as 61 bits in length. The most common format for local networks, with security scheme turned on with generall be around 22 bits in length.

Appendix B Base-32 Encoding for Pairing Codes

The recommended encoding for presenting the pairing-code and/or other challenge keys to the user is via base-32 according to the following scheme:

value	encoding
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	G
17	H
18	J
19	K
20	M
21	N

22	P
23	Q
24	R
25	S
26	T
27	V
28	W
29	X
30	Y
31	Z

The choice of letters used in the above scheme has been made so as to avoid ambiguity from letters which closely resembled numbers (such as `I` and `O`).

Appendix C The PBKDF2-HMAC-SHA1 Algorithm

For convenience for developers implementing the algorithm themselves we explain here how to implement it using only standard logical operators and the well-known SHA-1 hashing function. Library functions for SHA-1 are available in an even wider variety of platforms.

1. If the literal string P is longer than 64 octets (which will not be the case when using the LSGS as described above) then set the value K to be the expansion of the value SHA1(P) to 64 octets by appending the raw octet [00] as necessary, otherwise set K to be the expansion of the string P to be 64 octets long by appending the raw octet [00] to it as necessary.

2. Set *opad* to be the literal [5c] repeated 64 times, and set K_1 as follows:

$$K_1 := K \oplus opad$$

3. Set *ipad* to be the literal [36] repeated 64 times, and set K_2 as follows:

$$K_2 := K \oplus ipad$$

4. Set U_1 as follows:

$$U_1 := \mathrm{SHA1}(K_1 \sqcup \mathrm{SHA1}(K_2 \sqcup S \sqcup [00000001]))$$

5. For each value $i = 2 \ldots c$ let:

$$U_i := \mathrm{SHA1}(K_1 \sqcup \mathrm{SHA1}(K_2 \sqcup U_{i-1}))$$

6. Calculate the digest as

$$\bigoplus_{i=1}^{c} U_i$$

where square brackets denote a literal hex representation of a series of octets, the symbl $\sqcup$ denotes concatenation of two octet streams, the symbol $\oplus$ denotes the bit-wise exclusive-or of two equal length octet streams, and the function SHA1 denotes the well-known SHA-1 hashing function which can be applied to a string of any length but always produces a 64-octet output.

Appendix D XML schema for the Universal Control server

Resource representations sent between server and client MUST conform to the following schema. Upon receiving a PUT or POST request with a non-zero-length body which does not conform to this schema the server MAY return a 400 error regardless of the resource to which the request was made. Compliance with this schema is a necessary but not sufficient step towards compliance with this specification; additional necessary restrictions are set out in the other parts of this document.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://rd.bbc.co.uk/UniversalControl"
       xmlns="http://rd.bbc.co.uk/UniversalControl"
       elementFormDefault="qualified">

  <!-- This type defines valid version numbers which can be considered
       to conform to this schema. This should be updated when the version
       number is changed. -->

  <xs:simpleType name="version_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="0\.6\.0"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="response">
    <xs:complexType>
      <xs:choice>
        <xs:element name="ucserver" type="server_t"    />
        <xs:element name="power" type="power_t"    />
        <xs:element name="time" type="time_t"   />
        <xs:element name="output" type="output_t" />
        <xs:element name="settings" type="settings_t" />
        <xs:element name="programme" type="programme_reference_t" />
        <xs:element name="outputs" type="outputs_t" />
        <xs:element name="events" type="events_t" />
        <xs:element name="source" type="source_t" />
        <xs:element name="sources" type="sources_t" />
        <xs:element name="source-lists" type="source-lists_t"/>
        <xs:element name="playhead" type="playhead_t" />
        <xs:element name="feedback" type="feedback_t" />
        <xs:sequence minOccurs="1">
          <xs:element name="results" type="results_t" />
        </xs:sequence>
        <xs:element name="categories" type="categories_t" />
        <xs:element name="acquisitions" type="acquisitions_t" />
        <xs:element name="content-acquisition" type="content-acquisition_t" />
        <xs:element name="series-acquisition" type="series-acquisition_t" />
        <xs:element name="storage" type="storage_t" />
        <xs:element name="stored-content" type="stored-content_t" />
        <xs:element name="remote" type="remote_t" />
        <xs:element name="credentials" type="credentials_t" />
```

```
        <xs:element name="apps" type="apps_t" />
        <xs:element name="app" type="app_t" />
      </xs:choice>

      <xs:attribute name="resource"
    type="rref_t"
    use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="error">
    <xs:complexType>
      <xs:attribute name="code"
            type="http_response_t"
            use="required" />
    </xs:complexType>
  </xs:element>

<!-- Type definitions -->

  <xs:complexType name="server_t">
    <xs:sequence>
      <xs:element name="resource"
                  type="option_t" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name"
          type="xs:string"
          use="required" />
    <xs:attribute name="server-id"
                  type="uuid_t"
          use="required" />
    <xs:attribute name="security-scheme"
                  type="xs:boolean"
          use="required"/>
    <xs:attribute name="version"
          type="version_t"
          use="required" />
    <xs:attribute name="logo-href"
          type="logo-href_t" />
  </xs:complexType>

  <xs:complexType name="option_t">
    <xs:attribute name="rref"
                  type="option_rref_t"/>
  </xs:complexType>

  <xs:complexType name="power_t">
    <xs:attribute name="state"
          type="powerstate_t"
          use="required" />
    <xs:attribute name="transitioning-to"
```

```
        type="powerstate_t" />
</xs:complexType>

<xs:complexType name="time_t">
  <xs:attribute name="rcvdtime"
        type="xs:dateTime"
        use="required" />
  <xs:attribute name="replytime"
        type="xs:dateTime"
        use="required" />
</xs:complexType>

<xs:complexType name="output_t">
  <xs:all>
    <xs:element name="settings"
        type="settings_t"
        minOccurs="1"
        maxOccurs="1" />
    <xs:element name="programme"
        type="programme_reference_t"
        minOccurs="0"
        maxOccurs="1"/>
    <xs:element name="app"
        type="app_reference_t"
        minOccurs="0"
        maxOccurs="1"/>
    <xs:element name="playback"
        type="playback_t"
        minOccurs="0"
        maxOccurs="1"/>
  </xs:all>
  <xs:attribute name="name"
        type="xs:string"
        use="required"/>
</xs:complexType>

<xs:complexType name="settings_t">
  <xs:attribute name="volume"
        type="volume_t"/>
  <xs:attribute name="mute"
        type="xs:boolean"/>
  <xs:attribute name="aspect"
        type="aspect_t"/>
</xs:complexType>

<xs:complexType name="programme_reference_t" >
  <xs:sequence>
    <xs:element name="component-override"
        type="component-reference_t"
        minOccurs="0"
        maxOccurs="unbounded" />
```

```
  </xs:sequence>
  <xs:attribute name="sid"
        type="sid_t"
        use="required"  />
  <xs:attribute name="cid"
        type="content-id-or-blank_t"
        use="required"  />
</xs:complexType>

<xs:complexType name="component-reference_t">
  <xs:attribute name="type"
        type="component_t"
        use="required"/>
  <xs:attribute name="name"
        type="xs:string"/>
  <xs:attribute name="mcid"
        type="mcid_t"
        use="required"/>
  <xs:attribute name="lang"
        type="xs:string" />
  <xs:attribute name="aspect"
        type="aspect_t"    />
  <xs:attribute name="intent"
        type="intent_t" />
  <xs:attribute name="vidformat"
        type="vidformat_t" />
  <xs:attribute name="audformat"
        type="audformat_t" />
  <xs:attribute name="colour"
        type="xs:boolean"/>
  <xs:attribute name="default"
        type="xs:boolean"
        default="false" />
</xs:complexType>

<xs:complexType name="app_reference_t" >
  <xs:sequence>
    <xs:element name="controls"
        type="controls_t"
        minOccurs="0"
        maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="sid"
        type="sid_t"
        use="required"  />
  <xs:attribute name="cid"
        type="content-id-or-blank_t"
        use="required"  />
</xs:complexType>

<xs:complexType name="controls_t">
```

```
  <xs:attribute name="profile"
        type="profile_t"
        use="required"/>
</xs:complexType>

<xs:complexType name="playback_t" >
  <xs:attribute name="speed"
        type="xs:decimal"
        use="required"/>
</xs:complexType>

<xs:complexType name="outputs_t">
  <xs:sequence>
    <xs:element name="output"
        type="output_list_element_t"
        minOccurs="0"
        maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="output_list_element_t">
  <xs:sequence>
    <xs:element name="output"
        type="output_list_element_t"
        minOccurs="0"
        maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="oid"
        type="outputid_t"
        use="required" />
  <xs:attribute name="name"
        type="xs:string"
        use="required"/>
</xs:complexType>

<xs:complexType name="events_t">
  <xs:sequence>
    <xs:element name="resource"
        minOccurs="0"
        maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="rref"
          type="rref_t"
          use="required" />
  </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="notification-id"
          type="notificationid_t"
          use="required" />
</xs:complexType>
```

```
<xs:complexType name="source_t">
  <xs:sequence>
    <xs:element name="link"
        type="link_t"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="sid"
        type="sid_t"
        use="required" />
  <xs:attribute name="name"
        type="xs:string"
        use="required"/>
  <xs:attribute name="sref"
        type="xs:anyURI" />
  <xs:attribute name="linear"
        type="xs:boolean"
        default="false"/>
  <xs:attribute name="live"
        type="xs:boolean"
        default="false"/>
  <xs:attribute name="follow-on"
        type="xs:boolean"
        default="false"/>
  <xs:attribute name="audio-only"
        type="xs:boolean"
        default="false"/>
  <xs:attribute name="owner"
        type="xs:string"/>
  <xs:attribute name="logo-href"
        type="logo-href_t"/>
  <xs:attribute name="owner-logo-href"
        type="logo-href_t"/>
  <xs:attribute name="lcn"
        type="lcn_t" />
  <xs:attribute name="default-content-id"
        type="content-id_t" />
</xs:complexType>

<xs:complexType name="sources_t">
  <xs:sequence>
    <xs:element name="source"
        type="source_t"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="source-lists_t">
  <xs:sequence>
```

```
    <xs:element name="list"
        minOccurs="0"
        maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="link"
          type="link_t"
          minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="list-id"
          type="lid_t"
          use="required" />
    <xs:attribute name="name"
          type="xs:string"
          use="required" />
    <xs:attribute name="description"
          type="xs:string" />
    <xs:attribute name="logo-href"
          type="logo-href_t" />
  </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="link_t">
  <xs:attribute name="href"
            type="xs:anyURI"
            use="required" />
  <xs:attribute name="description"
            type="xs:string"
            use="required" />
</xs:complexType>

<xs:complexType name="position_t">
  <xs:attribute name="position"
type="xs:decimal"/>
  <xs:attribute name="seek-start"
type="xs:decimal"/>
  <xs:attribute name="seek-end"
type="xs:decimal"/>
</xs:complexType>


<xs:complexType name="playhead_t">
  <xs:all minOccurs="0">
    <xs:element name="aposition" type="position_t"/>
    <xs:element name="rposition" type="position_t"/>
    <xs:element name="playback"
        type="playback_t"/>
  </xs:all>
```

```
  <xs:attribute name="timestamp"
        type="xs:dateTime"/>
  <xs:attribute name="length"
        type="xs:decimal"/>
</xs:complexType>

<xs:complexType name="feedback_t">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="time"
          type="xs:dateTime"
          use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="results_t">
  <xs:sequence>
    <xs:element name="content"
        type="content_t"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="sid"
        type="sid_t"
        use="required" />
  <xs:attribute name="offset"
        type="xs:integer"/>
  <xs:attribute name="total"
        type="xs:integer"/>
</xs:complexType>

<xs:complexType name="content_t">
  <xs:sequence>
    <xs:element name="synopsis"
        type="xs:string"
        minOccurs="0"
        maxOccurs="1" />
    <xs:element name="category"
        type="category-reference_t"
        minOccurs="0"
        maxOccurs="unbounded" />
    <xs:element name="media-component"
        type="component-reference_t"
        minOccurs="0"
        maxOccurs="unbounded" />
    <xs:element name="controls"
        type="controls_t"
        minOccurs="0"
        maxOccurs="unbounded" />
    <xs:element name="link"
```

```
        type="link_t"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="global-content-id"
        type="global-id_t" />
  <xs:attribute name="series-id"
        type="series-id_t" />
  <xs:attribute name="global-series-id"
        type="global-id_t" />
  <xs:attribute name="title"
        type="xs:string" />
  <xs:attribute name="cid"
        type="content-id_t"
        use="required"     />
  <xs:attribute name="cref"
        type="xs:anyURI"   />
  <xs:attribute name="start"
        type="xs:dateTime" />
  <xs:attribute name="presentable-from"
        type="xs:dateTime" />
  <xs:attribute name="presentable-until"
        type="xs:dateTime" />
  <xs:attribute name="acquirable-from"
        type="xs:dateTime" />
  <xs:attribute name="acquirable-until"
        type="xs:dateTime" />
  <xs:attribute name="duration"
        type="xs:decimal" />
  <xs:attribute name="logo-href"
        type="logo-href_t" />
  <xs:attribute name="presentation-count"
        type="xs:nonNegativeInteger"
        default="0" />
  <xs:attribute name="last-presented"
        type="xs:dateTime" />
  <xs:attribute name="last-position"
        type="xs:decimal"  />
  <xs:attribute name="interactive"
        type="xs:boolean" />
  <xs:attribute name="extended"
        type="xs:boolean" />
  <xs:attribute name="acquirable"
        type="xs:boolean" />
  <xs:attribute name="presentable"
        type="xs:boolean" />
</xs:complexType>

<xs:complexType name="category-reference_t">
  <xs:attribute name="category-id"
        type="categoryid_t"
```

```
        use="required"/>
</xs:complexType>

<xs:complexType name="categories_t">
  <xs:sequence>
    <xs:element name="category"
        type="category_t"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="category_t">
  <xs:sequence>
    <xs:element name="category"
        type="category_t"
        minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="category-id"
        type="categoryid_t"/>
  <xs:attribute name="name"
        type="xs:string"
        use="required"/>
  <xs:attribute name="logo-href"
        type="logo-href_t"/>
</xs:complexType>

<xs:complexType name="acquisitions_t">
  <xs:sequence>
    <xs:element name="series-acquisition"
        type="series-acquisition_t"
        minOccurs="0"
        maxOccurs="unbounded" />
    <xs:element name="content-acquisition"
        type="content-acquisition_t"
        minOccurs="0"
        maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="series-acquisition_t">
  <xs:attribute name="acquisition-id"
        type="aid_t"
        use="required" />
  <xs:attribute name="series-id"
        type="xs:anyURI"
        use="required" />
  <xs:attribute name="speculative"
        type="xs:boolean"
        default="false"/>
```

```
</xs:complexType>

<xs:complexType name="content-acquisition_t" >
  <xs:attribute name="acquisition-id"
        type="aid_t"
        use="required" />
  <xs:attribute name="global-content-id"
        type="global-id_t" />
  <xs:attribute name="series-id"
        type="xs:anyURI" />
  <xs:attribute name="series-linked"
        type="xs:boolean"
        default="false" />
  <xs:attribute name="sid"
                type="sid_t" />
  <xs:attribute name="cid"
        type="content-id_t"
        use="required" />
  <xs:attribute name="priority"
        type="xs:boolean"
        default="false" />
  <xs:attribute name="start"
        type="xs:dateTime" />
  <xs:attribute name="end"
        type="xs:dateTime" />
  <xs:attribute name="speculative"
        type="xs:boolean"
        default="false" />
  <xs:attribute name="active"
        type="xs:boolean"/>
</xs:complexType>

<xs:complexType name="storage_t">
  <xs:sequence>
    <xs:element name="stored-content"
        type="stored-content_t"
        minOccurs="0"
        maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="size"
        type="xs:long" />
  <xs:attribute name="free"
        type="xs:long" />
</xs:complexType>

<xs:complexType name="stored-content_t">
  <xs:attribute name="cid"
        type="content-id_t"
        use="required"/>
  <xs:attribute name="interactive"
        type="xs:boolean" />
```

```
  <xs:attribute name="sid"
        type="sid_t"/>
  <xs:attribute name="global-content-id"
        type="global-id_t" />
  <xs:attribute name="created-time"
        type="xs:dateTime" />
  <xs:attribute name="size"
        type="xs:nonNegativeInteger" />
</xs:complexType>

<xs:complexType name="remote_t">
  <xs:sequence>
    <xs:element name="controls"
        type="controls_t"
        minOccurs="0"
        maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="credentials_t">
  <xs:sequence>
    <xs:element name="client"
        minOccurs="0"
        maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="client-id"
          type="uuid_t"
          use="required"/>
    <xs:attribute name="name"
          type="xs:string"
          use="required" />
  </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="apps_t">
  <xs:sequence>
    <xs:element name="app"
          type="app_t"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="app_t">
  <xs:attribute name="global-app-id"
                type="appid_t"/>
    <xs:attribute name="sid"
                type="sid_t"
                use="required"/>
```

```
      <xs:attribute name="cid"
                    type="content-id_t"
                    use="required"/>
      <xs:attribute name="remote-enabled"
                    type="xs:boolean"
                    default="false" />
  </xs:complexType>

<!-- Simple Types -->

  <xs:simpleType name="volume_t">
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0" />
      <xs:maxInclusive value="1" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="aspect_t">
    <xs:restriction base="xs:string" >
      <xs:enumeration value="source" />
      <xs:enumeration value="4:3"    />
      <xs:enumeration value="14:9"   />
      <xs:enumeration value="16:10"  />
      <xs:enumeration value="16:9"   />
      <xs:enumeration value="21:9"   />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="component_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="video"       />
      <xs:enumeration value="audio"       />
      <xs:enumeration value="subtitles"   />
      <xs:enumeration value="interactive" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="intent_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="admix"  />
      <xs:enumeration value="hhsubs" />
      <xs:enumeration value="signed" />
      <xs:enumeration value="iimix"  />
      <xs:enumeration value="commentary"  />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="vidformat_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="SD"  />
      <xs:enumeration value="HD"  />
```

```
    <xs:enumeration value="S3D" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="audformat_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="mono"  />
    <xs:enumeration value="stereo"  />
    <xs:enumeration value="surround" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tagtype_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="main" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="appstate_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="active"      />
    <xs:enumeration value="inactive"    />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="global-id_t">
  <xs:restriction base="xs:anyURI"/>
</xs:simpleType>

<xs:simpleType name="appid_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-zA-Z0-9]+(_+[a-zA-Z0-9]+)*(\.[a-zA-Z0-9]+(_+[a-zA-Z0-9]+)*)*)?:([
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="powerstate_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="on"     />
    <xs:enumeration value="standby"/>
    <xs:enumeration value="off"    />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="outputid_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="sid_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>
```

```
<xs:simpleType name="sid-or-blank_t">
  <xs:restriction base="id-or-blank_t"/>
</xs:simpleType>

<xs:simpleType name="content-id_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="content-id-or-blank_t">
  <xs:restriction base="id-or-blank_t"/>
</xs:simpleType>

<xs:simpleType name="series-id_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="mcid_t">
  <xs:restriction base="id-or-blank_t"/>
</xs:simpleType>

<xs:simpleType name="aid_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="lid_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="notificationid_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="categoryid_t">
  <xs:restriction base="id_t"/>
</xs:simpleType>

<xs:simpleType name="id_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-zA-Z0-9\-\._~]|%[0-9A-Fa-f][0-9A-Fa-f])+"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="id-or-blank_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-zA-Z0-9\-\._~]|%[0-9A-Fa-f][0-9A-Fa-f])*"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="uuid_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9a-f]{8}\-[0-9a-f]{4}\-[0-9a-f]{4}\-[0-9a-f]{4}\-[0-9a-f]{12}"/>
```

```
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="rref_t">
    <xs:restriction base="xs:anyURI">
      <xs:pattern value="uc.*"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="option_rref_t">
    <xs:restriction base="rref_t">
      <xs:enumeration value="uc/time"/>
      <xs:enumeration value="uc/events"/>
      <xs:enumeration value="uc/power"/>
      <xs:enumeration value="uc/outputs"/>
      <xs:enumeration value="uc/sources"/>
      <xs:enumeration value="uc/source-lists"/>
      <xs:enumeration value="uc/remote"/>
      <xs:enumeration value="uc/feedback"/>
      <xs:enumeration value="uc/search"/>
      <xs:enumeration value="uc/acquisitions"/>
      <xs:enumeration value="uc/storage"/>
      <xs:enumeration value="uc/credentials"/>
      <xs:enumeration value="uc/apps"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="logo-href_t">
    <xs:restriction base="xs:anyURI">
      <xs:pattern value="(http://|htts://).*"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="profile_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-zA-Z0-9]+(_+[a-zA-Z0-9]+)*(\.[a-zA-Z0-9]+(_+[a-zA-Z0-9]+)*)*|):([
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="lcn_t">
    <xs:restriction base="xs:nonNegativeInteger"/>
  </xs:simpleType>

  <xs:simpleType name="http_response_t">
    <xs:restriction base="xs:positiveInteger">
      <xs:minInclusive value="100"/>
      <xs:maxInclusive value="599"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Appendix E The UK_KEYBOARD Button profile

The button profile `:uk_keyboard` is based on the buttons defined in BS 4822 [19] and the DirectFB key identifiers [20], and consists of the following button codes:

- `:uk_keyboard:0`
- `:uk_keyboard:1`
- `:uk_keyboard:2`
- `:uk_keyboard:3`
- `:uk_keyboard:4`
- `:uk_keyboard:5`
- `:uk_keyboard:6`
- `:uk_keyboard:7`
- `:uk_keyboard:8`
- `:uk_keyboard:9`
- `:uk_keyboard:AMPERSAND`
- `:uk_keyboard:APOSTROPHE`
- `:uk_keyboard:ASTERISK`
- `:uk_keyboard:AT`
- `:uk_keyboard:BACKSLASH`
- `:uk_keyboard:BACKSPACE`
- `:uk_keyboard:CAPITAL_A`
- `:uk_keyboard:CAPITAL_B`
- `:uk_keyboard:CAPITAL_C`
- `:uk_keyboard:CAPITAL_D`
- `:uk_keyboard:CAPITAL_E`
- `:uk_keyboard:CAPITAL_F`
- `:uk_keyboard:CAPITAL_G`
- `:uk_keyboard:CAPITAL_H`
- `:uk_keyboard:CAPITAL_I`
- `:uk_keyboard:CAPITAL_J`
- `:uk_keyboard:CAPITAL_K`
- `:uk_keyboard:CAPITAL_L`
- `:uk_keyboard:CAPITAL_M`

- :uk_keyboard:CAPITAL_N
- :uk_keyboard:CAPITAL_O
- :uk_keyboard:CAPITAL_P
- :uk_keyboard:CAPITAL_Q
- :uk_keyboard:CAPITAL_R
- :uk_keyboard:CAPITAL_S
- :uk_keyboard:CAPITAL_T
- :uk_keyboard:CAPITAL_U
- :uk_keyboard:CAPITAL_V
- :uk_keyboard:CAPITAL_W
- :uk_keyboard:CAPITAL_X
- :uk_keyboard:CAPITAL_Y
- :uk_keyboard:CAPITAL_Z
- :uk_keyboard:CAPS_LOCK
- :uk_keyboard:CIRCUMFLEX_ACCENT
- :uk_keyboard:COLON
- :uk_keyboard:COMMA
- :uk_keyboard:CURLY_BRACKET_LEFT
- :uk_keyboard:CURLY_BRACKET_RIGHT
- :uk_keyboard:CURSOR_DOWN
- :uk_keyboard:CURSOR_LEFT
- :uk_keyboard:CURSOR_RIGHT
- :uk_keyboard:CURSOR_UP
- :uk_keyboard:DELETE
- :uk_keyboard:DOLLAR_SIGN
- :uk_keyboard:END
- :uk_keyboard:EQUALS_SIGN
- :uk_keyboard:ESCAPE
- :uk_keyboard:EXCLAMATION_MARK
- :uk_keyboard:GRAVE_ACCENT
- :uk_keyboard:GREATER_THAN_SIGN
- :uk_keyboard:HOME

- :uk_keyboard:INSERT
- :uk_keyboard:LESS_THAN_SIGN
- :uk_keyboard:MINUS_SIGN
- :uk_keyboard:NUMBER_SIGN
- :uk_keyboard:NUM_LOCK
- :uk_keyboard:PAGE_DOWN
- :uk_keyboard:PAGE_UP
- :uk_keyboard:PARENTHESIS_LEFT
- :uk_keyboard:PARENTHESIS_RIGHT
- :uk_keyboard:PAUSE
- :uk_keyboard:PERCENT_SIGN
- :uk_keyboard:PERIOD
- :uk_keyboard:PLUS_SIGN
- :uk_keyboard:POUND_SIGN
- :uk_keyboard:PRINT
- :uk_keyboard:QUESTION_MARK
- :uk_keyboard:QUOTATION
- :uk_keyboard:RETURN
- :uk_keyboard:SCROLL_LOCK
- :uk_keyboard:SEMICOLON
- :uk_keyboard:SLASH
- :uk_keyboard:SMALL_A
- :uk_keyboard:SMALL_B
- :uk_keyboard:SMALL_C
- :uk_keyboard:SMALL_D
- :uk_keyboard:SMALL_E
- :uk_keyboard:SMALL_F
- :uk_keyboard:SMALL_G
- :uk_keyboard:SMALL_H
- :uk_keyboard:SMALL_I
- :uk_keyboard:SMALL_J
- :uk_keyboard:SMALL_K

- :uk_keyboard:SMALL_L
- :uk_keyboard:SMALL_M
- :uk_keyboard:SMALL_N
- :uk_keyboard:SMALL_O
- :uk_keyboard:SMALL_P
- :uk_keyboard:SMALL_Q
- :uk_keyboard:SMALL_R
- :uk_keyboard:SMALL_S
- :uk_keyboard:SMALL_T
- :uk_keyboard:SMALL_U
- :uk_keyboard:SMALL_V
- :uk_keyboard:SMALL_W
- :uk_keyboard:SMALL_X
- :uk_keyboard:SMALL_Y
- :uk_keyboard:SMALL_Z
- :uk_keyboard:SPACE
- :uk_keyboard:SQUARE_BRACKET_LEFT
- :uk_keyboard:SQUARE_BRACKET_RIGHT
- :uk_keyboard:TAB
- :uk_keyboard:TILDE
- :uk_keyboard:UNDERSCORE
- :uk_keyboard:VERTICAL_BAR

Appendix F The MHEG 5 Broadcast Profile Button Profile

The button profile `:mheg5bp` is defined here for the control of MHEG-5 applications that conform to the "MHEG-5 Broadcast Profile" set out in ETSI ES 202184 v2.1.1 [21]. It consists of the following button codes:

- ::0
- ::1
- ::2
- ::3
- ::4
- ::5

- ::6
- ::7
- ::8
- ::9
- ::RED
- ::GREEN
- ::YELLOW
- ::BLUE
- ::TEXT
- ::CURSOR_UP
- ::CURSOR_DOWN
- ::CURSOR_LEFT
- ::CURSOR_RIGHT
- ::OK
- ::STOP
- ::PLAY
- ::PAUSE
- ::SKIPBACK
- ::SKIPFORWARD
- ::REWIND
- ::FASTFORWARD