

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2013

C. Boulton
NS-Technologies
L. Miniero
Meetecho
G. Munson
AT&T
October 31, 2012

Media Resource Brokering
draft-ietf-mediactrl-mrb-16

Abstract

The MediaCtrl work group in the IETF has proposed an architecture for controlling media services. The Session Initiation Protocol (SIP) is used as the signalling protocol which provides many inherent capabilities for message routing. In addition to such signalling properties, a need exists for intelligent, application level media service selection based on non-static signalling properties. This is especially true when considered in conjunction with deployment architectures that include 1:M and M:N combinations of Application Servers and Media Servers. This document introduces a Media Resource Broker (MRB) entity which manages the availability of Media Servers and the media resource demands of Application Servers. The document includes potential deployment options for an MRB and appropriate interfaces to Application Servers and Media Servers.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
2.	Conventions and Terminology	7
3.	Problem Discussion	8
4.	Deployment Scenario Options	9
4.1.	Query MRB	9
4.1.1.	Hybrid Query MRB	10
4.2.	In-Line MRB	11
5.	MRB Interface Definitions	14
5.1.	Media Server Resource Publish Interface	14
5.1.1.	Control Package Definition	15
5.1.2.	Element Definitions	16
5.1.3.	<mrbrequest>	17
5.1.4.	<mrbresponse>	19
5.1.5.	<mrbnofication>	20
5.2.	Media Service Resource Consumer Interface	32
5.2.1.	Query Mode / HTTP Consumer Interface Usage	32
5.2.2.	In-Line Aware Mode / SIP Consumer Interface Usage	33
5.2.3.	Consumer Interface Lease Mechanism	36
5.2.4.	<mrbcconsumer>	39
5.2.5.	Media Service Resource Request	39
5.2.6.	Media Service Resource Response	51
5.3.	In-Line Unaware MRB Interface	54
6.	MRB acting as a B2BUA	56
7.	Multi-modal MRB Implementations	57
8.	Relative Merits of Query Mode, IAMM, and IUMM	58
9.	Examples	60
9.1.	Publish Example	60
9.2.	Consumer Example	65
9.2.1.	Query Example	65
9.2.2.	IAMM Example	68
10.	Media Service Resource Publisher Interface XML Schema	84
11.	Media Service Resource Consumer Interface XML Schema	107
12.	Security Considerations	128

13.	IANA Considerations	131
13.1.	Media Control Channel Framework Package Registration . .	131
13.2.	application/mrb-publish+xml Media Type	131
13.3.	application/mrb-consumer+xml Media Type	132
13.4.	URN Sub-Namespace Registration for mrb-publish	133
13.5.	URN Sub-Namespace Registration for mrb-consumer	133
13.6.	XML Schema Registration for mrb-publish	133
13.7.	XML Schema Registration for mrb-consumer	134
14.	Changes	135
14.1.	Changes from 13 Version	135
14.2.	Changes from 12 Version	135
14.3.	Changes from 11 Version	135
14.4.	Changes from 10 Version	135
14.5.	Changes from 09 Version	136
14.6.	Changes from 08 Version	136
14.7.	Changes from 07 Version	136
14.8.	Changes from 06 Version	137
14.9.	Changes from 05 Version	137
14.10.	Changes from 04 Version	137
14.11.	Changes from 03 Version	137
14.12.	Changes from 02 Version	138
14.13.	Changes from 01 Version	138
14.14.	Changes from 00 Version	138
15.	Acknowledgements	139
16.	References	140
16.1.	Normative References	140
16.2.	Informative References	141
	Authors' Addresses	143

1. Introduction

As IP based multimedia infrastructures mature, the complexity and demands from deployments increase. Such complexity will result in a wide variety of capabilities from a range of vendors that should all be interoperable using the architecture and protocols produced by the MediaCtrl work group. It should be possible for a controlling entity to be assisted in Media Server selection so that the most appropriate resource is selected for a particular operation. The importance increases when you introduce a flexible level of deployment scenarios, as specified in the [RFC 5167](#) [[RFC5167](#)] and [RFC 5567](#) [[RFC5567](#)] documents. These documents make statements like "it should be possible to have a many-to-many relationship between Application Servers and Media Servers that use this protocol". This leads to the following deployment architectures being possible when considering media resources, to provide what can be effectively described as Media Resource Brokering.

The simplest deployment view is illustrated in Figure 1.

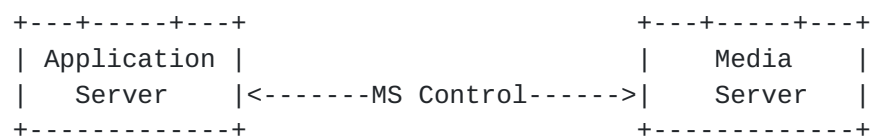


Figure 1: Basic Architecture

This simply involves a single Application Server and Media Server. Expanding on this view, it is also possible for an Application Server to control multiple (greater than 1) Media Server instances at any one time. This deployment view is illustrated in Figure 2. Typically, such architectures are associated with application logic that requires high demand media services. It is more than possible that each media server possesses a different media capability set. Media servers may offer different media services as specified in the MediaCtrl architecture document. A Media server may have similar media functionality but may have different capacity or media codec support.

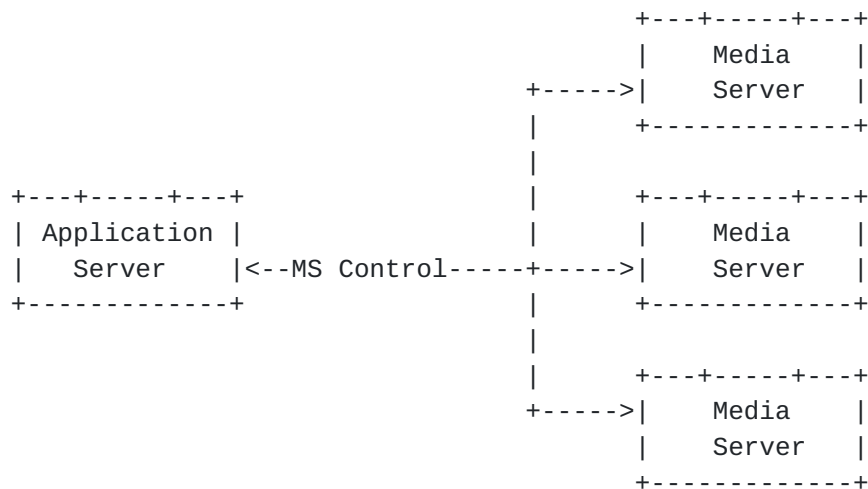


Figure 2: Multiple Media Servers

Figure 3 conveys the opposite view to that in Figure 2. In this model there are a number of (greater than 1) application servers, possibly supporting dissimilar applications, controlling a single media server. Typically, such architectures are associated with application logic that requires low demand media services.

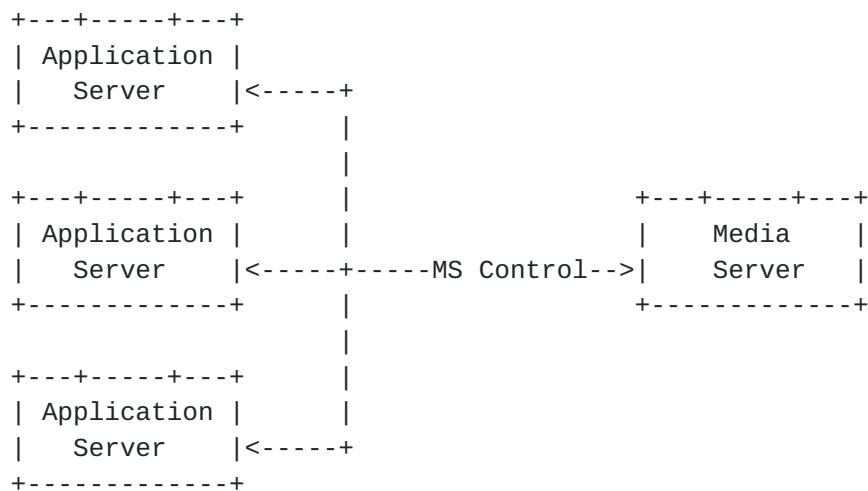


Figure 3: Multiple Application Servers

The final deployment view is the most complex. In this model (M:N) there exists any number of Application Servers and any number of Media Servers. It is again possible in this model that media servers might not be homogeneous and have different capability sets and

capacity.

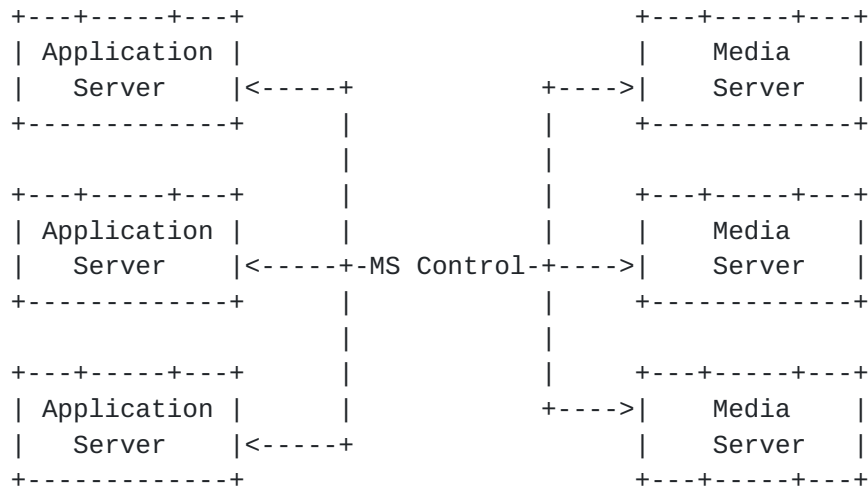


Figure 4: Basic Architecture

The remaining sections in this specification will focus on a new entity called a Media Resource Broker (MRB) which can be utilised in the deployment architectures described previously in this section. The MRB entity provides the ability to obtain media resource information and appropriately allocate(broker) on behalf of client applications.

The high level deployment options discussed in this section rely on network architecture and policy to prohibit inappropriate use. Such policies are out of the scope of this document.

This document will take a look at the specific problem areas related to such deployment architectures. It is recognised that the solutions proposed in this document should be equally adaptable to all of the previously described deployment models. It is also recognised that the solution is far more relevant to some of the previously discussed deployment models and can almost be viewed as redundant on others.

2. Conventions and Terminology

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](#) [[RFC2119](#)].

This document inherits terminology proposed in [RFC 5567](#) [[RFC5567](#)] and Media Control Channel Framework [[RFC6230](#)] documents. In addition, the following terms are defined for use in this document and for use in the context of the MediaCtrl Work group in the IETF:

Media Resource Broker (MRB): A logical entity that is responsible for both collection of appropriate published Media Server (MS) information and selecting appropriate Media Server resources on behalf of consuming entities.

Query MRB: An instantiation of an MRB (See previous definition) that provides an interface for an Application Server to retrieve the address of an appropriate Media Server. The result returned to the Application Server can be influenced by information contained in the query request.

In-line MRB: An instantiation of an MRB (See previous definition) that directly receives requests on the signalling path. There is no separate query.

CFW: Media Control Channel Framework, as specified in [[RFC6230](#)].

Within the context of In-line MRBs, additional terms are defined:

In-line Aware MRB Mode (IAMM): Defined in [Section 5.2.2.1](#).

In-line Unaware MRB Mode (IUMM): Defined in [Section 5.3](#).

The document will often specify when a specific identifier in a protocol message needs to be unique. Unless stated otherwise, such uniqueness will always be within the scope of the Media Servers controlled by the same Media Resource Broker. The interaction between different Media Resource Broker instances, as the partitioning of a logical Media Resource Broker, is out of scope to this document.

3. Problem Discussion

As discussed in [Section 1](#), a goal of the MediaCtrl group is to produce a solution that will service a wide variety of deployment architectures. Such architectures range from the simplest 1:1 relationship between Media Servers and Application Servers to potentially linearly scaling 1:M, M:1 and M:N deployments.

Managing such deployments is itself non-trivial for the proposed solution until an additional number of factors are included into the equation that increase complexity. As Media Servers evolve it must be taken into consideration that, where many can exist in a deployment, they may not have been produced by the same vendor and may not have the same capability set. It should be possible for an Application Server that exists in a deployment to select a Media Service based on a common, appropriate capability set. In conjunction with capabilities, it is also important to take available resources into consideration. The ability to select an appropriate Media Service function is an extremely useful feature but becomes even more powerful when considered with available resources for servicing a request.

In conclusion, the intention is to create a tool set that allows MediaCtrl deployments to effectively utilize the available media resources. It should be noted that in the simplest deployments where only a single media server exists, an MRB function is probably not required. Only a single capability set exists and resource availability can be handled using the appropriate underlying signalling, e.g., SIP response. This document does not prohibit such uses of an MRB, it simply provides the tools for various entities to interact where appropriate. It is also worth noting that the functions specified in this document aim to provide a 'best effort' view of media resources at the time of request for initial Media Server routing decisions. Any dramatic change in media capabilities or capacity after a request has taken place should be handled by the underlying protocol.

It should be noted that there may be additional information that it is desirable for the MRB to have for purposes of selecting a Media Server resource, such as resource allocation rules across different applications, planned or unplanned downtime of Media Server resources, the planned addition of future Media Server resources, or Media Server resource capacity models. How the MRB acquires such information is outside the scope of this document. The specific techniques used for selecting an appropriate Media Resource by an MRB is also outside the scope of this document.

4. Deployment Scenario Options

Research into Media Resource Brokering concluded that a couple of high level models provided an appropriate level of flexibility. The general principles of "in-line" and "query" MRB concepts are discussed in the rest of this section. It should be noted that while the interfaces are different they both use common underlying mechanisms defined in this specification.

4.1. Query MRB

The "Query" model for MRB interactions provides the ability for a client of media services (for example an Application Server) to "ask" an MRB for an appropriate Media Server, as illustrated in Figure 5.

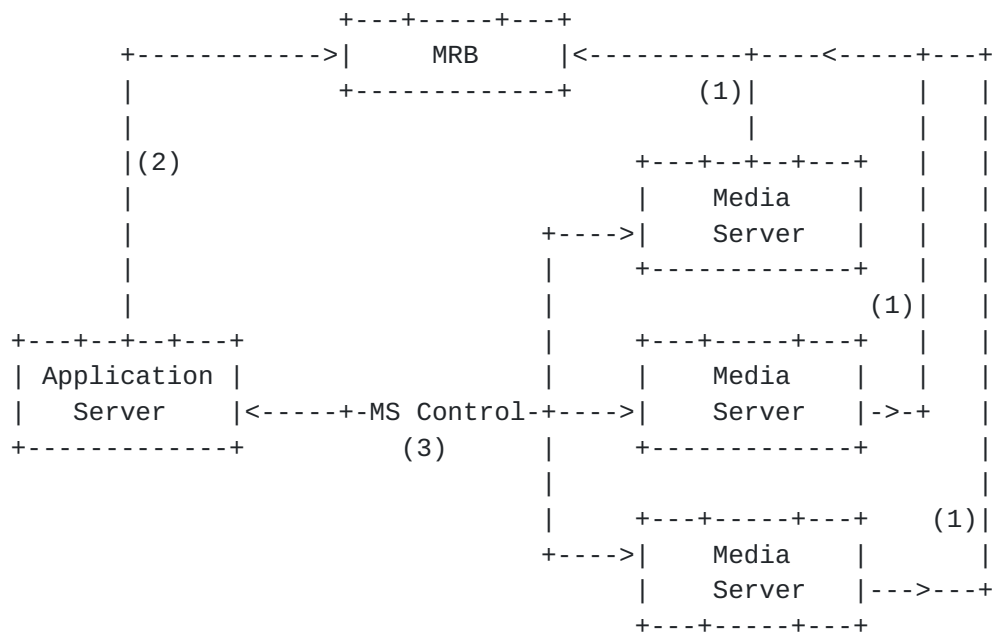


Figure 5: Query MRB

In this deployment, the Media Servers use the "Media Server Resource Publish Interface", as discussed in [Section 5.1](#), to convey capability sets as well as resource information. This is depicted by (1) in Figure 5. It is then the MRB's responsibility to accumulate all appropriate information relating to media services in the logical deployment cluster. The Application Server (or other media services client) is then able to query the MRB for an appropriate resource (as identified by (2) in Figure 5). Such a query would carry specific information related to the Media Service required and enable the MRB to provide an increased accuracy in its response. This particular

interface is discussed in "Media Resource Consumer Interface" in [Section 5.2](#). The Application Server is then able to direct control commands (for example create conference) and Media Dialogs to the appropriate Media Server, as shown by (3) in Figure 5. Additionally, with Query mode, the MRB is not directly in the signalling path between the Application Server and the selected Media Server resource.

[4.1.1.1](#). Hybrid Query MRB

As mentioned previously, it is the intention that a tool kit is provided for MRB functionality within a MediaCtrl architecture. It is expected that in specific deployment scenarios the role of the MRB might be co-hosted as a hybrid logical entity with an Application Server, as shown in Figure 6.

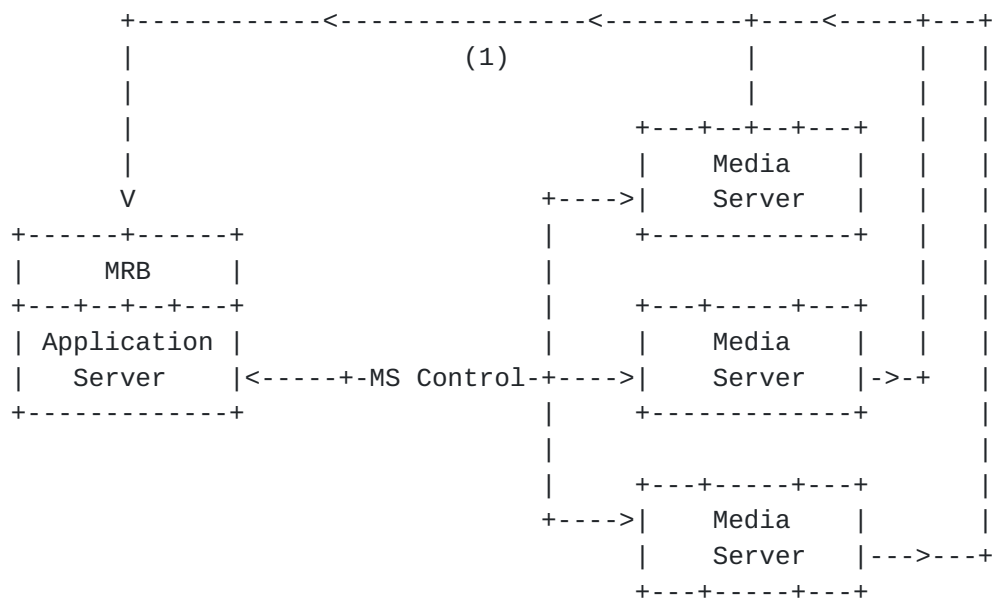


Figure 6: Hybrid Query MRB - Application Server Hosted

This diagram is identical to that in Figure 5 with the exception that the MRB is now hosted on the Application Server. The "Media Server Publish Interface" is still being used to accumulate resource information at the MRB but as it is co-hosted on the Application Server, the "Media Server Consumer Interface" has collapsed. It might still exist within the Application Server/MRB interaction but this is an implementation issue. This type of deployment suits a single Application Server environment but it should be noted that a "Media Server Consumer Interface" could then be offered from the hybrid if required.

In a similar manner, the Media Server could also act as a hybrid for the deployment cluster, as illustrated in Figure 7.

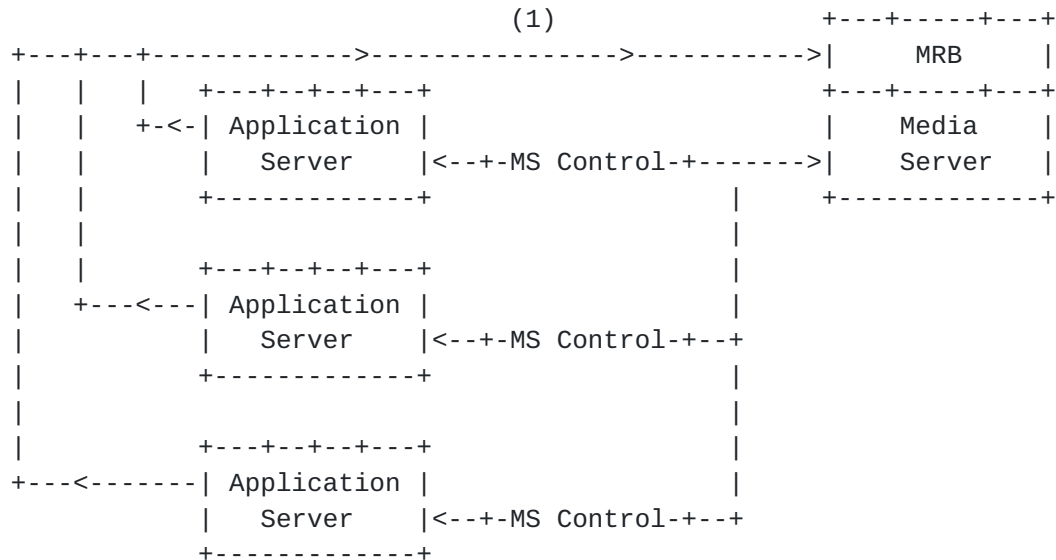


Figure 7: Hybrid Query MRB - MS Hosted

In this example, the MRB has collapsed and is co-hosted by the Media Server. The "Media Server Consumer Interface" is still available to the Application Servers (1) to query Media Server resources. The "Media Server Publish Interface" has collapsed onto the Media Server. It might still exist within the Media Server/MRB interaction but this is an implementation issue. This type of deployment suits a single Media Server environment but it should be noted that a "Media Server Publish Interface" could then be offered from the hybrid if required. A typical use case scenario for such a topology would be a single Media Server representing a pool of MSs in a cluster. In this case, the MRB would actually be handling a cluster of Media Servers, rather than one.

[4.2.](#) In-Line MRB

The "In-line" MRB is architecturally different from the "Query" model discussed in the previous section. The concept of a separate query disappears. The client of the MRB simply uses the media resource control and media dialog signalling to involve the MRB. This type of deployment is illustrated in Figure 8.

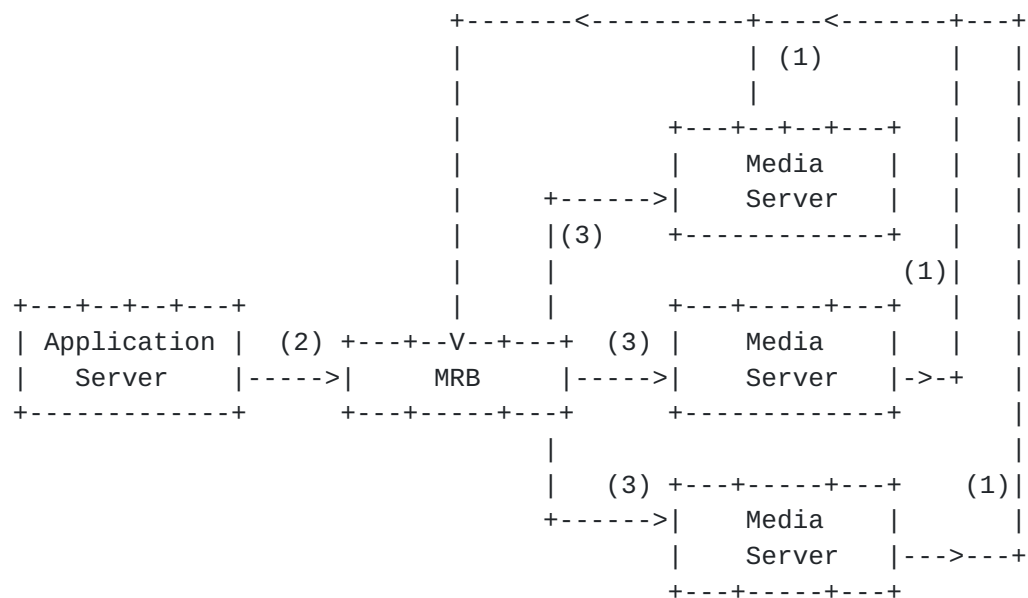


Figure 8: In-line MRB

The Media Servers still use the 'Media Server Publish Interface' to convey capabilities and resources to the MRB - as illustrated by (1). The media server Control (and Media dialogs as well, if required) are sent to the MRB (2) which then selects an appropriate Media Server (3) and remain in the signalling path between the Application Server and the Media Server resources.

In-line MRB can be split into two distinct logical roles which can be applied on a per request basis. They are:

In-line Unaware MRB Mode (IUMM): Allows an MRB to act on behalf of clients requiring media services who are not aware of an MRB or its operation. In this case the Application Server does not provide explicit information on the kind of Media Server resource it needs (as in [Section 5.2](#)) and the MRB is left to deduce it by potentially inspecting other information in the request from the Application Server; for example, SDP content, or address of the requesting Application Server, or additional Request-URI parameters as per [RFC 4240](#) [[RFC4240](#)].

In-line Aware MRB Mode (IAMM): Allows an MRB to act on behalf of clients requiring media services who are aware of an MRB and its operation. In particular it allows the Application Server to explicitly convey matching characteristics to those provided by media servers, as does the Query MRB mode (as in [Section 5.2](#)).

In either of the previously described roles, signalling as specified

by the Media Control Channel Framework ([[RFC6230](#)]) would be involved, and the MRB would deduce that the selected Media Server resources are no longer needed when the Application Server or Media Server terminates the corresponding SIP dialog. The two modes are discussed in more detail in [Section 5.3](#).

5. MRB Interface Definitions

The intention of this specification is to provide a tool-kit for a variety of deployment architectures where media resource brokering can take place. Two main interfaces are required to support the differing requirements. The two interfaces are described in the remainder of this section and have been named the 'Media Server Resource Publish' and 'Media Server Resource Consumer' interfaces.

It is beyond the scope of this document to define exactly how to construct an MRB using the interfaces described. It is, however, important that the two interfaces are complimentary so that development of appropriate MRB functionality is supported.

5.1. Media Server Resource Publish Interface

The Media Server Resource Publish interface is responsible for providing an MRB with appropriate Media Server resource information. As such, this interface is assumed to provide both general and specific details related to Media Server resources. This information needs to be conveyed using an industry standard mechanism to provide increased levels of adoption and interoperability. A Control Package for the Media Control Channel Framework will be specified to fulfil this interface requirement. It provides an establishment and monitoring mechanism to enable a Media Server to report appropriate statistics to an MRB. The Publish interface is used with both Query and In-line modes of MRB operation.

As already discussed in the [Section 1](#), the MRB view of Media Server resource availability will in reality be approximate - i.e., partial and imperfect. The MRB Publish interface does not provide an exhaustive view of current Media Server resource consumption, the Media Server may in some cases provide a best-effort computed view of resource consumption parameters conveyed in the Publish interface (e.g., DSP's with a fixed number of streams versus GPU's with CPU availability). Media Resource information may only be reported periodically over the Publish interface to an MRB.

It is also worth noting that, while the scope of the MRB is in providing interested Application Servers with the available resources, the MRB also allows for the retrieval of information about consumed resources. While this is of course a relevant piece of information (e.g., for monitoring purposes), such functionality inevitably raises security considerations, and implementations should take this into account. See [Section 12](#) for more details.

The MRB Publish interface uses the Media Control Channel Framework ([[RFC6230](#)]) as the basis for interaction between a Media Server and

an MRB. The Media Control Channel Framework uses an extension mechanism to allow specific usages which are known as control packages. [Section 5.1.1](#) defines the control package that MUST be implemented by any Media Server wanting to interact with an MRB entity.

[5.1.1](#). Control Package Definition

This section fulfils the mandatory requirement for information that must be specified during the definition of a Control Framework Package, as detailed in [Section 8 of \[RFC6230\]](#).

[5.1.1.1](#). Control Package Name

The Media Channel Control Framework requires a Control Package definition to specify and register a unique name and version.

The name and version of this Control Package is "mrbc-publish/1.0".

[5.1.1.2](#). Framework Message Usage

The MRB publish interface allows a media server to convey available capabilities and resources to an MRB entity.

This package defines XML elements in [Section 5.1.2](#) and provides an XML Schema in [Section 10](#).

The XML elements in this package are split into requests, responses and event notifications. Requests are carried in CONTROL message bodies; <mrbrequest> element is defined as a package request. This request can be used for creating new subscriptions and updating/removing existing subscriptions. Event notifications are also carried in CONTROL message bodies; the <mrbcnotification> element is defined for package event notifications. Responses are carried either in REPORT message or Control Framework 200 response bodies; the <mrbcresponse> element is defined as a package level response.

Note that package responses are different from framework response codes. Framework error response codes (see [Section 7 of \[RFC6230\]](#)) are used when the request or event notification is invalid; for example, a request has invalid XML (400), or is not understood (500). Package level responses are carried in framework 200 response or REPORT message bodies. This package's response codes are defined in [Section 5.1.4](#).

5.1.1.3. Common XML Support

The Media Control Channel Framework [[RFC6230](#)] requires a Control Package definition to specify if the attributes for media dialog or conference references are required.

The Publish interface defined in [Section 10](#) does import and make use of the common XML schema defined in the Media Control Channel Framework.

The Consumer interface defined in [Section 11](#) does import and make use of the common XML schema defined in the Media Control Channel Framework.

5.1.1.4. CONTROL Message Body

A valid CONTROL body message MUST conform to the schema defined in [Section 10](#) and described in [Section 5.1.2](#). XML messages appearing in CONTROL messages MUST contain either a <mrbrequest> or <mrnotification> element.

5.1.1.5. REPORT Message Body

A valid REPORT body MUST conform to the schema defined in [Section 10](#) and described in [Section 5.1.2](#). XML messages appearing in REPORT messages MUST contain a <mrresponse> element.

5.1.1.6. Audit

The 'mrb-publish/1.0' Media Control Channel Framework package does not require any additional auditing capability.

5.1.2. Element Definitions

This section defines the XML elements for the Publish interface Media Control Channel package defined in [Section 5.1](#). The formal XML schema definition for the Publish interface can be found in [Section 10](#).

The root element is <mrbpublish>. All other XML elements (requests, responses, notifications) are contained within it. The MRB Publish interface request element is detailed in [Section 5.1.3](#). The MRB Publish interface notification element is detailed in [Section 5.1.5](#). MRB Publish interface response element is contained in [Section 5.1.4](#).

The <mrbpublish> element has zero or more of the following attributes:

version: a token specifying the mrb-publish package version. The value is fixed as '1.0' for this version of the package. The attribute MUST be present.

The <mrbpublish> element has the following child elements, only one of which is allowed to occur in a request.

<mrbrequest> for sending an MRB request. See [Section 5.1.3](#).

<mrbresponse> for sending an MRB response. See [Section 5.1.4](#).

<mrbnotification> for sending an MRB notification. See [Section 5.1.5](#).

[5.1.3](#). <mrbrequest>

This section defines the <mrbrequest> element used to initiate requests from an MRB to a Media Server. The element describes information relevant for the interrogation of a media server.

The <mrbrequest> element has no defined attributes.

The <mrbrequest> element has zero or more of the following child elements:

<subscription> for initiating a subscription to a Media Server from an MRB. See [Section 5.1.3.1](#).

[5.1.3.1](#). <subscription>

The <subscription> element is included in a request from an MRB to a Media Server to provide the details relating to the configuration of updates. This element can be used either to request a new subscription or to update an existing one (e.g., to change the frequency of the updates), and to remove ongoing subscriptions as well (e.g., to stop an indefinite update). The MRB will inform the Media Server how long it wishes to receive updates for and the frequency that updates should be sent. Updates related to the subscription are sent using the <mrbnotification> element.

The <subscription> element has the following attributes:

id: indicates a unique token representing the subscription session between the MRB and the Media Server. The attribute MUST be present.

seqnumber: indicates a sequence number to be used in conjunction with the subscription session id to identify a specific subscription command. The first subscription MUST have 1 as 'seqnumber', and following subscriptions MUST increment by 1 the previous 'seqnumber' value. The attribute MUST be present.

action: provides the operation that should be carried out on the subscription:

- * The value of 'create' instructs the Media Server to attempt to set-up a new subscription.
- * The value of 'update' instructs the Media Server to attempt to update an existing subscription.
- * The value of 'remove' instructs the Media Server to attempt to remove an existing subscription and consequently stop any ongoing related notification.

The attribute MUST be present.

The <subscription> element has zero or more of the following child elements:

<expires>: Provides the amount of time in seconds that a subscription should be installed for notifications at the Media Server. Once the amount of time has passed, the subscription expires and the MRB has to subscribe again in case it is still interested in receiving notifications from the Media Server. The element MAY be present.

<minfrequency>: Provides the minimum frequency in seconds that the MRB wishes to receive notifications from the Media Server. The element MAY be present.

<maxfrequency>: Provides the maximum frequency in seconds that the MRB wishes to receive notifications from the Media Server. The element MAY be present.

Please note that these three optional pieces of information provided by the MRB only act as a suggestion: the Media Server MAY change the proposed values if it considers the suggestions unacceptable (e.g., if the MRB has requested a too high notification frequency). In such case, the request would not fail, but the updated, acceptable values would be reported in the <mrresponse> accordingly.

5.1.4. <mrbrresponse>

Responses to requests are indicated by a <mrbrresponse> element.

The <mrbrresponse> element has the following attributes:

status: numeric code indicating the response status. The attribute MUST be present.

reason: string specifying a reason for the response status. The attribute MAY be present.

The <mrbrresponse> element has zero or more of the following child elements:

<subscription> for providing details related to a subscription a Media Server requested (see below in this section).

The following status codes are defined for 'status':

code	description
200	OK
400	Syntax error
401	Unable to create Subscription
402	Unable to update Subscription
403	Unable to remove Subscription
404	Subscription does not exist
405	Subscription already exists
420	Unsupported attribute or element

Table 1: <mrbrresponse> status codes

In case a new subscription request made by an MRB (action='create') has been accepted, the Media Server MUST reply with a <mrbrresponse> with status code 200. The same rule applies whenever a request to update (action='update') or remove (action='remove') an existing transaction can be fulfilled by the Media Server.

A subscription request, nevertheless, may fail for several reasons. In such a case, the status codes defined in Table 1 must be used instead. Specifically, if the Media Server fails to handle a request due to a syntax error in the request itself (e.g., incorrect XML, violation of the schema constraints or invalid values in any of the attributes/elements) the Media Server MUST reply with a <mrbresponse> with status code 400. If a syntactically correct request fails because the request also includes any attribute/element the Media Server doesn't understand, the Media Server MUST reply with a <mrbresponse> with status code 420. If a syntactically correct request fails because the MRB wants to create a new subscription, but the provided unique 'id' for the subscription already exists, the Media Server MUST reply with a <mrbresponse> with status code 405. If a syntactically correct request fails because the MRB wants to update/remove a subscription that doesn't exist, the Media Server MUST reply with a <mrbresponse> with status code 404. If the Media Server is unable to accept a request for any other reason (e.g., the MRB has no more resources to fulfil the request), the Media Server MUST reply with a <mrbresponse> with status code 401/402/403, depending on the action the MRB provided in its request:

- o action='create' --> 401;
- o action='update' --> 402;
- o action='remove' --> 403;

A response to a subscription request that has a status of "200" indicates that the request is successful. The response MAY also contain a <subscription> child that describes the subscription. The <subscription> child MAY contain 'expires', 'minfrequency' and 'maxfrequency' values even if they were not contained in the request.

The Media Server MAY change the suggested 'expires', 'minfrequency' and 'maxfrequency' values provided by the MRB in its <mrbrequest>, if it considers them unacceptable (e.g., the requested frequency range is too high). In such a case, the response MUST contain a <subscription> element describing the subscription as the Media Server accepted it, and the Media Server MUST include in the <subscription> element all of those values that it modified relative to the request, to inform the MRB about the change.

5.1.5. <mrbnotification>

The <mrbnotification> element is included in a request from a Media Server to an MRB to provide the details relating current status. The Media Server will inform the MRB of its current status as defined by the information in the <subscription> element. Updates are sent

using the `<mrnotification>` element.

The `<mrnotification>` element has the following attributes:

`id`: indicates a unique token representing the session between the MRB and the Media Server and is the same as the one appearing in the `<subscription>` element. The attribute **MUST** be present.

`seqnumber`: indicates a sequence number to be used in conjunction with the subscription session id to identify a specific notification update. The first notification **MUST** have 1 as 'seqnumber', and following notifications **MUST** increment by 1 the previous 'seqnumber' value. The attribute **MUST** be present.

It's important to point out that the 'seqnumber' that appears in a `<mrnotification>` is not related to the 'seqnumber' appearing in a `<mrsubscription>`. In fact, the latter is associated with subscriptions and would increase at every command issued by the MRB, while the former is associated with the asynchronous notifications the Media Server would trigger according to the subscription, and as such would increase at every notification message to enable the MRB keep track of them.

The following subsections provide details of the child elements that are the content of the `<mrnotification>` element.

5.1.5.1. `<media-server-id>`

The `<media-server-id>` element provides a unique system wide identifier for a Media Server instance. The element **MUST** be present, and **MUST** be chosen such that it is extremely unlikely that two different media servers would present the same id to a given MRB.

5.1.5.2. `<supported-packages>`

The `<supported-packages>` element provides the list of Media Control Channel Packages supported by the media server. The element **MAY** be present.

The `<supported-packages>` element has no attributes.

The `<supported-packages>` element has zero or more of the following child elements:

`<package>`: The `<package>` element gives the name of a package supported by the media server. The `<package>` element has a single attribute, 'name', which provides the name of the supported Media Control Channel Framework package, compliant with the Section

13.1.1 of [[RFC6230](#)].

5.1.5.3. <active-rtp-sessions>

The <active-rtp-sessions> element provides information detailing the current active Real-time Transport Protocol(RTP) sessions. The element MAY be present.

The <active-rtp-sessions> element has no attributes.

The <active-rtp-sessions> element has zero or more of the following child elements:

<rtp-codec>: Describes a supported codec and the number of active sessions using that codec. The <rtp-codec> element has one attribute. The value of the attribute 'name' is a media type (which can include parameters per [[RFC6381](#)]). The <rtp-codec> element has two child elements. The child element, <decoding>, has as content the decimal number of RTP sessions being decoded using the specified codec. The child element, <encoding>, has as content the decimal number of RTP sessions being encoded using the specified codec.

5.1.5.4. <active-mixer-sessions>

The <active-mixer-sessions> element provides information detailing the current active mixed RTP sessions. The element MAY be present.

The <active-mixer-sessions> element has no attributes.

The <active-mixer-sessions> element has zero or more of the following child elements:

<active-mix>: Describes a mixed active RTP session. The <active-mix> element has one attribute. The value of the attribute 'conferenceid' is the name of the mix. The <active-mix> element has one child element. The child element, <rtp-codec>, contains the same information relating to RTP sessions as defined in [Section 5.1.5.3](#). The element MAY be present.

5.1.5.5. <non-active-rtp-sessions>

The <non-active-rtp-sessions> element provides information detailing the currently available inactive RTP sessions, that is, how many more RTP streams this Media Server can support. The element MAY be present.

The <non-active-rtp-sessions> element has no attributes.

The `<non-active-rtp-sessions>` element has zero or more of the following child elements:

`<rtp-codec>`: Describes a supported codec and the number of non-active sessions for that codec. The `<rtp-codec>` element has one attribute. The value of the attribute 'name' is a media type (which can include parameters per [RFC6381](#)). The `<rtp-codec>` element has two child elements. The child element, `<decoding>`, has as content the decimal number of RTP sessions available for decoding using the specified codec. The child element, `<encoding>`, has as content the decimal number of RTP sessions available for encoding using the specified codec.

[5.1.5.6.](#) `<non-active-mixer-sessions>`

The `<non-active-mixer-sessions>` element provides information detailing the current inactive mixed RTP sessions, that is, how many more mixing sessions this Media Server can support. The element MAY be present.

The `<non-active-mixer-sessions>` element has no attributes.

The `<non-active-mixer-sessions>` element has zero or more of the following child element:

`<non-active-mix>`: Describes available mixed RTP sessions. The `<non-active-mix>` element has one attribute. The value of the attribute 'available' is the number of mixes that could be used using that profile. The `<non-active-mix>` element has one child element. The child element, `<rtp-codec>`, contains the same information relating to RTP sessions as defined in [Section 5.1.5.5](#). The element MAY be present.

[5.1.5.7.](#) `<media-server-status>`

The `<media-server-status>` element provides information detailing the current status of the media server. The element MUST be present. It can return one of the following values:

active: Indicating that the Media Server is available for service.

deactivated: Indicating that the Media Server has been withdrawn from service, and as such requests should not be sent to it before it becomes 'active' again.

unavailable: Indicating that the Media Server continues to process past requests but cannot accept new requests, and as such should not be contacted before it becomes 'active' again.

The <media-server-status> element has no attributes.

The <media-server-status> element has no child elements.

5.1.5.8. <supported-codecs>

The <supported-codecs> element provides information detailing the current codecs supported by a media server and associated actions. The element MAY be present.

The <supported-codecs> element has no attributes.

The <supported-codecs> element has zero or more of the following child element:

<supported-codec>: has a single attribute, 'name', which provides the name of the codec about which this element provides information. A valid value is a media type which, depending on its definition, can include additional parameters (e.g., [[RFC6381](#)]). The <supported-codec> element then has a further child element, <supported-codec-package>. The <supported-codec-package> element has a single attribute, 'name', which provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the codec support applies. The <supported-codec-package> element has zero or more <supported-action> children, each one of which describes an action that a Media Server can apply to this codec:

- * 'decoding', meaning a decoder for this codec is available;
- * 'encoding', meaning an encoder for this codec is available;
- * 'passthrough', meaning the Media Server is able to pass a stream encoded using that codec through without re-encoding.

5.1.5.9. <application-data>

The <application-data> element provides an arbitrary string of characters as application level data. This data is meant to only have meaning at the application level logic and as such is not otherwise restricted by this specification. The set of allowed characters are the same as those in XML (viz., tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [see <http://www.w3.org/TR/xml/section 2.2>]). The element MAY be present.

The <application-data> element has no attributes.

The <application-data> element has no child elements.

5.1.5.10. <file-formats>

The <file-formats> element provides a list of file formats supported for the purpose of playing media. The element MAY be present.

The <file-formats> element has no attributes.

The <file-formats> element has zero or more of the following child elements:

<supported-format>: has a single attribute, 'name', which provides the type of file format that is supported. A valid value is a media type which, depending on its definition, can include additional parameters (e.g., [[RFC6381](#)]). The <supported-format> element then has a further child element, <supported-file-package>. The <supported-file-package> element provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the file format support applies.

5.1.5.11. <max-prepared-duration>

The <max-prepared-duration> element provides the maximum amount of time a media dialog will be kept in the prepared state before timing out before it is executed (see section 4.4.2.2.6 of [RFC 6231](#)[[RFC6231](#)]). The element MAY be present.

The <max-prepared-duration> element has no attributes.

The <max-prepared-duration> element has zero or more of the following child elements:

<max-time>: has a single attribute, 'max-time-seconds', which provides the amount of time in seconds that a media dialog can be in the prepared state. The <max-time> element then has a further child element, <max-time-package>. The <max-time-package> element provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the time period applies.

5.1.5.12. <dtmf-support>

The <dtmf-support> element specifies the supported methods to detect DTMF tones and to generate them. The element MAY be present.

The <dtmf-support> element has no attributes.

The <dtmf-support> element has zero or more of the following child elements:

<detect>: Indicates the support for DTMF detection. The <detect> element has no attributes. The <detect> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it can only be a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (detecting tones as signals from the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

<generate>: Indicates the support for DTMF generation. The <generate> element has no attributes. The <generate> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it can only be a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (generating tones as signals in the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

<passthrough>: Indicates the support for passing DTMF through without re-encoding. The <passthrough> element has no attributes. The <passthrough> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it can only be a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (passing tones as signals through the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

5.1.5.13. <mixing-modes>

The <mixing-modes> element provides information about the support for audio and video mixing of a Media Server, specifically a list of supported algorithms to mix audio and a list of supported video presentation layouts. The element MAY be present.

The <mixing-modes> element has no attributes.

The <mixing-modes> element has zero or more of the following child elements:

<audio-mixing-modes>: Describes the available algorithms for audio mixing. The <audio-mixing-modes> element has no attributes. The <audio-mixing-modes> element has one child element. The child element, <audio-mixing-mode>, contains a specific available algorithm. Valid values for the <audio-mixing-mode> element are algorithm names, e.g., 'nbest' and 'controller' as defined in [\[RFC6505\]](#). The element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the algorithm support applies.

<video-mixing-modes>: Describes the available video presentation layouts and the supported functionality for what concerns video mixing. The <video-mixing-modes> element has two attributes, 'vas' and 'activespeakermix'. The 'vas' attribute is of type boolean with a value of 'true' indicating the Media Server supports automatic Voice Activated Switching. The 'activespeakermix' is of type boolean with a value of 'true' indicating that the Media Server is able to prepare an additional video stream for the loudest speaker participant without its contribution. The <video-mixing-modes> element has one child element. The child element, <video-mixing-mode>, contains the name of a specific video presentation layout. The name may refer to one of predefined video layouts defined in the XCON conference information data model, or to non-XCON layouts as well, as long as they are properly prefixed according to the schema they belong to. The <video-mixing-mode> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the algorithm support applies.

5.1.5.14. <supported-tones>

The <supported-tones> element provides information about which tones a media server is able to play and recognize. In particular, the support is reported referring to both country codes support (ISO

3166-1 [[ISO.3166-1](#)]) and supported functionality (ITU-T Recommendation Q.1950 [[ITU-T.Q.1950](#)]). The element MAY be present.

The <supported-tones> element has no attributes.

The <supported-tones> element has zero or more of the following child elements:

<supported-country-codes>: Describes the supported country codes with respect to tones. The <supported-country-codes> element has no attributes. The <supported-country-codes> has one child element. The child element, <country-code>, reports support for a specific country code, compliant with the ISO 3166-1 [[ISO.3166-1](#)] specification. The <country-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), in which the tones from the specified country code are supported.

<supported-h248-codes>: Describes the supported H.248 codes with respect to tones. The <supported-h248-codes> element has no attributes. The <supported-h248-codes> has one child element. The child element, <h248-code>, reports support for a specific H.248 code, compliant with the ITU-T Recommendation Q.1950 [[ITU-T.Q.1950](#)] specification. The codes can be either specific (e.g., cg/dt to only report the Dial Tone from the Call Progress Tones package) or generic (e.g., cg/* to report all the tones from the Call Progress Tones package) using wild-cards. The <h248-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), in which the specified codes are supported.

[5.1.5.15](#). <file-transfer-modes>

The <file-transfer-modes> element allows the Media Server to specify which scheme names are supported for transferring files to a Media Server for each Media Control Channel Framework package type. For example, whether the Media Server supports fetching resources via HTTP, HTTPS, NFS, RTSP etc protocols. The element MAY be present.

The <file-transfer-modes> element has no attributes.

The <file-transfer-modes> element has zero or more of the following child element:

`<file-transfer-mode>`: has two attributes, 'name' and 'package'. The 'name' attribute provides the scheme name of the protocol that can be used for file transfer (e.g., "HTTP", "RTSP", etc.): the value of the attribute is case insensitive. The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the specification in the related IANA registry (e.g., "msc-ivr/1.0"), for which the scheme name applies.

It is important to point out that this element provides no information about whether or not the Media Server supports any flavour of live streaming: for instance, a value of "HTTP" for the IVR Package would only mean the 'http' scheme makes sense to the Media Server within the context of that package. Whether or not the Media Server can make use of HTTP to only fetch resources, or also to attach an HTTP live stream to a call, is to be considered implementation specific to the Media Server and irrelevant to the Application Server and/or MRB. Besides, the Media Server supporting a scheme does not imply it also supports the related secure versions: for instance, if the Media Server supports both "HTTP" and "HTTPS", both the schemes will appear in the element. A lack of the "HTTPS" value would need to be interpreted as a lack of support for the 'https' scheme.

5.1.5.16. `<asr-tts-support>`

The `<asr-tts-support>` element provides information about the support for Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) functionality in a media server. The functionality are reported by referring to the supported languages (using ISO-639-1 [[ISO.639.1988](#)] codes) for what regards both ASR and TTS. The element MAY be present.

The `<asr-tts-support>` element has no attributes.

The `<asr-tts-support>` element has zero or more of the following child elements:

`<asr-support>`: Describes the available languages for ASR. The `<asr-support>` element has no attributes. The `<asr-support>` has one child element. The child element, `<language>`, reports the Media Server supports ASR for a specific language. The `<language>` element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO-639-1 [[ISO.639.1988](#)] code of the supported language.

`<tts-support>`: Describes the available languages for TTS. The `<tts-support>` element has no attributes. The `<tts-support>` has one child element. The child element, `<language>`, reports the Media Server supports tts for a specific language. The `<language>` element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO-639-1 [[ISO.639.1988](#)] code of the supported language.

[5.1.5.17.](#) `<vxml-support>`

The `<vxml-support>` element specifies if the Media Server supports VoiceXML and if it does which protocols the support is exposed through (e.g., via the control framework, [RFC4240](#) [[RFC4240](#)], or [RFC5552](#) [[RFC5552](#)]). The element MAY be present.

The `<vxml-support>` element has no attributes.

The `<vxml-support>` element has zero or more of the following child elements:

`<vxml-mode>`: has two attributes, 'package' and 'support'. The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the VXML support applies. The 'support' attribute provides the type of VXML support provided by the Media Server (e.g., [RFC5552](#) [[RFC5552](#)], [RFC4240](#) [[RFC4240](#)] or IVR Package [[RFC6231](#)]), and valid values are case insensitive RFC references (e.g., "[rfc6231](#)" to specify the Media Server supports VoiceXML as provided by the IVR Package [[RFC6231](#)]).

The presence of at least one `<vxml-mode>` child element would indicate that the Media Server does support VXML as specified by the child element itself. An empty `<vxml>` element would otherwise indicate the Media Server does not support VXML at all.

[5.1.5.18.](#) `<media-server-location>`

The `<media-server-location>` element provides information about the civic location of a media server. Its description makes use of the Civic Address Schema standardized in [RFC 5139](#) [[RFC5139](#)]. The element MAY be present. More precisely, this section is entirely optional, and it's implementation specific to fill it with just the details each implementor deems necessary for any optimization that may be needed.

The `<media-server-location>` element has no attributes.

The `<media-server-location>` element has zero or more of the following

child elements:

`<civicAddress>`: Describes the civic address location of the media server, whose representation refers to the [Section 4 of RFC 5139](#) [[RFC5139](#)].

[5.1.5.19.](#) `<label>`

The `<label>` element allows a Media Server to declare a piece of information that will be understood by the MRB. For example, the Media Server can declare if it's a blue or green one. It's a string to allow arbitrary values to be returned to allow arbitrary classification. The element MAY be present.

The `<label>` element has no attributes.

The `<label>` element has no child elements.

[5.1.5.20.](#) `<media-server-address>`

The `<media-server-address>` element allows a Media Server to provide a direct SIP URI where it can be reached (e.g., the URI Application Server would call to in order to set-up a Control Channel and relay SIP media dialogs). The element MAY be present.

The `<media-server-address>` element has a single attribute.

The `<media-server-address>` element has no child elements.

[5.1.5.21.](#) `<encryption>`

The `<encryption>` element allows a Media Server to declare support for encrypting RTP media streams using [RFC 3711](#) [[RFC3711](#)]. The element MAY be present.

The `<encryption>` element has no attributes.

The `<encryption>` element has zero or more of the following child elements:

`<keying-mechanism>`: has no attributes. The element provides the name of a keying mechanism the Media Server supports for encrypting RTP media streams.

The presence of at least one `<keying-mechanism>` child element would indicate that the Media Server does support RTP media stream encryption as specified by the child element itself. An empty `<encryption>` element would otherwise indicate the Media Server does

not support RTP encryption at all.

5.2. Media Service Resource Consumer Interface

The Media Server Consumer interface provides the ability for clients of an MRB, such as Application Servers, to request an appropriate Media Server to satisfy specific criteria. The interface allows a client to pass detailed meta-information to the MRB to help select an appropriate Media Server. The MRB is then able to make an informed decision and provide the client with an appropriate media server resource. The MRB Consumer interface includes both 1) In-Line Aware MRB Mode (IAMM) that uses the Session Initiation Protocol (SIP) and 2) Query mode that uses the Hypertext Transfer Protocol (HTTP) [RFC2616]. The MRB Consumer interface does not include In-Line Unaware Mode (IUMM) which is further explained in [Section 5.3](#). The following subsections provide guidance on using the Consumer interface, which is represented by the 'application/mrb-consumer+xml' media type in [Section 11](#), with HTTP and SIP.

5.2.1. Query Mode / HTTP Consumer Interface Usage

An appropriate interface for such a 'query' style interface is in fact a HTTP usage. Using HTTP and XML combined reduces complexity and encourages use of common tools that are widely available in the industry today. The following information explains the primary operations required to request and then receive information from an MRB, by making use of HTTP [RFC2616] and HTTPS [RFC2818] as transport for a query for media resource and the appropriate response.

The media resource query, as defined by the <mediaResourceRequest> element from [Section 11](#), MUST be carried in the body of an HTTP/HTTPS POST request. The media type contained in the HTTP/HTTPS request/response MUST be 'application/mrb-consumer+xml'. This value MUST be reflected in the appropriate HTTP headers like 'Content-Type' and 'Accept'. The body of the HTTP/HTTPS POST request MUST only contain a <mrbconsumer> root element with only one child <mediaResourceRequest> element as defined in [Section 11](#).

The media resource response to a query, as defined by the <mediaResourceResponse> element from [Section 11](#), MUST be carried in the body of an HTTP/HTTPS 200 response to the original HTTP/HTTPS POST request. The media type contained in the HTTP/HTTPS request/response MUST be 'application/mrb-consumer+xml'. This value MUST be reflected in the appropriate HTTP headers like 'Content-Type' and 'Accept'. The body of the HTTP/HTTPS 200 response MUST only contain a <mrbconsumer> root element with only one child <mediaResourceResponse> element as defined in [Section 11](#).

When an Application Server wants to release previously awarded media resources granted through a prior request/response exchange with MRB, it will send a new request with an <action> element with value 'remove' as described in [Section 5.2.3](#) about the use of the Consumer interface lease mechanism.

5.2.2. In-Line Aware Mode / SIP Consumer Interface Usage

This document provides a complete tool-kit for MRB deployment which includes the ability to interact with an MRB using SIP for the Consumer interface. The following information explains the primary operations required to request and then receive information from an MRB, by making use of SIP [[RFC3261](#)] as transport for a request for media resources and the appropriate response when used with IAMM of operation (as discussed in [Section 5.2.2.1](#)).

Use of IAMM, besides having the MRB select appropriate media resources on behalf of a client application, includes setting up either a Control Framework control channel between an application server and one of the media servers ([Section 5.2.2.1](#)) or a media dialog session between an application server and one of the media servers ([Section 5.2.2.2](#)). Note that in either case the SIP URIs of the selected media servers are made known to the requesting application server in the SIP 200 OK response by means of one or more <media-server-address> child elements in the <response-session-info> element ([Section 5.2.6](#)).

5.2.2.1. IAMM and Setting up a Control Framework Control Channel

The media resource request information, as defined by the <mediaResourceRequest> element from [Section 11](#), is carried in a SIP INVITE request. The INVITE request will be constructed as it would have been to connect to a Media Server, as defined by the Media Control Channel Framework [[RFC6230](#)]. It should be noted that this specification does not exclude the use of an offer-less INVITE as defined in [RFC3261](#) [[RFC3261](#)]. Using offer-less INVITE messages to an MRB can potentially cause confusion when applying resource selection algorithms and an MRB, like any other SIP device, can choose to reject with a 4xx response. For an offer-less INVITE to be treated appropriately, additional contextual information would need to be provided with the request which is out of the scope for this document. The following additional steps MUST be followed when using the Consumer interface:

- o The Consumer Client will Include a payload in the SIP INVITE request of type 'multipart/mixed' [[RFC2046](#)]. One of the parts to be included in the 'multipart/mixed' payload MUST be the 'application/sdp' format which is constructed as specified in the

Media Control Channel Framework [[RFC6230](#)].

- o Another part of the 'multipart/mixed' payload MUST be of type 'application/mrb-consumer+xml', as specified in this document and defined in [Section 11](#). The body part MUST be an XML document without prolog and whose root element is <mediaResourceRequest>.
- o The INVITE request will then be dispatched to the MRB, as defined by [[RFC6230](#)].

On receiving a SIP INVITE request containing the multipart/mixed payload as specified previously, the MRB will complete a number of steps to fulfil the request. It will:

- o Extract the multipart MIME payload from the SIP INVITE request. It will then use the contextual information provided by the client in the 'application/mrb-consumer+xml' part to determine which media server (or media servers, if more than one is deemed to be needed) should be selected to service the request.
- o Extract the 'application/sdp' part from the payload and use it as the body of a new SIP INVITE request for connecting the client to one of the selected media servers, as defined in the Media Channel Control Framework [[RFC6230](#)]. The policy the MRB follows to pick a specific Media Server out of the Media Servers it selects is implementation specific, and out of scope to this document. It is important to configure the SIP elements between the MRB and the Media Server in such a way that the INVITE will not fork. In case of a failure in reaching the chosen Media Server, the MRB SHOULD proceed to the next one, if available.

If none of the available Media Server can be reached, the MRB MUST reply with a SIP 503 error message including a Retry-After header with a non-zero value. The Application Server MUST NOT attempt to setup a new session before the time the MRB asked it to wait has passed.

In case at least one Media Server is reachable, the MRB acts as a Back-to-Back UA (B2BUA) that extracts the 'application/mrb-consumer+xml' information from the SIP INVITE request and then sends a corresponding SIP INVITE request to the Media Server it has selected, to negotiate a control channel as defined in the Media Channel Control Framework [[RFC6230](#)].

In case of a failure in negotiating the control channel with the Media Server, the MRB SHOULD proceed to the next one, if available, as explained above. If none of the available Media Servers can be reached, or the negotiation of the control channel with all of them

fail, the MRB MUST reply with a SIP 503 error message including a Retry-After header with a non-zero value. The Application Server MUST NOT attempt to setup a new session before the time the MRB asked it to wait has expired.

Once the MRB receives the SIP response from the selected media resource (i.e., media server), it will in turn respond to the requesting client (i.e., Application Server).

The media resource response generated by an MRB to a request, as defined by the <mediaResourceResponse> element from [Section 11](#), MUST be carried in the payload of a SIP 200 OK response to the original SIP INVITE request. The SIP 200 OK response will be constructed as it would have been to connect from a media server, as defined by the Media Control Channel Framework [[RFC6230](#)]. The following additional steps MUST be followed when using the Consumer interface:

- o Include a payload in the SIP 200 response of type 'multipart/mixed' as per [RFC 2046](#) [[RFC2046](#)]. One of the parts to be included in the 'multipart/mixed' payload MUST be the 'application/sdp' format which is constructed as specified in the Media Control Channel Framework [[RFC6230](#)] and based on the incoming response from the selected Media Resource.
- o Another part of the 'multipart/mixed' payload MUST be of type 'application/mrb-consumer+xml', as specified in this document and defined in [Section 11](#). Only the <mediaResourceResponse> and its child elements can be included in the payload.
- o The SIP 200 response will then be dispatched from the MRB.
- o A SIP ACK to the 200 response will then be sent back to the MRB.

Considering that the use of SIP as a transport for Consumer transactions may result in failure, the IAMB relies on a successful INVITE transaction to address the previously discussed sequence (using 'seq' XML element) increment mechanism. This means that, if the INVITE is unsuccessful for any reason, the Application Server MUST use the same 'seq' value as previously used for the next Consumer request it may want to send to the MRB for the same session.

An MRB implementation may be programmed to conclude that the requested resources are no longer needed when it receives a SIP BYE from the Application Server or Media Server that concludes the SIP dialog that initiated the request, or when the lease interval expires.

5.2.2.2. IAMM and Setting up a Media Dialog

This scenario is identical to the description in the previous section for setting up a Control Framework control channel with the exception that the application/sdp payload conveys content appropriate for setting up the media dialog to the media resource, as per [RFC 3261](#) [[RFC3261](#)], instead of application/sdp payload for setting up a control channel.

5.2.3. Consumer Interface Lease Mechanism

The Consumer interface defined in [Section 5.2](#) and [Section 11](#) allows a client to request an appropriate media resource based on information included in the request (either a HTTP POST or SIP INVITE message). In the case of success, the response that is returned to the client MUST contain a <response-session-info> element in either the SIP 200 or HTTP 200 response. The success response contains the description of certain resources that have been reserved to a specific Consumer client in a (new or revised) "resource session", which is identified in the <response-session-info>. The resource session is a "lease", in that the reservation is scheduled to expire at a particular time in the future, releasing the resources to be assigned for other uses. The lease may be extended or terminated earlier by future Consumer client requests that identify and reference a specific resource session.

Before delving into the details of such lease mechanism, it is worth clarifying its role within the context of the Consumer interface. As explained in [Section 5.1](#), the knowledge the MRB has of the resources of all the Media Servers it is provisioned to manage is non real-time. How an MRB actually manages such resources is implementation specific - for example an implementation may choose to have the MRB keeping track and state of the allocated resources, or simply rely on the Media Servers themselves to provide the information using the Publish interface. Further information may be also be inferred by the signalling, in the case where an MRB is in the path of media dialogs.

The <mediaResourceResponse> element returned from the MRB contains a <response-session-info> element if the request is successful. The <response-session-info> element has zero or more of the following child elements which provide the appropriate resource session information:

- o <session-id> is a unique identifier that enables a Consumer client and MRB to correlate future media resource requests related to an initial media resource request. The <session-id> MUST be included in all future related requests (see <session-id> use later in this

section when constructing a subsequent request).

- o `<seq>` is a numeric value returned to the Consumer client. On issuing any future requests related to the media resource session (as determined by the `<session-id>` element) the consumer client MUST increment the value returned in the `<seq>` element and include in the request (see `<seq>` use later in this section when constructing a subsequent request). Its value is a non-negative integer, which MUST be limited within the $0..2^{31}-1$ range.
- o `<expires>` provides a value which provides the number of seconds the request for media resources is deemed alive. The Consumer client should issue a refresh of the request, as discussed later in this section, if the expires timer is due to fire and the media resources are still required.
- o `<media-server-address>` provides information representing an assigned Media Server. More instances of this element may appear, should the MRB assign more Media Servers to a Consumer request.

The `<mediaResourceRequest>` element is used in subsequent Consumer interface requests if the client wishes to manipulate the session. The Consumer client MUST include the `<session-info>` element which enables the receiving MRB to determine an existing media resource allocation session. The `<session-info>` element has the following child elements which provide the appropriate resource session information to the MRB:

- o `<session-id>` is a unique identifier that allows a Consumer client to indicate the appropriate existing media resource session to be manipulated by the MRB for this request. The value was provided by the MRB in the initial request for media resources, as discussed earlier in this section (`<session-id>` element included as part of the `<session-info>` element in the initial `<mediaResourceResponse>`).
- o `<seq>` is a numeric value returned to Consumer client in the initial request for media resources, as discussed earlier in this section (`<seq>` element included as part of the `<session-info>` element in the initial `<mediaResourceResponse>`). On issuing any future requests related to the specific media resource session (as determined by the `<session-id>` element) the consumer client MUST increment the value returned in the `<seq>` element from the initial response (contained in the `<mediaResourceResponse>`) for every new request. The value of the `<seq>` element in requests acts as a counter to and in conjunction with the unique `<session-id>` allows for unique identification of a request. As anticipated before, the value of a `<seq>` value is limited to the $0..2^{31}-1$ range: in

the unlikely case the counter increases to reach the highest allowed value, the <seq> value MUST be set to 0. The first numeric value for the <seq> element is not meant to be '1', but SHOULD be generated randomly by the MRB: this is to reduce the chances a malicious MRB disrupts the session created by this MRB, as explained in [Section 12](#).

- o <action> element provides the operation to be carried out by the MRB on receiving the request:
 - * The value of 'update' is a request by the Consumer client to update the existing session at the MRB with alternate media resource requirements. If the requested resource information is identical to the existing MRB session, the MRB will attempt a session refresh. If the information has changed, the MRB will attempt to update the existing session with the new information. If the operation is successful, the 200 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element. If the operation is not successful, a 409 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element.
 - * The value of 'remove' is a request by the Consumer client to remove the session at the MRB. This provides a mechanism for Consumer clients to release unwanted resources before they expire. If the operation is successful, a 200 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element. If the operation is not successful, a 410 status code in the response is returned in the status attribute of the <mediaResourceResponseType> element.

Omitting the 'action' attribute means requesting a new set of resources.

When used with HTTP the <session-info> element MUST be included in a HTTP POST message (as defined in [\[RFC2616\]](#)). When used with SIP, instead, the <session-info> element MUST be included in either a SIP INVITE, or a SIP re-INVITE (as defined in [\[RFC3261\]](#)) or a SIP UPDATE (as defined in [\[RFC3311\]](#)) request: in fact, any SIP dialog, be it a new or an existing one, can be exploited to carry leasing information, and as such new SIP INVITE messages can update other leases as well as requesting a new one.

With IAMM, the application server or media server will eventually send a SIP BYE to end the SIP session, whether it was for a control channel or a media dialog. That BYE contains no Consumer interface

lease information.

5.2.4. <mrbcconsumer>

This section defines the XML elements for the Consumer interface. The formal XML schema definition for the Consumer interface can be found in [Section 11](#).

The root element is <mrbcconsumer>. All other XML elements (requests, responses) are contained within it. The MRB Consumer interface request element is detailed in [Section 5.2.5.1](#). MRB Consumer interface response element is contained in [Section 5.2.6.1](#).

The <mrbcconsumer> element has the following attributes:

version: a token specifying the mrbc-consumer package version. The value is fixed as '1.0' for this version of the package. The attribute MUST be present.

The <mrbcconsumer> element may have zero or more children of one of the following child element types:

<mediaResourceRequest> for sending a Consumer request. See [Section 5.2.5.1](#).

<mediaResourceResponse> for sending a Consumer response. See [Section 5.2.6.1](#).

5.2.5. Media Service Resource Request

This section provides the element definitions for use in Consumer interface requests. The requests are carried in the <mediaResourceRequest> element.

5.2.5.1. <mediaResourceRequest> element

The <mediaResourceRequest> element provides information for clients wishing to query an external MRB entity. The <mediaResourceRequest> element has a single mandatory attribute, 'id': this attribute contains a random identifier, generated by the client, which will be included in the response in order to map it to a specific request. The <mediaResourceRequest> element has <generalInfo>, <ivrInfo> and <mixerInfo> as child elements. These three elements are used to describe the requirements of a client requesting a Media Server and are covered in the following sub-sections.

[5.2.5.1.1](#). `<generalInfo>` element

The `<generalInfo>` element provides a information for general Consumer request information that is neither IVR or Mixer specific. This includes session information that can be used for subsequent requests as part of the leasing mechanism described in [Section 5.2.3](#). The following sub-sections describe the elements of the `<generalInfo>` element, `<session-info>` and `<packages>`.

[5.2.5.1.1.1](#). `<session-info>` element

The `<session-info>` element is included in Consumer requests when an update is being made to an existing media resource session. The ability to change and remove an existing media resource session is described in more detail in [Section 5.2.3](#). The element MAY be present.

The `<session-info>` element has no attributes.

The `<session-info>` element has zero or more of the following child elements:

`<session-id>`: is a unique identifier that explicitly references an existing media resource session on the MRB. The identifier is included to update the existing session and is described in more detail in [Section 5.2.3](#).

`<seq>`: is used in association with the `<session-id>` element in a subsequent request to update an existing media resource session on an MRB. The `<seq>` number is incremented from its original value returned in response to the initial request for media resources. Its value is a non-negative integer, which MUST be limited within the $0..2^{31}-1$ range. In the unlikely case the counter increases to reach the highest allowed value, the `<seq>` value MUST be set to 0. More information about its use is provided in [Section 5.2.3](#).

`<action>`: provides the operation that should be carried out on an existing media resource session on an MRB:

- * The value of 'update' instructs the MRB to attempt to update the existing media resource session with the information contained in the `<ivrInfo>` and `<mixerInfo>` elements.
- * The value of 'remove' instructs the MRB to attempt to remove the existing media resource session. More information on its use is provided in [Section 5.2.3](#).

5.2.5.1.1.2. <packages> element

The <packages> element provides a list of Media Control Channel Framework compliant packages that are required by the Consumer client. The element MAY be present.

The <packages> element has no attributes.

The <packages> element has zero or more of the following child element:

<package>: child element contains a string representing the Media Control Channel Framework package required by the Consumer client. The <package> element can appear multiple times. A valid value is a Control Package name compliant with the [Section 13.1.1 of \[RFC6230\]](#).

5.2.5.1.2. <ivrInfo> element

The <ivrInfo> element provides information for general Consumer request information that is IVR specific. The following sub-sections describe the elements of the <ivrInfo> element, <ivr-sessions>, <file-formats>, <dtmf>, <tones>, <asr-tts>, <vxml>, <location>, <encryption>, <application-data>, <max-prepared-duration> and <stream-mode>.

5.2.5.1.2.1. <ivr-sessions> element

The <ivr-sessions> element indicates the number of IVR sessions a Consumer client requires from a media resource. The element MAY be present.

The <ivr-sessions> element has no attributes.

The <ivr-sessions> element has zero or more of the following child element:

<rtp-codec>: Describes a required codec and the number of sessions using that codec. The <rtp-codec> element has one attribute. The value of the attribute 'name' is a media type (which can include parameters per [\[RFC6381\]](#)). The <rtp-codec> element has two child elements. The child element, <decoding>, contains the number of RTP sessions required for decoding using the specified codec. The child element, <encoding>, contains the number of RTP sessions required for encoding using the specified codec.

5.2.5.1.2.2. <file-formats> element

The <file-formats> element provides a list of file formats required for the purpose of playing media. It should be noted that this element describes media types, and might better have been named "media-format" but the name "file-format" is being used due to existing implementations. The element MAY be present.

The <file-formats> element has no attributes.

The <file-formats> element has zero or more of the following child element:

<required-format>: has a single attribute, 'name', which provides the type of file format that is required. A valid value is a media type which, depending on its definition, can include additional parameters (e.g., [[RFC6381](#)]). The <required-format> element then has a further child element, <required-file-package>. The <required-file-package> element has a single attribute, 'required-file-package-name', which contains the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the file format support applies.

5.2.5.1.2.3. <dtmf> element

The <dtmf> element specifies the required methods to detect DTMF tones and to generate them. The element MAY be present.

The <dtmf> element has no attributes.

The <dtmf> element has zero or more of the following child elements:

<detect>: Indicates the required support for DTMF detection. The <detect> element has no attributes. The <detect> element has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF required, and is a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (detecting tones as signals from the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

<generate>: Indicates the required support for DTMF generation. The <generate> element has no attributes. The <generate> element has a single child element, <dtmf-type>. The <dtmf-type> element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF required, and is a case insensitive

string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (generating tones as signals in the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

<passthrough>: Indicates the required support for passing DTMF through without re-encoding. The <passthrough> element has no attributes. The <passthrough> element then has a further child element, <dtmf-type>. The <dtmf-type> element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF required, and is a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (passing tones as signals through the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

[5.2.5.1.2.4](#). <tones>

The <tones> element provides requested tones a media server must support for IVR. In particular, the request refers to both country codes support (ISO 3166-1 [[ISO.3166-1](#)]) and requested functionality (ITU-T Recommendation Q.1950 [[ITU-T.Q.1950](#)]). The element MAY be present.

The <tones> element has no attributes.

The <tones> element has zero or more of the following child elements:

<country-codes>: Describes the requested country codes in relation to tones. The <country-codes> element has no attributes. The <country-codes> has one child element. The child element, <country-code>, requests a specific country code, compliant with the ISO 3166-1 [[ISO.3166-1](#)] specification. The <country-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), in which the tones from the specified country code are requested.

<h248-codes>: Describes the requested H.248 codes in relation to tones. The <h248-codes> element has no attributes. The <h248-codes> has one child element. The child element, <h248-code>, requests a specific H.248 code, compliant with the ITU-T Recommendation Q.1950 [[ITU-T.Q.1950](#)] specification. The codes can be either specific (e.g., cg/dt to only report the Dial Tone from the Call Progress Tones package) or generic (e.g., cg/* to report all the tones from the Call Progress Tones package) using wild-cards. The <h248-code> element has a single attribute, 'package'.

The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), in which the specified codes are requested.

5.2.5.1.2.5. <asr-tts>

The <asr-tts> element requests information about the support for Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) functionality in a media server. The functionality is requested by referring to the supported languages (using ISO-639-1 [\[ISO.639.1988\]](#) codes) in relation to both ASR and TTS. The <asr-tts> element has no attributes. The <asr-tts> element has zero or more of the following child elements:

<asr-support>: Describes the available languages for ASR. The <asr-support> element has no attributes. The <asr-support> has one child element. The child element, <language>, requests the Media Server supports ASR for a specific language. The <language> element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO-639-1 [\[ISO.639.1988\]](#) code of the supported language.

<tts-support>: Describes the available languages for TTS. The <tts-support> element has no attributes. The <tts-support> has one child element. The child element, <language>, requests the Media Server supports tts for a specific language. The <language> element has a single attribute, 'xml:lang'. The attribute 'xml:lang' contains the ISO-639-1 [\[ISO.639.1988\]](#) code of the supported language.

5.2.5.1.2.6. <vxml> element

The <vxml> element specifies if the Consumer client requires VoiceXML and if so the supported protocols (e.g., via the control framework, [RFC4240 \[RFC4240\]](#), or [RFC5552 \[RFC5552\]](#)). The element MAY be present.

The <vxml> element has zero or more of the following child elements:

<vxml-mode>: has two attributes, 'package' and 'require'. The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the VXML support applies. The 'require' attribute specifies the type of VXML support required by the Consumer client (e.g., [RFC5552 \[RFC5552\]](#), [RFC4240 \[RFC4240\]](#) or IVR Package [\[RFC6231\]](#)), and valid values are case insensitive RFC references (e.g., "[rfc6231](#)" to specify the Client requests support for VoiceXML as provided by the IVR Package [\[RFC6231\]](#)).

The presence of at least one <vxml> child element would indicate that the Consumer client requires VXML support as specified by the child element itself. An empty <vxml> element would otherwise indicate the Consumer client does not require VXML support.

5.2.5.1.2.7. <location>

The <location> element requests a civic location for an IVR media server. The request makes use of the Civic Address Schema standardized in [RFC 5139](#) [RFC5139]. The element MAY be present. More precisely, this section is entirely optional, and is implementation specific in its level of population.

The <location> element has no attributes.

The <location> element has a single child element:

<civicAddress>: Describes the civic address location of the requested media server, whose representation refers to [Section 4 of RFC 5139](#) [RFC5139].

5.2.5.1.2.8. <encryption>

The <encryption> element allows a Consumer client to request support for encrypting RTP media streams using [RFC 3711](#) [RFC3711]. The element MAY be present.

The <encryption> element has no attributes.

The <encryption> element has zero or more of the following child elements:

<keying-mechanism>: has no attributes. The element provides the name of a keying mechanism the Client requires for encrypting RTP media streams.

The presence of at least one <keying-mechanism> child element would indicate that the Consumer client does request RTP media stream encryption as specified by the child element itself. An empty <encryption> element would otherwise indicate the Consumer client does not require RTP encryption at all.

5.2.5.1.2.9. <application-data>

The <application-data> element provides an arbitrary string of characters as IVR application level data. This data is meant to only have meaning at the application level logic and as such is not otherwise restricted by this specification. The set of allowed

characters are the same as those in XML (viz., tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [see <http://www.w3.org/TR/xml/section 2.2>]). The element MAY be present.

The <application-data> element has no attributes.

The <application-data> element has no child elements.

5.2.5.1.2.10. <max-prepared-duration>

The <max-prepared-duration> element indicates the amount of time required by the Consumer client representing media dialog preparation in the system before it is executed. The element MAY be present.

The <max-prepared-duration> element has no attributes.

The <max-prepared-duration> element has a single child element:

<max-time>: has a single attribute, 'max-time-seconds', which provides the amount of time in seconds that a media dialog can be in the prepared state. The <max-time> element then has a further child element, <max-time-package>. The <max-time-package> element provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the time period applies.

5.2.5.1.2.11. <file-transfer-modes>

The <file-transfer-modes> element allows the Consumer client to specify which scheme names are required for file transfer to a Media Server for each Media Control Channel Framework package type. For example, does the Media Server support fetching media resources via RTSP, HTTP, NFS, etc protocols. The element MAY be present.

The <file-transfer-modes> element has no attributes.

The <file-transfer-modes> element has a single child element:

<file-transfer-mode>: has two attributes, 'name' and 'package'. The 'name' attribute provides the scheme name of the protocol required for fetching resources: valid values are case insensitive scheme names (e.g., RTSP, HTTP, HTTPS, NFS, etc.). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the scheme name applies.

The same considerations relating to file transfer and live streaming are explained further in [Section 5.1.5.15](#), which apply here as well.

[5.2.5.1.3.](#) **<mixerInfo> element**

The <mixerInfo> element provides information for general Consumer request information that is Mixer specific. The following sub-sections describe the elements of the <mixerInfo> element, <mixers>, <file-formats>, <dtmf-type>, <tones>, <mixing-mode>, <application-data>, <location> and <encryption>.

[5.2.5.1.3.1.](#) **<mixers>**

The <mixers> element provides information detailing the required mixed RTP sessions. The element MAY be present.

The <mixers> element has no attributes.

The <mixers> element has a single child element:

<mix>: Describes the required mixed RTP sessions. The <mix> element has one attribute. The value of the attribute, 'users', is the number of participants required in the mix. The <mix> element has one child element. The child element, <rtp-codec>, contains the same information relating to RTP sessions as defined in [Section 5.1.5.3](#). The element MAY be present.

[5.2.5.1.3.2.](#) **<file-formats>**

The <file-formats> element provides a list of file formats required by the Consumer client for the purpose of playing media to a mix. The element MAY be present.

The <file-formats> element has no attributes.

The <file-formats> element has a single child element:

<required-format>: has a single attribute, 'name', which provides the type of file format supported. A valid value is a media type which, depending on its definition, can include additional parameters (e.g., [\[RFC6381\]](#)). The <required-format> element has a child element, <required-file-package>. The <required-file-package> element contains a single attribute, 'required-file-package-name', which contains the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the file format support applies.

[5.2.5.1.3.3.](#) `<dtmf>` element

The `<dtmf>` element specifies the required methods to detect DTMF tones and to generate them in a mix. The element MAY be present.

The `<dtmf>` element has no attributes.

The `<dtmf>` element has zero or more of the following child elements:

`<detect>`: Indicates the required support for DTMF detection. The `<detect>` element has no attributes. The `<detect>` element then has a further child element, `<dtmf-type>`. The `<dtmf-type>` element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and it is a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (detecting tones as signals from the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

`<generate>`: Indicates the required support for DTMF generation. The `<generate>` element has no attributes. The `<generate>` element has a single child element, `<dtmf-type>`. The `<dtmf-type>` element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and is a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (generating tones as signals in the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

`<passthrough>`: Indicates the required support for passing DTMF through without re-encoding. The `<passthrough>` element has no attributes. The `<passthrough>` element has a single child element, `<dtmf-type>`. The `<dtmf-type>` element has two attributes, 'name' and 'package'. The 'name' attribute provides the type of DTMF being used, and is a case insensitive string containing either '[RFC4733](#)' [[RFC4733](#)] or 'Media' (passing tones as signals through the audio stream). The 'package' attribute provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the DTMF type applies.

[5.2.5.1.3.4.](#) `<tones>`

The `<tones>` element provides requested tones a media server must support for a mix. In particular, the request refers to both country codes support (ISO 3166-1 [[ISO.3166-1](#)]) and requested functionality (ITU-T Recommendation Q.1950 [[ITU-T.Q.1950](#)]). The element MAY be

present.

The <tones> element has no attributes.

The <tones> element has zero or more of the following child elements:

<country-codes>: Describes the requested country codes in relation to tones. The <country-codes> element has no attributes. The <country-codes> has a single child element. The child element, <country-code>, requests a specific country code, compliant with the ISO 3166-1 [[ISO.3166-1](#)] specification. The <country-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the specification in the related IANA registry (e.g., "msc-ivr/1.0"), in which the tones from the specified country code are requested.

<h248-codes>: Describes the requested H.248 codes with respect to tones. The <h248-codes> element has no attributes. The <h248-codes> has a single child element. The child element, <h248-code>, requests a specific H.248 code, compliant with the ITU-T Recommendation Q.1950 [[ITU-T.Q.1950](#)] specification. The codes can be either specific (e.g., cg/dt to only report the Dial Tone from the Call Progress Tones package) or generic (e.g., cg/* to report all the tones from the Call Progress Tones package) using wild-cards. The <h248-code> element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), in which the specified codes are requested.

[5.2.5.1.3.5](#). <mixing-modes>

The <mixing-modes> element requests information relating to support for audio and video mixing; more specifically a list of supported algorithms to mix audio and a list of supported video presentation layouts. The element MAY be present.

The <mixing-modes> element has no attributes.

The <mixing-modes> element has zero or more of the following child elements:

<audio-mixing-modes>: Describes the requested algorithms for audio mixing. The <audio-mixing-modes> element has no attributes. The <audio-mixing-modes> element has one child element. The child element, <audio-mixing-mode>, contains a requested mixing algorithm. Valid values for the <audio-mixing-mode> element are algorithm names, e.g., 'nbest' and 'controller' as defined in

[[RFC6505](#)]. The element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the algorithm support is requested.

<video-mixing-modes>: Describes the requested video presentation layouts for video mixing. The **<video-mixing-modes>** element has two attributes, 'vas' and 'activespeakermix'. The 'vas' attribute is of type boolean with a value of 'true' indicating that the Consumer Client requires automatic Voice Activated Switching. The 'activespeakermix' attribute is of type boolean with a value of 'true' indicating that the Consumer Client requires an additional video stream for the loudest speaker participant without its contribution. The **<video-mixing-modes>** element has one child element. The child element, **<video-mixing-mode>**, contains the name of a specific video presentation layout. The name may refer to one of predefined video layouts defined in the XCON conference information data model, or to non-XCON layouts as well, as long as they are appropriately prefixed. The **<video-mixing-mode>** element has a single attribute, 'package'. The attribute 'package' provides the name of the Media Control Channel Framework package, compliant with the [Section 13.1.1 of \[RFC6230\]](#), for which the algorithm support is requested.

[5.2.5.1.3.6](#). <application-data>

The **<application-data>** element provides an arbitrary string of characters as Mixer application level data. This data is meant to only have meaning at the application level logic and as such is not otherwise restricted by this specification. The set of allowed characters are the same as those in XML (viz., tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646 [see <http://www.w3.org/TR/xml/> [section 2.2](#)]). The element MAY be present.

The **<application-data>** element has no attributes.

The **<application-data>** element has no child elements.

[5.2.5.1.3.7](#). <location>

The **<location>** element requests a civic location for a mixer media server. The request makes use of the Civic Address Schema standardized in [RFC 5139](#) [[RFC5139](#)]. The element MAY be present. More precisely, this section is entirely optional, and it's implementation specific to fill it with just the details each implementor deems necessary for any optimization that may be needed.

The contents of a **<location>** element has no attributes.

The contents of a <location> element has a single child element:

<civicAddress>: Describes the civic address location of the requested media server, whose representation refers to [Section 4 of RFC 5139](#) [[RFC5139](#)].

[5.2.5.1.3.8.](#) <encryption>

The <encryption> element allows a Consumer client to request support for encrypting mixed RTP media streams using [RFC 3711](#) [[RFC3711](#)]. The element MAY be present.

The <encryption> element has no attributes.

The <encryption> element has zero or more of the following child elements:

<keying-mechanism>: has no attributes. The element provides the name of a keying mechanism the Consumer client requires for encrypting mixed RTP media streams.

The presence of at least one <keying-mechanism> child element would indicate that the Consumer client does require mixed RTP media stream encryption as specified by the child element itself. An empty <encryption> element would otherwise indicate the client does not require RTP encryption at all.

[5.2.6.](#) Media Service Resource Response

This section provides the element definitions for use in Consumer interface responses. The responses are carried in the <mediaResourceResponse> element.

[5.2.6.1.](#) <mediaResourceResponse> element

The <mediaResourceResponse> element provides information for clients receiving response information from an external MRB entity.

The <mediaResourceResponse> element has two mandatory attributes, 'id' and 'status'. The 'id' attribute must contain the same value the client provided in the 'id' attribute in the <mediaResourceRequest> the response is for. The 'status' attribute indicates the status code of the operation. The following status codes are defined for 'status':

code	description
200	OK
400	Syntax error
405	Wrong sequence number
408	Unable to find Resource
409	Unable to update Resource
410	Unable to remove Resource
420	Unsupported attribute or element

Table 2: <response> status codes

In case a new media resource request made by an client application has been accepted, the MRB MUST reply with a <mediaResourceResponse> with status code 200. The same rule applies whenever a request to update (action='update') or remove (action='remove') an existing transaction can be fulfilled by the MRB.

A media resource request, nevertheless, may fail for several reasons. In such a case, the status codes defined in Table 2 must be used instead. Specifically, if the MRB fails to handle a request due to a syntax error in the request itself (e.g., incorrect XML, violation of the schema constraints or invalid values in any of the attributes/elements) the MRB MUST reply with a <mediaResourceResponse> with status code 400. If a syntactically correct request fails because the request also includes any attribute/element the MRB doesn't understand, the MRB MUST reply with a <mediaResourceResponse> with status code 420. If a syntactically correct request fails because it contains a wrong sequence number, that is, a 'seq' value not consistent with the increment the MRB expects according to [Section 5.2.3](#), the MRB MUST reply with a <mediaResourceResponse> with status code 405. If a syntactically correct request fails because the MRB couldn't find any Media Server able to fulfil the requirements presented by the Application Server in its request, the MRB MUST reply with a <mediaResourceResponse> with status code 408. If a syntactically correct request fails because the MRB couldn't update an existing request according to the new requirements presented by the Application Server in its request, the MRB MUST reply with a <mediaResourceResponse> with status code 409. If a syntactically correct request fails because the MRB couldn't remove

an existing request and release the related resources as requested by the Application Server, the MRB MUST reply with a `<mediaResourceResponse>` with status code 410.

Further details on status codes 409 and 410 are included in [Section 5.2.3](#) where the leasing mechanism, along with its related scenarios, are described in more detail.

The `<mediaResourceResponse>` element has `<response-session-info>` as a child element. This element is used to describe the response of a Consumer interface query and is covered in the following sub-section.

[5.2.6.1.1](#). `<response-session-info>` element

The `<response-session-info>` element is included in Consumer responses. This applies to responses to both requests for new resources and requests to update an existing media resource session. The ability to change and remove an existing media resource session is described in more detail in [Section 5.2.3](#). If the request was successful, the `<mediaResourceResponse>` MUST have one `<response-session-info>` child, which describes the media resource session addressed by the request. If the request was not successful, the `<mediaResourceResponse>` MUST NOT have a `<response-session-info>` child.

The contents of a `<response-session-info>` element has no attributes.

The contents of a `<response-session-info>` element has zero or more of the following child elements:

`<session-id>`: is a unique identifier that explicitly references an existing media resource session on the MRB. The identifier is included to update the existing session and is described in more detail in [Section 5.2.3](#).

`<seq>`: is used in association with the `<session-id>` element in a subsequent request to update an existing media resource session on an MRB. The `<seq>` number is incremented from its original value returned in response to the initial request for media resources. More information its use is provided in [Section 5.2.3](#).

`<expires>`: includes the number of seconds that the media resources are reserved as part of this interaction. If the lease is not refreshed before expiry, the MRB will re-claim the resources and they will no longer be guaranteed. It is RECOMMENDED that a minimum value of 300 seconds be used for the value of the 'expires' attribute. It is also RECOMMENDED that a Consumer client refresh the lease at an interval that is not too close to

the expiry time. A value of 80% of the time-out period could be used. For example, if the time-out period is 300 seconds, the Consumer Client would refresh the transaction at 240 seconds. More information on its use is provided in [Section 5.2.3](#).

`<media-server-address>`: provides information to reach the Media Server handling the requested media resource. One or more instances of these element may appear. The `<media-server-address>` element has a single attribute named 'uri' which supplies a SIP URI that reaches the specified media server. It also has three optional elements `<connection-id>`, `<ivr-sessions>`, and `<mixers>`. The `<ivr-sessions>` and `<mixers>` are defined in [Section 5.2.5.1.2.1](#) and [Section 5.2.5.1.3.1](#) and have the same meaning but are applied to individual media server instances as a subset of the overall resources reported in the `<connection-id>` element. If multiple Media Servers are assigned in an IAMM operation, exactly one `<media-server-address>` element, more specifically the media server that provided the media dialog or CFW response, will have a `<connection-id>` element. Additional information relating to the use of the `<connection-id>` element for media dialogs is included in [Section 6](#).

5.3. In-Line Unaware MRB Interface

An entity acting as an In-Line MRB can act in one of two roles for a request, as introduced in [Section 4.2](#). In-Line Unaware MRB Mode (IUMM) of operation and In-Line Aware MRB Mode (IAMM) of operation. This section further describes IUMM.

It should be noted that the introduction of an MRB entity into the network, as specified in this document, requires interfaces to be implemented by those requesting Media Server resources (for example an Application Server). This applies when using the Consumer interface as discussed in [Section 5.2.1](#) (Query mode) and [Section 5.2.2](#) (IAMM). An MRB entity can also act in a client unaware mode when deployed into the network. This allows any SIP compliant client entity, as defined by [RFC 3261](#) [[RFC3261](#)] and its extensions, to send requests to an MRB which in turn will select an appropriate media server based on knowledge of media server resources it currently has available transparently to the client entity. Using an MRB in this mode allows for easy migration of current applications and services that are unaware of the MRB concept and would simply require a configuration change resulting in the MRB being set as a SIP outbound proxy for clients requiring media services.

With IUMM, the MRB may conclude that an assigned media resource is no longer needed when it receives a SIP BYE from the application server or media server that ends that SIP dialog that initiated the request.

As with IAMM, in IUMM the SIP INVITE from the Application Server could convey application/sdp payload to either set up a media dialog or a Control Framework control channel. In either case, in order to permit the Application Server to associate a media dialog with a control channel to the same Media Server, using the procedures of [\[RFC6230\] section 6](#), the MRB should be acting as a SIP proxy (and not a B2BUA). This allows the SIP URI of the targeted Media Server to be transparently passed back to the Application Server in the SIP response resulting in a SIP dialog directly between the Application Server and the Media Server.

While IUMM has the least impact on legacy application servers, it also provides the least versatility. See [Section 8](#).

6. MRB acting as a B2BUA

An MRB entity can act as a SIP Back-2-Back-User-Agent (B2BUA) or a SIP Proxy Server as defined in [RFC 3261](#) [[RFC3261](#)]. When acting as a B2BUA issues can arise when using Media Control Channel packages such as the IVR[RFC6231] and Mixer[RFC6505] Packages. Specifically the Framework attribute 'connectionid' provided in the appendix titled 'Appendix: Common Package Components' of Media Control Channel Framework[RFC6230] uses a concatenation of the SIP dialog identifiers to be used for referencing SIP dialogs within the media control channel. When a request traverses an MRB acting as a B2BUA, the SIP dialog identifiers change and so the 'connectionid' can not be used as intended due to the SIP dialog identifiers changing. For this reason when a MRB wishes to act as a SIP B2BUA when handling a request from an Application Server to set up a media dialog to a Media Server it MUST include the optional <connection-id> element in a Consumer interface response with a value that provides the equivalent for the 'connectionid' ('Local Dialog Tag' + 'Remote Dialog Tag') for the far side of the B2BUA. If present, this value MUST be used as the value for the 'connectionid' in packages where the Common Package Components are used. The <connection-id> element MUST NOT be included in a HTTP Consumer interface response.

It is important to point out that, although more Media Server instances may be returned in a Consumer response (i.e., the MRB has assigned more than one Media Server to a Consumer request to fulfill the Application Server requirements), in IAMM the MRB will only act as a B2BUA with a single Media Server. In this case exactly one <media-server-address> element, describing the media dialog or CFW response, will have a <connection-id> element which will not be included in any additional <media-server-address> elements.

7. Multi-modal MRB Implementations

An MRB implementation may operate multi-modally with a collection of Application Server clients all sharing the same pool of media resources. I.e., an MRB may be simultaneously operating in Query mode, IAMM and IUMM. It knows in which mode to act on any particular request from a client depending on the context of the request:

- o If the received request is a HTTP POST message with application/mrb-consumer+xml content, then MRB processes it in Query mode.
- o If the received request is a SIP INVITE with application/mrb-consumer+xml content and application/sdp content, then MRB processes it in IAMM.
- o If the received request is a SIP INVITE without application/mrb-consumer+xml content but with application/sdp content then MRB processes it in IUMM.

8. Relative Merits of Query Mode, IAMM, and IUMM

At a high level, the possible Application Server MRB interactions can be distinguished by the following basic types:

- a. Query mode - the client is requesting the assignment by MRB of suitable Media Server resources;
- b. IAMM - the client is requesting the assignment by MRB of suitable Media Server resources and the establishment of a media dialog to one of the Media Servers;
- c. IAMM - the client is requesting the assignment by MRB of suitable Media Server resources and the establishment of a CFW control channel to one of the Media Servers;
- d. IUMM - the client is requesting the establishment of a media dialog to a Media Server resource;
- e. IUMM - the client is requesting the establishment of a CFW control channel to a Media Server resource.

Each type of interaction has advantages and disadvantages, where such considerations related to the versatility of what MRB can provide, technical aspects such as efficiency in different application scenarios, complexity, delay, use with legacy Application Servers, or use with the Media Control Channel Framework. Depending on the characteristics of a particular setting that an MRB is intended to support, some of the above interaction types may be more appropriate than others. This section provides a few observations on relative merits, but is not intended to be exhaustive. Some constraints of a given interaction type may be subtle.

- o Operation with other types of media control: Any of the types of interactions work with the use [RFC 4240](#) [[RFC4240](#)] and [RFC 5552](#) [[RFC5552](#)] where initial control instructions are conveyed in the SIP INVITE from the Application Server for the media dialog to the Media Server and subsequent instructions may be fetched using HTTP. Query mode (a), IAMM/media dialog (b) and IUMM/media dialog (d) work with MSML as per [RFC 5707](#) [[RFC5707](#)] or MSCML as per [RFC 5022](#) [[RFC5022](#)].
- o As stated previously, IUMM has no interface impacts on an Application Server. When using IUMM the Application Server does not specify the characteristics of the type of media resource it requires as the <mediaResourceRequest> element is not passed to the MRB. For IUMM media dialog (d), the MRB can deduce an appropriate media resource on a best effort basis using

information gleaned from examining information in the SIP INVITE. This includes the SDP information for the media dialog, or initial control information in the SIP Request URI as per [RFC 4240](#) [[RFC4240](#)]. With IUMM/control channel (e) there is even less information for the MRB to use.

- o If using IUMM/control channel (e), the subsequent sending of the media dialog to the media server should not be done using IUMM/media dialog. I.e., the SIP signalling to send the media dialog to the selected media server must be directly between the Application Server and that media server, and not through the MRB. Unless resources can be confidentially identified, the MRB could send the media dialog to a different Media Server. Likewise, if using IUMM/media dialog (d), the subsequent establishment of a control channel should not be done with IUMM/control channel (e) unless definitive information is available.
- o Query mode (a) and IAMM/control channel (c) lend themselves to requesting a pool of media resources (e.g., a number of IVR or conferencing ports) in advance of use and retaining use over a period of time, independent of whether there are media dialogs to those resources at any given moment, whereas the other types of interactions do not. Likewise for making a subsequent request to increase or decrease the amount of resources previously awarded.
- o While Query mode (a) and IAMM/control channel (c) are the most versatile interaction types, the former is completely decoupled from the use or not of a control channel, whereas the latter requires the use of a control channel.
- o When Media Control Channel Framework control channels are to be used in conjunction with the use of MRB, Query mode (a) would typically result in fewer such channels being established over time as compared to IAMM/control channel (c). That is because the latter would involve setting up an additional control channel every time an Application Server has a new request for MRB for media resources.

9. Examples

This section provides examples of both the Publish and Consumer interfaces. Both Query and Inline modes are addressed.

Note that due to RFC formatting conventions, this section often splits HTTP, SIP/SDP and CFW across lines whose content would exceed 72 characters. A backslash character marks where this line folding has taken place. This backslash and its trailing CRLF and whitespace would not appear in the actual protocol contents. Besides, also note that the indentation of the XML content is only provided for readability: actual messages will follow strict XML syntax, which allows for, but does not require, indentation.

9.1. Publish Example

The following example assumes a control channel has been established and synced as described in the Media Control Channel Framework ([RFC6230]).

Figure 9 shows the subscription/notification mechanism the Publish interface is based on, as defined in [Section 5.1](#). The MRB subscribes for information at the Media Server (message A1.), and the Media Server accepts the subscription (A2). Notifications are triggered by the Media Server (A3.) and acknowledged by the MRB (A4.).

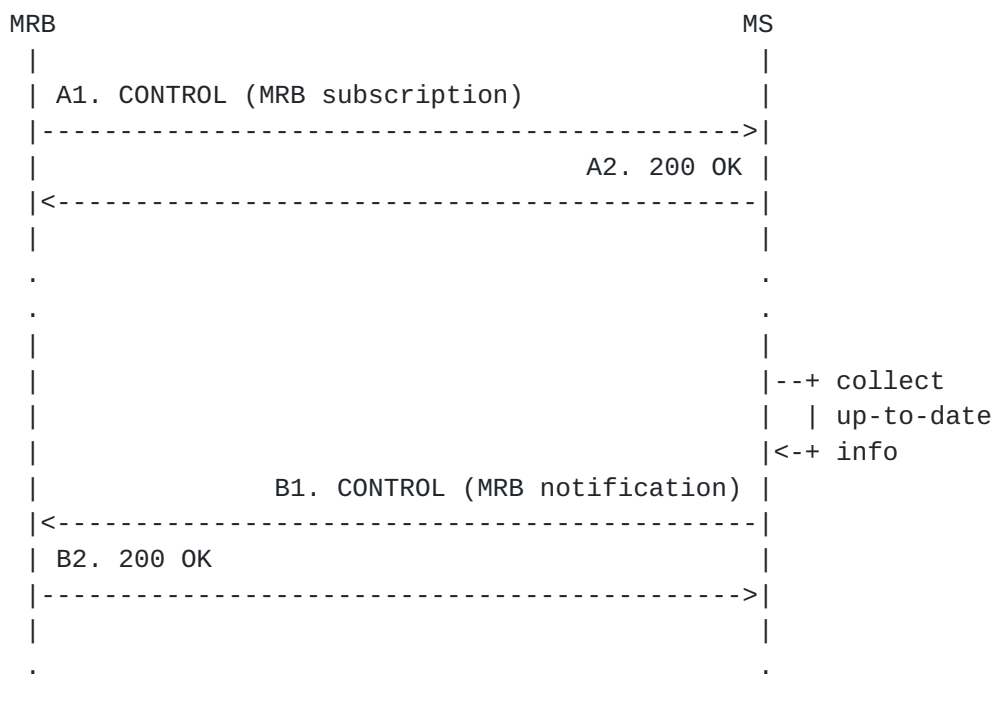


Figure 9: Publish Example: Sequence Diagram

The rest of this section includes a full dump of the messages associated with the previous sequence diagram, specifically:

1. the subscription (A1), in an <mrbrequest> (CFW CONTROL);
2. the Media Server accepting the subscription (A2), in an <mrbresponse> (CFW 200);
3. a notification (A3), in a <mrnotification> (CFW CONTROL event);
4. the ack to the notification (A4), in a framework level 200 message (CFW 200);

A1. MRB -> MS (CONTROL, publish request)

CFW lidc30BZ0biC CONTROL
Control-Package: mrb-publish/1.0
Content-Type: application/mrb-publish+xml
Content-Length: 337

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-publish">
  <mrbrequest>
    <subscription action="create" seqnumber="1" id="p0T65U">
      <expires>600</expires>
      <minfrequency>20</minfrequency>
      <maxfrequency>20</maxfrequency>
    </subscription>
  </mrbrequest>
</mrbpublish>
```

A2. MRB <- MS (200 to CONTROL, request accepted)

CFW lidc30BZ0biC 200
Timeout: 10
Content-Type: application/mrb-publish+xml
Content-Length: 139

```
<mrbpublish version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-publish">
  <mrbresponse status="200" reason="OK: Request accepted"/>
</mrbpublish>
```


B1. MRB <- MS (CONTROL, event notification from MS)

CFW 03fff52e7b7a CONTROL

Control-Package: mrb-publish/1.0

Content-Type: application/mrb-publish+xml

Content-Length: 4226

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <mrbpublish version="1.0"
    xmlns="urn:ietf:params:xml:ns:mrb-publish">
    <mrbnotification seqnumber="1" id="QQ6J3c">
      <media-server-id>a1b2c3d4</media-server-id>
      <supported-packages>
        <package name="msc-ivr/1.0"/>
        <package name="msc-mixer/1.0"/>
        <package name="mrb-publish/1.0"/>
        <package name="msc-example-pkg/1.0"/>
      </supported-packages>
      <active-rtp-sessions>
        <rtp-codec name="audio/basic">
          <decoding>10</decoding>
          <encoding>20</encoding>
        </rtp-codec>
      </active-rtp-sessions>
      <active-mixer-sessions>
        <active-mix conferenceid="7cfigs43">
          <rtp-codec name="audio/basic">
            <decoding>3</decoding>
            <encoding>3</encoding>
          </rtp-codec>
        </active-mix>
      </active-mixer-sessions>
      <non-active-rtp-sessions>
        <rtp-codec name="audio/basic">
          <decoding>50</decoding>
          <encoding>40</encoding>
        </rtp-codec>
      </non-active-rtp-sessions>
      <non-active-mixer-sessions>
        <non-active-mix available="15">
          <rtp-codec name="audio/basic">
            <decoding>15</decoding>
            <encoding>15</encoding>
          </rtp-codec>
        </non-active-mix>
      </non-active-mixer-sessions>
      <media-server-status>active</media-server-status>
      <supported-codecs>
```



```
<supported-codec name="audio/basic">
  <supported-codec-package name="msc-ivr/1.0">
    <supported-action>encoding</supported-action>
    <supported-action>decoding</supported-action>
  </supported-codec-package>
  <supported-codec-package name="msc-mixer/1.0">
    <supported-action>encoding</supported-action>
    <supported-action>decoding</supported-action>
  </supported-codec-package>
</supported-codec>
</supported-codecs>
<application-data>TestbedPrototype</application-data>
<file-formats>
  <supported-format name="audio/x-wav">
    <supported-file-package>
      msc-ivr/1.0
    </supported-file-package>
  </supported-format>
</file-formats>
<max-prepared-duration>
  <max-time max-time-seconds="3600">
    <max-time-package>msc-ivr/1.0</max-time-package>
  </max-time>
</max-prepared-duration>
<dtmf-support>
  <detect>
    <dtmf-type package="msc-ivr/1.0" name="RFC4733"/>
    <dtmf-type package="msc-mixer/1.0" name="RFC4733"/>
  </detect>
  <generate>
    <dtmf-type package="msc-ivr/1.0" name="RFC4733"/>
    <dtmf-type package="msc-mixer/1.0" name="RFC4733"/>
  </generate>
  <passthrough>
    <dtmf-type package="msc-ivr/1.0" name="RFC4733"/>
    <dtmf-type package="msc-mixer/1.0" name="RFC4733"/>
  </passthrough>
</dtmf-support>
<mixing-modes>
  <audio-mixing-modes>
    <audio-mixing-mode package="msc-ivr/1.0">
      nbest
    </audio-mixing-mode>
  </audio-mixing-modes>
  <video-mixing-modes activespeakermix="true" vas="true">
    <video-mixing-mode package="msc-mixer/1.0">
      single-view
    </video-mixing-mode>
```



```
<video-mixing-mode package="msc-mixer/1.0">
  dual-view
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  dual-view-crop
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  dual-view-2x1
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  dual-view-2x1-crop
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  quad-view
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  multiple-5x1
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  multiple-3x3
</video-mixing-mode>
<video-mixing-mode package="msc-mixer/1.0">
  multiple-4x4
</video-mixing-mode>
</video-mixing-modes>
</mixing-modes>
<supported-tones>
  <supported-country-codes>
    <country-code package="msc-ivr/1.0">GB</country-code>
    <country-code package="msc-ivr/1.0">IT</country-code>
    <country-code package="msc-ivr/1.0">US</country-code>
  </supported-country-codes>
  <supported-h248-codes>
    <h248-code package="msc-ivr/1.0">cg/*</h248-code>
    <h248-code package="msc-ivr/1.0">biztn/ofque</h248-code>
    <h248-code package="msc-ivr/1.0">biztn/erwt</h248-code>
    <h248-code package="msc-mixer/1.0">conftn/*</h248-code>
  </supported-h248-codes>
</supported-tones>
<file-transfer-modes>
  <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
</file-transfer-modes>
<asr-tts-support>
  <asr-support>
    <language xml:lang="en"/>
  </asr-support>
  <tts-support>
    <language xml:lang="en"/>
```



```

        </tts-support>
    </asr-tts-support>
    <vxml-support>
        <vxml-mode package="msc-ivr/1.0" support="RFC6231"/>
    </vxml-support>
    <media-server-location>
        <civicAddress xml:lang="it">
            <country>IT</country>
            <A1>Campania</A1>
            <A3>Napoli</A3>
            <A6>Via Claudio</A6>
            <HNO>21</HNO>
            <LMK>University of Napoli Federico II</LMK>
            <NAM>Dipartimento di Informatica e Sistemistica</NAM>
            <PC>80210</PC>
        </civicAddress>
    </media-server-location>
    <label>TestbedPrototype-01</label>
    <media-server-address>sip:MS1@ms.example.net</media-server-address>
    <encryption>
        <keying-mechanism>SDS-SRTP</keying-mechanism>
    </encryption>
</mrnotification>
</mrbpublish>

```

B2. MRB -> MS (200 to CONTROL)

CFW 03fff52e7b7a 200

9.2. Consumer Example

As specified in [Section 5.2](#), the Consumer interface can be involved in two different modes: Query and Inline-aware. When in Query mode, Consumer messages are transported in HTTP messages: an example of such an approach is presented in [Section 9.2.1](#). When in Inline-aware mode, instead, messages are transported as part of SIP negotiations: considering that SIP negotiations may be related to either the creation of a control channel or to a UAC media dialog, two separate examples of such an approach are presented in [Section 9.2.2](#).

9.2.1. Query Example

The following example assumes the interested Application Server already knows the HTTP URL where an MRB is listening for Consumer messages.

Figure 10 shows the HTTP-based transaction between the Application Server and the MRB. The Application Server sends a consumer request as payload of an HTTP POST message (1.), and the MRB provides an answer in an HTTP 200 OK message (2.). Specifically, as it will be shown in the examples, the Application Server is interested in 100 IVR ports: the MRB finds two Media Servers that can satisfy the request (one providing 60 ports, the other 40 ports) and reports them to the Application Server.

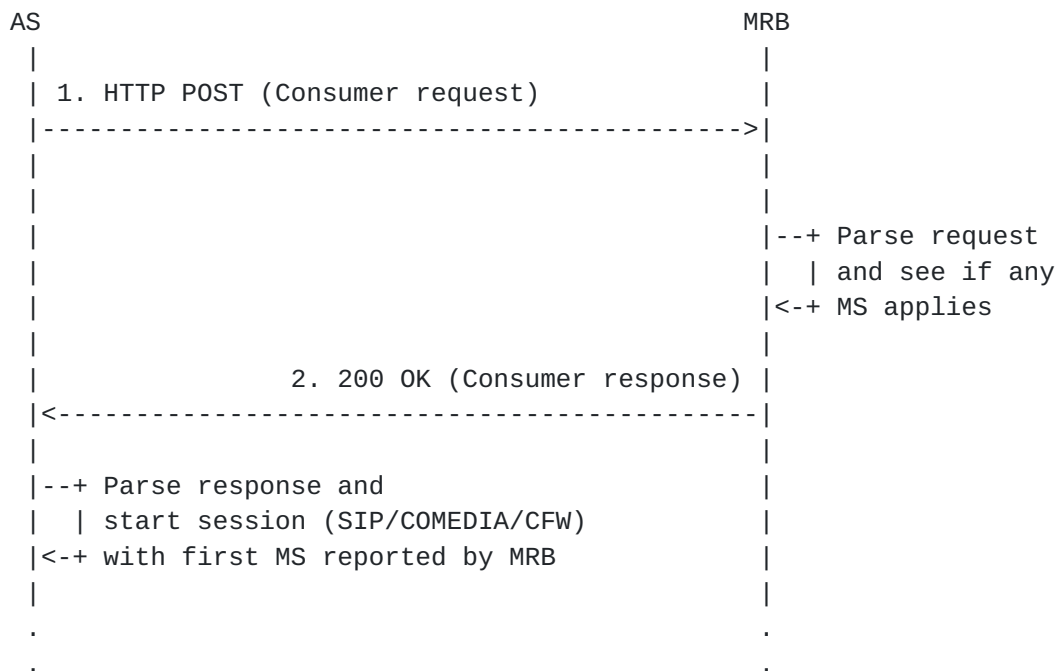


Figure 10: Consumer Example (Query): Sequence Diagram

The rest of this section includes a full dump of the messages associated with the previous sequence diagram, specifically:

1. the Consumer request (1), in a `<mediaResourceRequest>` (HTTP POST, Content-Type 'application/mrb-consumer+xml');
2. the Consumer response (2), in an `<mediaResourceResponse>` (HTTP 200 OK, Content-Type 'application/mrb-consumer+xml').

1. AS -> MRB (HTTP POST, Consumer request)

POST /Mrb/Consumer HTTP/1.1

Content-Length: 893
Content-Type: application/mrb-consumer+xml
Host: mrb.example.net:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.0.1 (java 1.5)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbconsumer version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-consumer">
  <mediaResourceRequest id="gh11x23v">
    <generalInfo>
      <packages>
        <package>msc-ivr/1.0</package>
        <package>msc-mixer/1.0</package>
      </packages>
    </generalInfo>
    <ivrInfo>
      <ivr-sessions>
        <rtp-codec name="audio/basic">
          <decoding>100</decoding>
          <encoding>100</encoding>
        </rtp-codec>
      </ivr-sessions>
      <file-formats>
        <required-format name="audio/x-wav"/>
      </file-formats>
      <file-transfer-modes>
        <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
      </file-transfer-modes>
    </ivrInfo>
  </mediaResourceRequest>
</mrbconsumer>
```

2. AS <- MRB (200 to POST, Consumer response)

```
-----
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun GlassFish Communications Server 1.5
Content-Type: application/mrb-consumer+xml; charset=ISO-8859-1
Content-Length: 1133
Date: Mon, 12 Apr 2011 14:59:26 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbconsumer version="1.0" xmlns="urn:ietf:params:xml:ns:mrb-consumer" >
  <mediaResourceResponse reason="Resource found" status="200"
    id="gh11x23v">
    <response-session-info>
      <session-id>5t3Y4IQ84gY1</session-id>
```



```
<seq>9</seq>
<expires>3600</expires>
<media-server-address
    uri="sip:MediaServer@ms.example.com:5080">
    <ivr-sessions>
        <rtp-codec name="audio/basic">
            <decoding>60</decoding>
            <encoding>60</encoding>
        </rtp-codec>
    </ivr-sessions>
</media-server-address>
<media-server-address
    uri="sip:OtherMediaServer@pool.example.net:5080">
    <ivr-sessions>
        <rtp-codec name="audio/basic">
            <decoding>40</decoding>
            <encoding>40</encoding>
        </rtp-codec>
    </ivr-sessions>
</media-server-address>
</response-session-info>
</mediaResourceResponse>
</mrbconsumer>
```

As the example show, the request and response are associated by means of the 'id' attribute (id="gh11x23v"). The MRB has picked '9' as the random sequence number that needs to be incremented by the Application Server for the following request associated with the same session.

The rest of the scenario is omitted for brevity. After having received the 'mediaResourceResponse', the Application Server has the URIs of two Media Servers able to fulfil its media requirements, and can start a Control Dialog with one or both of them.

[9.2.2.](#) IAMM Example

Two separate examples are presented for the IAMM case: in fact, IAMM-mode can take advantage of two different approaches with respect to the SIP dialogs to be exploited to carry consumer messages, i.e.: i) a SIP control dialog to create a control channel, and, ii) a UAC media dialog to attach to a Media Server. To make things clearer for the reader, the same consumer request as the one presented in the Query mode will be sent, in order to clarify how the behaviour of the involved parties may differ.

9.2.2.1. IAMM Example: CFW-based approach

The following example assumes the interested Application Server already knows the SIP URI for an MRB.

Figure 11 shows the first approach, i.e. SIP-based transactions between the Application Server, the MRB and one Media Server that the MRB chooses from the two that are allocated to fulfil the request. The diagram is more complex than before. This is basically a scenario envisaging the MRB as a B2BUA. The Application Server sends a SIP INVITE (1.), containing both a CFW-related SDP and a Consumer request (multipart body). The MRB sends a provisional response to the Application Server (2.) and starts working on the request. First of all, it makes use of the Consumer request from the Application Server to determine which Media Servers should be exploited. Once the right Media Servers have been chosen (MS1 and MS2 in the example), the MRB sends a new SIP INVITE to one of the Media Servers (MS1 in the example) by just including the SDP part of the original request (3.). That Media Server negotiates this INVITE as specified in [RFC6230] (4., 5., 6.), providing the MRB with its own CFW-related SDP. The MRB replies to the original Application Server INVITE preparing a SIP 200 OK with another multipart body (7.): this multipart body includes the Consumer response used by the MRB to determine the right Media Servers and the SDP returned by the Media Server (MS1) in 5. The Application Server finally acknowledges the 200 OK (8.), and can start a CFW connection towards that Media Server (MS1). Since the MRB provided the Application Server with two Media Server instances to fulfil its requirements, the Application Server can use the URI in the <media-server-address> element in the <mediaResourceResponse> that describes the other Media Server to establish a CFW channel with that Media Server (MS2) as well.

Please note that, to ease the reading of the protocol contents, a simple '=_Part' is used whenever a boundary for a 'multipart/mixed' payload is provided, instead of the actual boundary that would be inserted in the SIP messages.

AS	MRB	MS1	MS2
1. INVITE			
(multipart/mixed)			
----->			
2. 100 (Trying)			
<-----			
	--> Extract SDP and		
	MRB payloads; handle		


```

|<--+ Consumer request to pick MSs (MS1 and MS2)|
|
|3. INVITE
|(only copy SDP from 1.)
|----->
|4. 100 (Trying)
|<-----
|
|                                     |--+ Negotiate
|                                     | CFW Control
|                                     |--+ Channel
|
|5. 200 OK
|<-----
|6. ACK
|----->
|
|   Prepare new +--
|   payload with |
|SDP from MS and +->
|   Consumer reply
|
|       7. 200 OK
|   (multipart/mixed)
|<-----
|8. ACK
|----->
|
|--+ Read Cons. reply
|  | and use SDP to
|--+ create CFW Chn.
|
|
|   Create TCP CFW channel towards MS1 (if needed)
|----->
|
|<<##### TCP CONNECTION #####>>
|
| CFW SYNC
|+++++++>
|
|.
|.
|.
|
|   Negotiate SIP Control Dialog with MS2
|<----->
|   Create TCP CFW channel towards MS2 as well (if needed)
|----->
|
|<<##### TCP CONNECTION #####>>

```

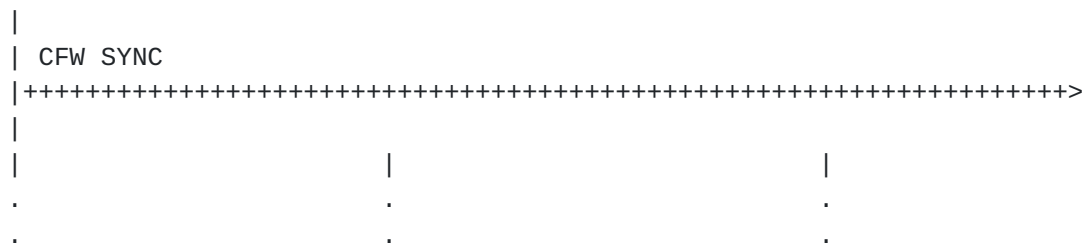



Figure 11: Consumer Example (IAMM-CFW): Sequence Diagram

The rest of this section includes an almost full trace of the messages associated with the previous sequence diagram. Only the relevant SIP messages are shown (both the INVITEs and the 200 OKs), and only the relevant headers are preserved for brevity (Content-Type and multipart-related information). Specifically:

1. the original INVITE (1), containing both a CFW-related SDP (COMEDIA information to negotiate a new Control Channel) and a Consumer <mediaResourceRequest>;
2. the INVITE sent by the MRB to the Media Server as a B2BUA (3.), containing only the CFW-related SDP from the original INVITE;.
3. the 200 OK sent by the Media Server back to the MRB (5.), to complete the CFW-related negotiation (SDP only);
4. the 200 OK sent by the MRB back to the Application Server in response to the original INVITE (7.), containing both the CFW-related information sent by the Media Server and a Consumer <mediaResourceRequest> documenting the MRB's decision to use that Media Server.

1. AS -> MRB (INVITE multipart/mixed)

```

[.]
Content-Type: multipart/mixed;boundary="_Part"

_Part
Content-Type: application/sdp

v=0
o=- 2890844526 2890842807 IN IP4 as.example.com
s=MediaCtrl
c=IN IP4 as.example.com
t=0 0
  
```


m=application 48035 TCP cfw

a=connection:new

a=setup:active

a=cfw-id:vF0zD4xzUAW9

a=ctrl-package:msc-mixer/1.0

a=ctrl-package:msc-ivr/1.0

=_Part

Content-Type: application/mrb-consumer+xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<mrbconsumer version="1.0"
```

```
  xmlns="urn:ietf:params:xml:ns:mrb-consumer">
```

```
  <mediaResourceRequest id="pz78hnq1">
```

```
    <generalInfo>
```

```
      <packages>
```

```
        <package>msc-ivr/1.0</package>
```

```
        <package>msc-mixer/1.0</package>
```

```
      </packages>
```

```
    </generalInfo>
```

```
    <ivrInfo>
```

```
      <ivr-sessions>
```

```
        <rtp-codec name="audio/basic">
```

```
          <decoding>100</decoding>
```

```
          <encoding>100</encoding>
```

```
        </rtp-codec>
```

```
      </ivr-sessions>
```

```
      <file-formats>
```

```
        <required-format name="audio/x-wav"/>
```

```
      </file-formats>
```

```
      <file-transfer-modes>
```

```
        <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>
```

```
      </file-transfer-modes>
```

```
    </ivrInfo>
```

```
  </mediaResourceRequest>
```

```
</mrbconsumer>
```

=_Part

3. MRB -> MS (INVITE sdp only)

[...]

Content-Type: application/sdp

v=0

o=- 2890844526 2890842807 IN IP4 as.example.com


```
s=MediaCtrl
c=IN IP4 as.example.com
t=0 0
m=application 48035 TCP cfw
a=connection:new
a=setup:active
a=cfw-id:vF0zD4xzUAW9
a=ctrl-package:msc-mixer/1.0
a=ctrl-package:msc-ivr/1.0
```

5. MRB <- MS (200 OK sdp)

```
-----
[...]  
Content-Type: application/sdp  
  
v=0  
o=lminiero 2890844526 2890842808 IN IP4 ms.example.net  
s=MediaCtrl  
c=IN IP4 ms.example.net  
t=0 0  
m=application 7575 TCP cfw  
a=connection:new  
a=setup:passive  
a=cfw-id:vF0zD4xzUAW9  
a=ctrl-package:msc-mixer/1.0  
a=ctrl-package:msc-ivr/1.0  
a=ctrl-package:mrbr-publish/1.0  
a=ctrl-package:msc-example-pkg/1.0
```

7. AS <- MRB (200 OK multipart/mixed)

```
-----
[...]  
Content-Type: multipart/mixed;boundary="_Part"  
  
_Part  
Content-Type: application/sdp  
  
v=0  
o=lminiero 2890844526 2890842808 IN IP4 ms.example.net  
s=MediaCtrl  
c=IN IP4 ms.example.net  
t=0 0  
m=application 7575 TCP cfw  
a=connection:new
```



```
a=setup:passive
a=cfw-id:vF0zD4xzUAW9
a=ctrl-package:msc-mixer/1.0
a=ctrl-package:msc-ivr/1.0
a=ctrl-package:mrbr-publish/1.0
a=ctrl-package:msc-example-pkg/1.0
```

=_Part

Content-Type: application/mrb-consumer+xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbrconsumer version="1.0"
  xmlns="urn:ietf:params:xml:ns:mrb-consumer" >
  <mediaResourceResponse reason="Resource found" status="200"
    id="pz78hnq1">
    <response-session-info>
      <session-id>z1skKYZQ3eFu</session-id>
      <seq>9</seq>
      <expires>3600</expires>
      <media-server-address
        uri="sip:MediaServer@ms.example.com:5080">
        <connection-id>32pbdxZ8:KQw677BF</connection-id>
        <ivr-sessions>
          <rtp-codec name="audio/basic">
            <decoding>60</decoding>
            <encoding>60</encoding>
          </rtp-codec>
        </ivr-sessions>
      </media-server-address>
      <media-server-address
        uri="sip:OtherMediaServer@pool.example.net:5080">
        <ivr-sessions>
          <rtp-codec name="audio/basic">
            <decoding>40</decoding>
            <encoding>40</encoding>
          </rtp-codec>
        </ivr-sessions>
      </media-server-address>
    </response-session-info>
  </mediaResourceResponse>
</mrbrconsumer>
```

=_Part

As the previous example illustrates, the only difference in the response the MRB provides the Application Server with is in the 'connection-id' attribute that is added to the first allocated Media

Server instance: this allows the Application Server to understand the MRB has sent the CFW channel negotiation to that specific Media Server, and that the connection-id to be used is the one provided. This will be described in more detail in the following section, for the media dialog-based approach.

The continuation of the scenario (the Application Server connecting to MS1 to start the Control Channel and the related SYNC message, the Application Server connecting to MS2 as well later on, all the media dialogs being attached to either Media Server) are omitted for brevity.

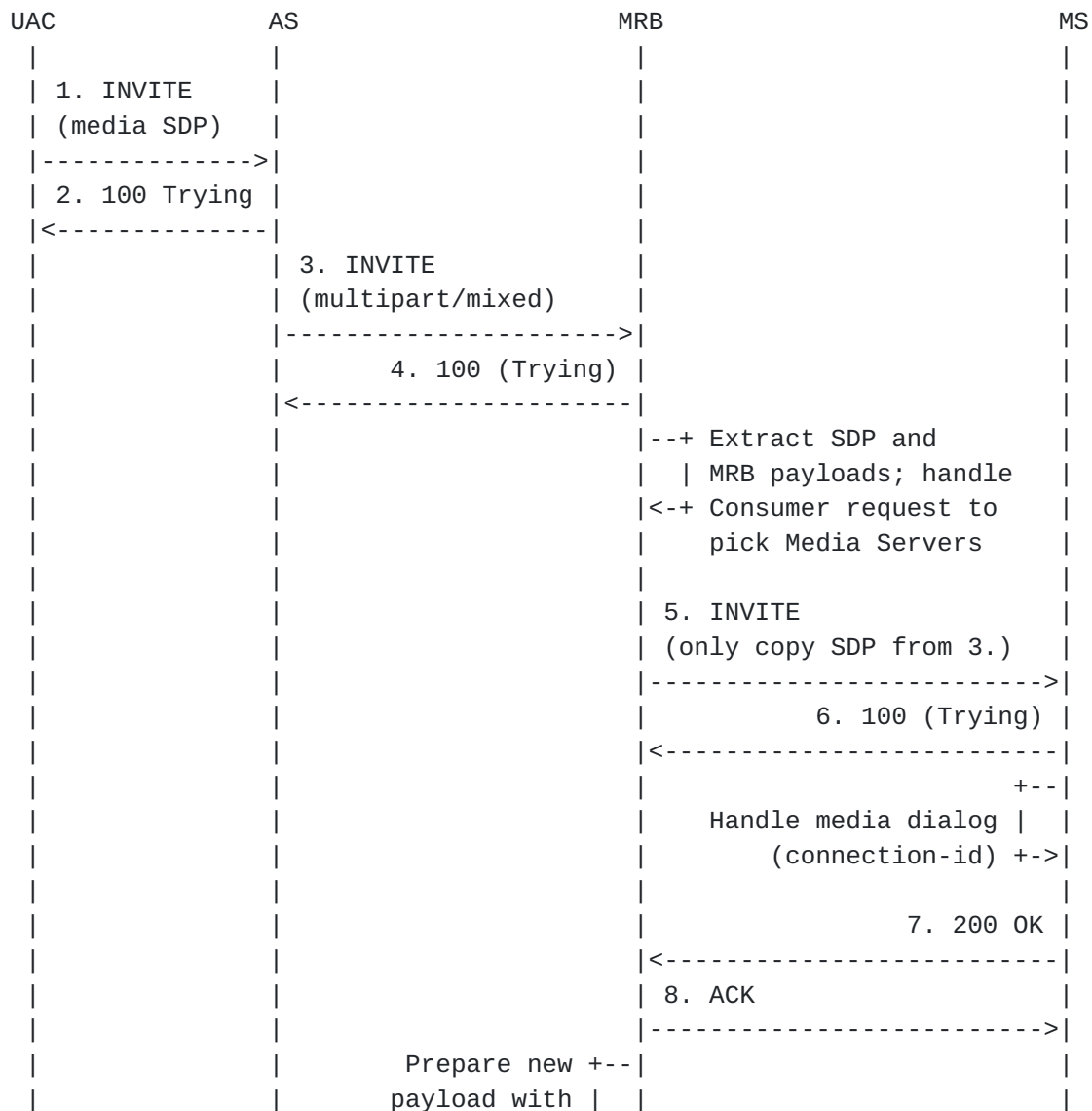
9.2.2.2. IAMM Example: Media dialog-based approach

The following example assumes the interested Application Server already knows the SIP URI of an MRB.

Figure 12 shows the second approach, i.e. SIP-based transactions between a SIP client, the Application Server, the MRB and the Media Server that the MRB chooses. The interaction is basically the same as previous examples (e.g. contents of the multipart body) but considering a new party is involved in the communication, the diagram is slightly more complex than before. As before, the MRB acts as a B2BUA. A UAC sends a SIP INVITE to a SIP URI handled by the Application Server, since it is interested to its services (1.). The Application Server sends a provisional response (2.) and, since it doesn't have the resources yet, sends to the MRB a new SIP INVITE (3.), containing both the UAC media-related SDP and a Consumer request (multipart body). The MRB sends a provisional response to the Application Server (4.) and starts working on the request. First of all, it makes use of the Consumer request from the Application Server to determine which Media Servers should be chosen. Once the Media Server has been chosen, the MRB sends a new SIP INVITE to one of the Media Servers by including the SDP part of the original request (5.). The Media Server negotiates this INVITE as specified in [[RFC6230](#)] (6., 7., 8.) to allocate the needed media resources to handle the new media dialog, eventually providing the MRB with its own media-related SDP. The MRB replies to the original Application Server INVITE preparing a SIP 200 OK with a multipart body (9.): this multipart body includes the Consumer response from the MRB indicating the chosen Media Servers and the SDP returned by the Media Server in 7. The Application Server finally acknowledges the 200 OK (10.), and ends the scenario by eventually providing the UAC with the SDP it needs to set-up the RTP channels with the chosen Media Server: a separate direct SIP control dialog may be initiated by the Application Server to the same Media Server in order to set up a control channel to manipulate the media dialog media.

As with the IAMM - CFW example in the prior section, this example has the MRB selecting Media Server resources across two Media Server instances. The convention could be that the MRB sent the SIP INVITE to the first Media Server in the list provided to the Application Server in the Consumer response information. For the sake of brevity, the considerations about connecting to the other Media Servers as well are omitted, since they have already been addressed in the previous section.

Please note that, to ease the reading of the protocol contents, a simple '=_Part' is used whenever a boundary for a 'multipart/mixed' payload is provided, instead of the actual boundary that would be inserted in the SIP messages.



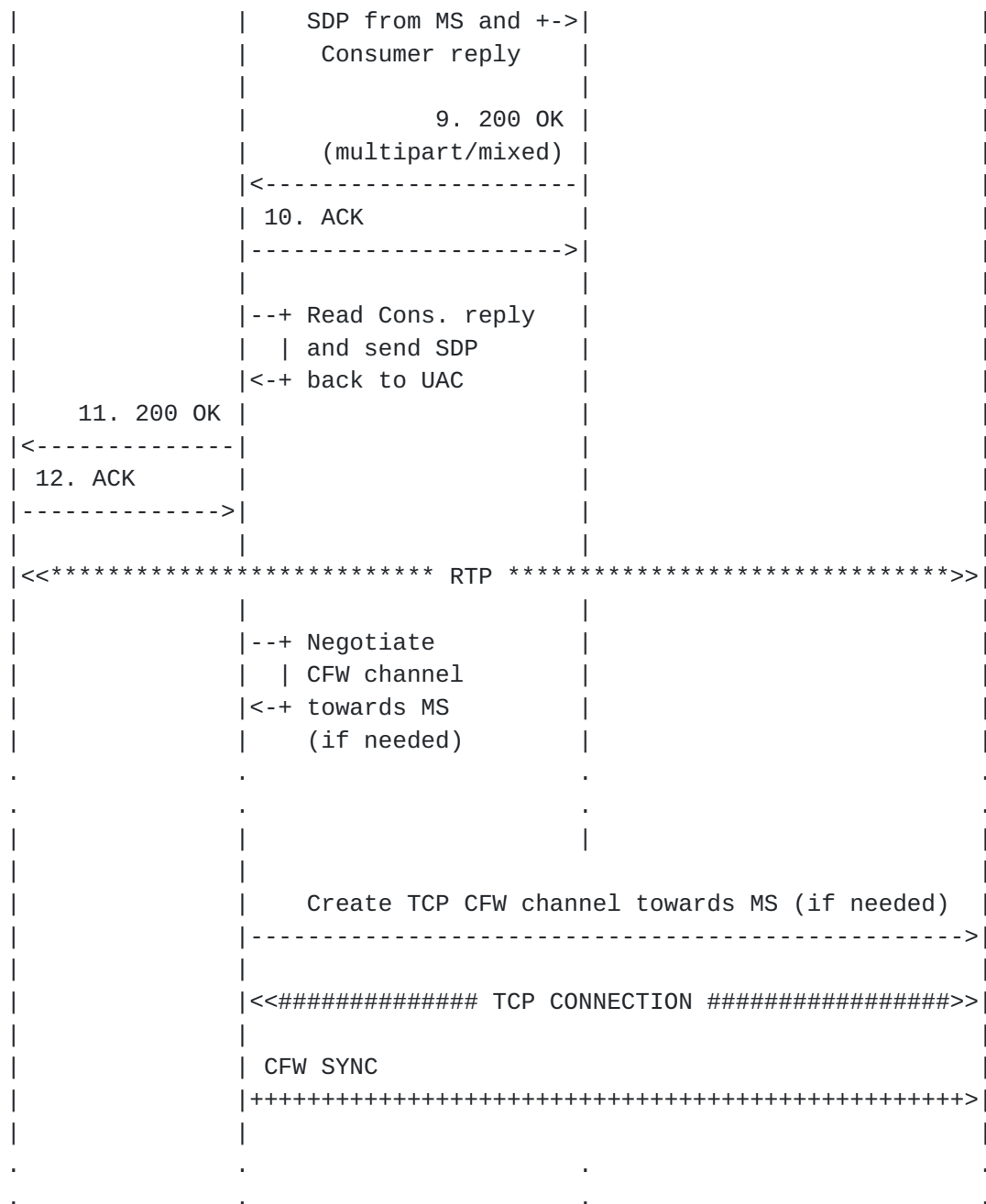


Figure 12: Consumer Example (IAMM-MediaDialog): Sequence Diagram

The rest of this section includes an trace of the messages associated with the previous sequence diagram. Only the relevant SIP messages are shown (both the INVITEs and the 200 OKs), and only the relevant headers are preserved for brevity (Content-Type, From/To and multipart-related information). Specifically:

1. the original INVITE (1), containing the media-related SDP sent by a UAC;
2. the original INVITE (3), containing both the media-related SDP and a Consumer <mediaResourceRequest>;
3. the INVITE sent by the MRB to the Media Server as a B2BUA (5.), containing only the media-related SDP from the original INVITE;
4. the 200 OK sent by the Media Server back to the MRB (7.), to complete the media-related negotiation (SDP only);
5. the 200 OK sent by the MRB back to the Application Server in response to the original INVITE (9.), containing both the media-related information sent by the Media Server and a Consumer <mediaResourceRequest> documenting the MRB's decision to use that Media Server;
6. the 200 OK sent by the Application Server back to the UAC to have it set-up the RTP channel(s) with the Media Server (11.).

1. UAC -> AS (INVITE with media SDP)

```
[...]  
From: <sip:lminiero@users.example.com>;tag=1153573888  
To: <sip:mediactrlDemo@as.example.com>  
[...]  
Content-Type: application/sdp  
  
v=0  
o=lminiero 123456 654321 IN IP4 203.0.113.2  
s=A conversation  
c=IN IP4 203.0.113.2  
t=0 0  
m=audio 7078 RTP/AVP 0 3 8 101  
a=rtpmap:0 PCMU/8000/1  
a=rtpmap:3 GSM/8000/1  
a=rtpmap:8 PCMA/8000/1  
a=rtpmap:101 telephone-event/8000  
a=fmtp:101 0-11  
m=video 9078 RTP/AVP 98
```

3. AS -> MRB (INVITE multipart/mixed)

```
[...]  
From: <sip:ApplicationServer@as.example.com>;tag=fd4fush5  
To: <sip:Mrb@mrbs.example.org>
```

```
[...]  
Content-Type: multipart/mixed;boundary="_Part"
```

```
_Part
```

```
Content-Type: application/sdp
```

```
v=0  
o=lminiero 123456 654321 IN IP4 203.0.113.2  
s=A conversation  
c=IN IP4 203.0.113.2  
t=0 0  
m=audio 7078 RTP/AVP 0 3 8 101  
a=rtpmap:0 PCMU/8000/1  
a=rtpmap:3 GSM/8000/1  
a=rtpmap:8 PCMA/8000/1  
a=rtpmap:101 telephone-event/8000  
a=fmtp:101 0-11  
m=video 9078 RTP/AVP 98
```

```
_Part
```

```
Content-Type: application/mrb-consumer+xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<mrbconsumer version="1.0"  
  xmlns="urn:ietf:params:xml:ns:mrb-consumer">  
  <mediaResourceRequest id="ns56g1x0">  
    <generalInfo>  
      <packages>  
        <package>msc-ivr/1.0</package>  
        <package>msc-mixer/1.0</package>  
      </packages>  
    </generalInfo>  
    <ivrInfo>  
      <ivr-sessions>  
        <rtp-codec name="audio/basic">  
          <decoding>100</decoding>  
          <encoding>100</encoding>  
        </rtp-codec>  
      </ivr-sessions>  
      <file-formats>  
        <required-format name="audio/x-wav"/>  
      </file-formats>  
      <file-transfer-modes>  
        <file-transfer-mode package="msc-ivr/1.0" name="HTTP"/>  
      </file-transfer-modes>
```



```
</ivrInfo>
</mediaResourceRequest>
</mrbconsumer>
```

=_Part

5. MRB -> MS (INVITE sdp only)

```
[..]
From: <sip:Mrb@mrb.example.org:5060>;tag=32pbdxZ8
To: <sip:MediaServer@ms.example.com:5080>
[..]
Content-Type: application/sdp
```

```
v=0
o=lminiero 123456 654321 IN IP4 203.0.113.2
s=A conversation
c=IN IP4 203.0.113.2
t=0 0
m=audio 7078 RTP/AVP 0 3 8 101
a=rtpmap:0 PCMU/8000/1
a=rtpmap:3 GSM/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11
m=video 9078 RTP/AVP 98
```

7. MRB <- MS (200 OK sdp)

```
[..]
From: <sip:Mrb@mrb.example.org:5060>;tag=32pbdxZ8
To: <sip:MediaServer@ms.example.com:5080>;tag=KQw677BF
[..]
Content-Type: application/sdp
```

```
v=0
o=lminiero 123456 654322 IN IP4 203.0.113.1
s=MediaCtrl
c=IN IP4 203.0.113.1
t=0 0
m=audio 63442 RTP/AVP 0 3 8 101
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:8 PCMA/8000
```



```
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
a=label:7eda834
m=video 33468 RTP/AVP 98
a=rtpmap:98 H263-1998/90000
a=fmtp:98 CIF=2
a=label:0132ca2
```

9. AS <- MRB (200 OK multipart/mixed)

```
[...]
From: <sip:ApplicationServer@as.example.com>;tag=fd4fush5
To: <sip:Mrb@mrbs.example.org>;tag=117652221
[...]
Content-Type: multipart/mixed;boundary="=_Part"

=_Part
Content-Type: application/sdp

v=0
o=lminiero 123456 654322 IN IP4 203.0.113.1
s=MediaCtrl
c=IN IP4 203.0.113.1
t=0 0
m=audio 63442 RTP/AVP 0 3 8 101
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
a=label:7eda834
m=video 33468 RTP/AVP 98
a=rtpmap:98 H263-1998/90000
a=fmtp:98 CIF=2
a=label:0132ca2

=_Part
Content-Type: application/mrb-consumer+xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrbsconsumer version="1.0"
  xmlns="urn:ietf:params:xml:ns:mrbs-consumer" >
  <mediaResourceResponse reason="Resource found" status="200"
    id="ns56g1x0">
```



```

<response-session-info>
  <session-id>z1skKYZQ3eFu</session-id>
  <seq>9</seq>
  <expires>3600</expires>
  <media-server-address
    uri="sip:MediaServer@ms.example.com:5080">
  <connection-id>32pbdxZ8:KQw677BF</connection-id>
  <ivr-sessions>
    <rtp-codec name="audio/basic">
      <decoding>60</decoding>
      <encoding>60</encoding>
    </rtp-codec>
  </ivr-sessions>
</media-server-address>
<media-server-address
  uri="sip:OtherMediaServer@pool.example.net:5080">
  <ivr-sessions>
    <rtp-codec name="audio/basic">
      <decoding>40</decoding>
      <encoding>40</encoding>
    </rtp-codec>
  </ivr-sessions>
</media-server-address>
</response-session-info>
</mediaResourceResponse>
</mrbcconsumer>

=_Part

```

11. UAC <- AS (200 OK sdp)

```

[.]
From: <sip:lminiero@users.example.com>;tag=1153573888
To: <sip:mediactrlDemo@as.example.com>;tag=bcd47c32
[.]
Content-Type: application/sdp

v=0
o=lminiero 123456 654322 IN IP4 203.0.113.1
s=MediaCtrl
c=IN IP4 203.0.113.1
t=0 0
m=audio 63442 RTP/AVP 0 3 8 101
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:8 PCMA/8000

```



```
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
a=label:7eda834
m=video 33468 RTP/AVP 98
a=rtpmap:98 H263-1998/90000
a=fmtp:98 CIF=2
a=label:0132ca2
```

As the examples illustrate, as in the IAMM-CFW example, the MRB provides the Application Server with a 'media-server-address' element in the consumer response: the 'uri' attribute identifies the specific Media Server to which the MRB has sent the SDP media negotiation, and the 'connection-id' enables the Application Server to identify to the Media Server the dialog between the MRB and Media Server. This attribute is needed, since, according to the framework specification, the connection-id is built out of the From/To tags of the dialog between the MRB and Media Server; since the MRB acts as a B2BUA in this scenario, without that attribute the Application Server does not know the relevant tags, thus preventing the CFW protocol to work as expected.

The continuation of the scenario (the Application Server connecting to the Media Server to start the Control Channel, the SYNC message, etc.) are omitted for brevity.

10. Media Service Resource Publisher Interface XML Schema

This section gives the XML Schema Definition [W3C.REC-xmlschema-1-20041028], [W3C.REC-xmlschema-2-20041028] of the "application/mrb-publish+xml" format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:ietf:params:xml:ns:mrb-publish"
  elementFormDefault="qualified" blockDefault="#all"
  xmlns="urn:ietf:params:xml:ns:mrb-publish"
  xmlns:fw="urn:ietf:params:xml:ns:control:framework-attributes"
  xmlns:ca="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<xsd:annotation>
  <xsd:documentation>
    IETF MediaCtrl MRB 1.0
```

This is the schema of the IETF MediaCtrl MRB package.

The schema namespace is urn:ietf:params:xml:ns:mrb-publish

```
</xsd:documentation>
</xsd:annotation>
```

```
<!--
#####

SCHEMA IMPORTS

#####
-->
```

```
<xsd:import namespace="http://www.w3.org/XML/1998/namespace"
  schemaLocation="http://www.w3.org/2001/xml.xsd">
  <xsd:annotation>
    <xsd:documentation>
      This import brings in the XML attributes for
      xml:base, xml:lang, etc
    </xsd:documentation>
  </xsd:annotation>
</xsd:import>
```

```
<xsd:import
  namespace="urn:ietf:params:xml:ns:control:framework-attributes"
  schemaLocation="framework.xsd">
```



```
<xsd:annotation>
  <xsd:documentation>
    This import brings in the framework attributes for
    conferenceid and connectionid.
  </xsd:documentation>
</xsd:annotation>
</xsd:import>

<xsd:import
  namespace="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
  schemaLocation="civicAddress.xsd">
  <xsd:annotation>
    <xsd:documentation>
      This import brings in the civicAddress specification
      from RFC5139.
    </xsd:documentation>
  </xsd:annotation>
</xsd:import>

<!--
#####

Extensible core type

#####
-->

<xsd:complexType name="Tcore">
  <xsd:annotation>
    <xsd:documentation>
      This type is extended by other (non-mixed) component types to
      allow attributes from other namespaces.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<!--
#####

TOP LEVEL ELEMENT: mrbpublish

#####
-->
```



```

<xsd:complexType name="mrbpublishType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element ref="mrbrequest" />
          <xsd:element ref="mrbresponse" />
          <xsd:element ref="mrnotification" />
          <xsd:any namespace="##other" minOccurs="0"
            maxOccurs="unbounded" processContents="lax" />
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="version" type="version.datatype"
        use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="mrbpublish" type="mrbpublishType" />

```

```

<!--

```

```

#####

```

```

mrbrequest TYPE

```

```

#####

```

```

-->

```

```

<!-- mrbrequest -->

```

```

<xsd:complexType name="mrbrequestType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="subscription" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="mrbrequest" type="mrbrequestType" />

```

```

<!-- subscription -->

```



```

<xsd:complexType name="subscriptionType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="expires" type="xsd:nonNegativeInteger"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="minfrequency" type="xsd:nonNegativeInteger"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="maxfrequency" type="xsd:nonNegativeInteger"
          minOccurs="0" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="id" type="id.datatype" use="required" />
      <xsd:attribute name="seqnumber" type="xsd:nonNegativeInteger"
        use="required" />
      <xsd:attribute name="action" type="action.datatype"
        use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="subscription" type="subscriptionType" />

<!--
#####

mrbrresponse TYPE

#####
-->

<!-- mrbrresponse -->

<xsd:complexType name="mrbrresponseType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="subscription" minOccurs="0" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="status" type="status.datatype"
        use="required" />
      <xsd:attribute name="reason" type="xsd:string" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```



```
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="mrbresponse" type="mrbresponseType" />

<!--
#####

mrbnotification TYPE

#####
-->

<!-- mrbnotification -->

<xsd:complexType name="mrbnotificationType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="media-server-id"
          type="subscriptionid.datatype"/>
        <xsd:element ref="supported-packages" minOccurs="0" />
        <xsd:element ref="active-rtp-sessions" minOccurs="0" />
        <xsd:element ref="active-mixer-sessions" minOccurs="0" />
        <xsd:element ref="non-active-rtp-sessions" minOccurs="0" />
        <xsd:element ref="non-active-mixer-sessions" minOccurs="0" />
        <xsd:element ref="media-server-status" minOccurs="0" />
        <xsd:element ref="supported-codecs" minOccurs="0" />
        <xsd:element ref="application-data" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:element ref="file-formats" minOccurs="0" />
        <xsd:element ref="max-prepared-duration" minOccurs="0" />
        <xsd:element ref="dtmf-support" minOccurs="0" />
        <xsd:element ref="mixing-modes" minOccurs="0" />
        <xsd:element ref="supported-tones" minOccurs="0" />
        <xsd:element ref="file-transfer-modes" minOccurs="0" />
        <xsd:element ref="asr-tts-support" minOccurs="0" />
        <xsd:element ref="vxml-support" minOccurs="0" />
        <xsd:element ref="media-server-location" minOccurs="0" />
        <xsd:element ref="label" minOccurs="0" />
        <xsd:element ref="media-server-address" minOccurs="0" />
        <xsd:element ref="encryption" minOccurs="0" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="id" type="subscriptionid.datatype"
```



```
        use="required" />
        <xsd:attribute name="seqnumber" type="xsd:nonNegativeInteger"
            use="required" />
        <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="mrnotification" type="mrnotificationType" />

<!-- supported-packages -->

<xsd:complexType name="supported-packagesType">
    <xsd:complexContent>
        <xsd:extension base="Tcore">
            <xsd:sequence>
                <xsd:element ref="package" minOccurs="0"
                    maxOccurs="unbounded" />
                <xsd:any namespace="##other" minOccurs="0"
                    maxOccurs="unbounded" processContents="lax" />
            </xsd:sequence>
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-packages" type="supported-packagesType"/>

<xsd:complexType name="packageType">
    <xsd:complexContent>
        <xsd:extension base="Tcore">
            <xsd:sequence>
                <xsd:any namespace="##other" minOccurs="0"
                    maxOccurs="unbounded" processContents="lax" />
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required" />
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="package" type="packageType" />

<!-- active-rtp-sessions -->
```



```
<xsd:complexType name="active-rtp-sessionsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="rtp-codec" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="active-rtp-sessions" type="active-rtp-sessionsType"/>

<xsd:complexType name="rtp-codecType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="decoding" type="xsd:nonNegativeInteger" />
        <xsd:element name="encoding" type="xsd:nonNegativeInteger" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="rtp-codec" type="rtp-codecType" />

<!-- active-mixer-sessions -->

<xsd:complexType name="active-mixer-sessionsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="active-mix" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="active-mixer-sessions"
  type="active-mixer-sessionsType" />

<xsd:complexType name="active-mixType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="rtp-codec" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attributeGroup ref="fw:framework-attributes" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="active-mix" type="active-mixType" />

<!-- non-active-rtp-sessions -->

<xsd:complexType name="non-active-rtp-sessionsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="rtp-codec" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="non-active-rtp-sessions"
  type="non-active-rtp-sessionsType" />

<!-- non-active-mixer-sessions -->

<xsd:complexType name="non-active-mixer-sessionsType">
  <xsd:complexContent>
```



```
<xsd:extension base="Tcore">
  <xsd:sequence>
    <xsd:element ref="non-active-mix" minOccurs="0"
      maxOccurs="unbounded" />
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="non-active-mixer-sessions"
  type="non-active-mixer-sessionsType" />

<xsd:complexType name="non-active-mixType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="rtp-codec" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
        <xsd:attribute name="available" type="xsd:nonNegativeInteger"
          use="required" />
        <xsd:anyAttribute namespace="##other" processContents="lax" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="non-active-mix" type="non-active-mixType" />

<!-- media-server-status -->

  <xsd:element name="media-server-status" type="msstatus.datatype" />

<!-- supported-codecs -->

<xsd:complexType name="supported-codecsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="supported-codec"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
        <xsd:anyAttribute namespace="##other" processContents="lax" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```



```
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-codecs" type="supported-codecsType" />

<xsd:complexType name="supported-codecType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="supported-codec-package"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-codec" type="supported-codecType" />

<xsd:complexType name="supported-codec-packageType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="supported-action" type="actions.datatype"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-codec-package"
  type="supported-codec-packageType" />

<!-- application-data -->

<xsd:element name="application-data" type="appdata.datatype" />

<!-- file-formats -->
```



```
<xsd:complexType name="file-formatsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="supported-format"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-formats" type="file-formatsType" />

<xsd:complexType name="supported-formatType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="supported-file-package"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-format" type="supported-formatType" />

<xsd:element name="supported-file-package"
  type="xsd:string" />

<!-- max-prepared-duration -->

<xsd:complexType name="max-prepared-durationType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="max-time" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="max-prepared-duration"
  type="max-prepared-durationType" />

<xsd:complexType name="max-timeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="max-time-package" type="xsd:string" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="max-time-seconds" type="xsd:nonNegativeInteger"
        use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="max-time" type="max-timeType" />

<!-- dtmf-support -->

<xsd:complexType name="dtmf-supportType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="detect" />
        <xsd:element ref="generate" />
        <xsd:element ref="passthrough" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="dtmf-support" type="dtmf-supportType" />

<xsd:complexType name="detectType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="dtmf-type"
```



```
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="detect" type="detectType" />

<xsd:complexType name="generateType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="dtmf-type"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="generate" type="generateType" />

<xsd:complexType name="passthroughType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="dtmf-type"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="passthrough" type="passthroughType" />

<xsd:complexType name="dtmf-typeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
```



```
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="dtmf.datatype" use="required" />
      <xsd:attribute name="package" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="dtmf-type" type="dtmf-typeType" />

<!-- mixing-modes -->

<xsd:complexType name="mixing-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="audio-mixing-modes"
          minOccurs="0" maxOccurs="1" />
        <xsd:element ref="video-mixing-modes"
          minOccurs="0" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixing-modes" type="mixing-modesType" />

<xsd:complexType name="audio-mixing-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="audio-mixing-mode"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="audio-mixing-modes" type="audio-mixing-modesType" />
```



```
<xsd:complexType name="audio-mixing-modeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="audio-mixing-mode" type="audio-mixing-modeType" />

<xsd:complexType name="video-mixing-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="video-mixing-mode"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="vas" type="boolean.datatype"
        default="false" />
      <xsd:attribute name="activespeakermix" type="boolean.datatype"
        default="false" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="video-mixing-modes" type="video-mixing-modesType" />

<xsd:complexType name="video-mixing-modeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="video-mixing-mode" type="video-mixing-modeType" />

<!-- supported-tones -->

<xsd:complexType name="supported-tonesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
```



```
<xsd:sequence>
  <xsd:element ref="supported-country-codes"
    minOccurs="0" maxOccurs="1" />
  <xsd:element ref="supported-h248-codes"
    minOccurs="0" maxOccurs="1" />
  <xsd:any namespace="##other" minOccurs="0"
    maxOccurs="unbounded" processContents="lax" />
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-tones" type="supported-tonesType" />

<xsd:complexType name="supported-country-codesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="country-code"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-country-codes"
  type="supported-country-codesType" />

<xsd:complexType name="country-codeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="country-code" type="country-codeType" />

<xsd:complexType name="supported-h248-codesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="h248-code"
```



```
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="supported-h248-codes"
  type="supported-h248-codesType" />

<xsd:complexType name="h248-codeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="h248-code" type="h248-codeType" />

<!-- file-transfer-modes -->

<xsd:complexType name="file-transfer-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="file-transfer-mode"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-transfer-modes"
  type="file-transfer-modesType" />

<xsd:complexType name="file-transfer-modeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0"
```



```
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="name" type="transfermode.datatype"
        use="required" />
    <xsd:attribute name="package" type="xsd:string" use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="file-transfer-mode" type="file-transfer-modeType" />

<!-- asr-tts-support -->

<xsd:complexType name="asr-tts-supportType">
    <xsd:complexContent>
        <xsd:extension base="Tcore">
            <xsd:sequence>
                <xsd:element ref="asr-support"
                    minOccurs="0" maxOccurs="1" />
                <xsd:element ref="tts-support"
                    minOccurs="0" maxOccurs="1" />
                <xsd:any namespace="##other" minOccurs="0"
                    maxOccurs="unbounded" processContents="lax" />
            </xsd:sequence>
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-tts-support" type="asr-tts-supportType" />

<xsd:complexType name="asr-supportType">
    <xsd:complexContent>
        <xsd:extension base="Tcore">
            <xsd:sequence>
                <xsd:element ref="language"
                    minOccurs="0" maxOccurs="unbounded" />
                <xsd:any namespace="##other" minOccurs="0"
                    maxOccurs="unbounded" processContents="lax" />
            </xsd:sequence>
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-support" type="asr-supportType" />
```



```
<xsd:complexType name="tts-supportType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="language"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="tts-support" type="tts-supportType" />

<xsd:complexType name="languageType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute ref="xml:lang" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="language" type="languageType" />

<!-- media-server-location -->

<xsd:complexType name="media-server-locationType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="civicAddress" type="ca:civicAddress"
          minOccurs="1" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
<xsd:element name="media-server-location"
  type="media-server-locationType" />

<!-- vxml-support -->

<xsd:complexType name="vxml-supportType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="vxml-mode"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="vxml-support" type="vxml-supportType" />

<xsd:complexType name="vxml-modeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="package" type="xsd:string" use="required" />
      <xsd:attribute name="support" type="vxml.datatype" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="vxml-mode" type="vxml-modeType" />

<!-- label -->

<xsd:element name="label" type="label.datatype" />

<!-- media-server-address -->

<xsd:element name="media-server-address" type="xsd:anyURI" />
```



```

<!-- encryption -->

<xsd:complexType name="encryptionType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="keying-mechanism"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="encryption" type="encryptionType" />

<xsd:element name="keying-mechanism" type="keying.datatype" />

<!--
#####

DATATYPES

#####
-->

<xsd:simpleType name="version.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="1.0" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="id.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

<xsd:simpleType name="status.datatype">
  <xsd:restriction base="xsd:positiveInteger">
    <xsd:pattern value="[0-9][0-9][0-9]" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="msstatus.datatype">
  <xsd:restriction base="xsd:NMTOKEN">

```



```
<xsd:enumeration value="active" />
<xsd:enumeration value="deactivated" />
<xsd:enumeration value="unavailable" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="action.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="create" />
    <xsd:enumeration value="update" />
    <xsd:enumeration value="remove" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="actions.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="encoding" />
    <xsd:enumeration value="decoding" />
    <xsd:enumeration value="passthrough" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="appdata.datatype">
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>

<xsd:simpleType name="dtmf.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="transfermode.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

<xsd:simpleType name="boolean.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="true" />
    <xsd:enumeration value="false" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="vxml.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="label.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>
```



```
<xsd:simpleType name="subscriptionid.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

<xsd:simpleType name="keying.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

</xsd:schema>
```

Figure 13

11. Media Service Resource Consumer Interface XML Schema

This section gives the XML Schema Definition [W3C.REC-xmlschema-1-20041028], [W3C.REC-xmlschema-2-20041028] of the "application/mrb-consumer+xml" format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:ietf:params:xml:ns:mrb-consumer"
  elementFormDefault="qualified" blockDefault="#all"
  xmlns="urn:ietf:params:xml:ns:mrb-consumer"
  xmlns:ca="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation>
      IETF MediaCtrl MRB 1.0

      This is the schema of the IETF MediaCtrl MRB Consumer interface.

      The schema namespace is urn:ietf:params:xml:ns:mrb-consumer

    </xsd:documentation>
  </xsd:annotation>

  <!--
  #####

  SCHEMA IMPORTS

  #####
  -->

  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd">
    <xsd:annotation>
      <xsd:documentation>
        This import brings in the XML attributes for
        xml:base, xml:lang, etc
      </xsd:documentation>
    </xsd:annotation>
  </xsd:import>

  <xsd:import
    namespace="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
    schemaLocation="civicAddress.xsd">
    <xsd:annotation>
```



```
<xsd:documentation>
  This import brings in the civicAddress specification
  from RFC5139.
</xsd:documentation>
</xsd:annotation>
</xsd:import>

<!--
#####

Extensible core type

#####
-->

<xsd:complexType name="Tcore">
  <xsd:annotation>
    <xsd:documentation>
      This type is extended by other (non-mixed) component types to
      allow attributes from other namespaces.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence/>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<!--
#####

TOP LEVEL ELEMENT: mrbconsumer

#####
-->

<xsd:complexType name="mrbconsumerType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element ref="mediaResourceRequest" />
          <xsd:element ref="mediaResourceResponse" />
          <xsd:any namespace="##other" minOccurs="0"
            maxOccurs="unbounded" processContents="lax" />
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="version" type="version.datatype">
```



```
        use="required" />
        <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="mrbcconsumer" type="mrbcconsumerType" />

<!--
#####

mediaResourceRequest TYPE

#####
-->

<!-- mediaResourceRequest -->

<xsd:complexType name="mediaResourceRequestType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="generalInfo" minOccurs="0" />
        <xsd:element ref="ivrInfo" minOccurs="0" />
        <xsd:element ref="mixerInfo" minOccurs="0" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string"
        use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mediaResourceRequest"
  type="mediaResourceRequestType" />

<!--
#####

generalInfo TYPE

#####
-->

<!-- generalInfo -->
```



```
<xsd:complexType name="generalInfoType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="session-info" minOccurs="0" />
        <xsd:element ref="packages" minOccurs="0" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="generalInfo" type="generalInfoType" />
```

```
<!-- session-info -->
```

```
<xsd:complexType name="session-infoType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="session-id" type="id.datatype"/>
        <xsd:element name="seq" type="xsd:nonNegativeInteger"/>
        <xsd:element name="action" type="action.datatype"/>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:element name="session-info" type="session-infoType" />
```

```
<!-- packages -->
```

```
<xsd:complexType name="packagesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="package" type="xsd:string" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="packages" type="packagesType"/>

<!--
#####

ivrInfo TYPE

#####
-->

<!-- ivrInfo -->

<xsd:complexType name="ivrInfoType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="ivr-sessions" minOccurs="0" />
        <xsd:element ref="file-formats" minOccurs="0" />
        <xsd:element ref="dtmf-type" minOccurs="0" />
        <xsd:element ref="tones" minOccurs="0" />
        <xsd:element ref="asr-tts" minOccurs="0" />
        <xsd:element ref="vxml" minOccurs="0" />
        <xsd:element ref="location" minOccurs="0" />
        <xsd:element ref="encryption" minOccurs="0" />
        <xsd:element ref="application-data" minOccurs="0" />
        <xsd:element ref="max-prepared-duration" minOccurs="0" />
        <xsd:element ref="file-transfer-modes" minOccurs="0" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="ivrInfo" type="ivrInfoType" />

<!--
#####

mixerInfo TYPE
```



```
#####
-->

<!-- mixerInfo -->

<xsd:complexType name="mixerInfoType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="mixers" minOccurs="0"/>
        <xsd:element ref="file-formats" minOccurs="0"/>
        <xsd:element ref="dtmf-type" minOccurs="0"/>
        <xsd:element ref="tones" minOccurs="0"/>
        <xsd:element ref="mixing-modes" minOccurs="0"/>
        <xsd:element ref="application-data" minOccurs="0"/>
        <xsd:element ref="location" minOccurs="0"/>
        <xsd:element ref="encryption" minOccurs="0"/>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixerInfo" type="mixerInfoType" />

<!--
#####

mediaResourceResponse TYPE

#####
-->

<!-- mediaResourceResponse -->

<xsd:complexType name="mediaResourceResponseType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="response-session-info" minOccurs="0" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string"
        use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```

    <xsd:attribute name="status" type="status.datatype"
      use="required" />
    <xsd:attribute name="reason" type="xsd:string" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="mediaResourceResponse"
  type="mediaResourceResponseType" />

```

```

<!--
#####

ELEMENTS

#####
-->

```

```

<!-- response-session-info -->

```

```

<xsd:complexType name="response-session-infoType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="session-id" type="id.datatype"/>
        <xsd:element name="seq" type="xsd:nonNegativeInteger"/>
        <xsd:element name="expires" type="xsd:nonNegativeInteger"/>
        <xsd:element ref="media-server-address"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

<xsd:element name="response-session-info"
  type="response-session-infoType" />

```

```

<!-- media-server-address -->

```

```

<xsd:complexType name="media-server-addressTYPE">
  <xsd:complexContent>
    <xsd:extension base="Tcore">

```



```
<xsd:sequence>
  <xsd:element name="connection-id" type="xsd:string"
    minOccurs="0" maxOccurs="unbounded" />
  <xsd:element ref="ivr-sessions" minOccurs="0"/>
  <xsd:element ref="mixers" minOccurs="0"/>
  <xsd:any namespace="##other" minOccurs="0"
    maxOccurs="unbounded" processContents="lax" />
</xsd:sequence>
<xsd:attribute name="uri" type="xsd:anyURI" use="required" />
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="media-server-address"
  type="media-server-addressTYPE" />

<!-- ivr-sessions -->

<xsd:complexType name="ivr-sessionsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="rtp-codec" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="ivr-sessions" type="ivr-sessionsType" />

<xsd:complexType name="rtp-codecType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="decoding" type="xsd:nonNegativeInteger" />
        <xsd:element name="encoding" type="xsd:nonNegativeInteger" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
```



```
</xsd:complexType>

<xsd:element name="rtp-codec" type="rtp-codecType" />

<!-- file-formats -->

<xsd:complexType name="file-formatsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="required-format"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-formats" type="file-formatsType" />

<xsd:complexType name="required-formatType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="required-file-package"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="required-format" type="required-formatType" />

<xsd:complexType name="required-file-packageType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="required-file-package-name" type="xsd:string"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="required-file-package"
  type="required-file-packageType" />

<!-- dtmf-type -->

<xsd:complexType name="dtmfType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="detect" />
        <xsd:element ref="generate" />
        <xsd:element ref="passthrough" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="dtmf" type="dtmfType" />

<xsd:complexType name="detectType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="dtmf-type"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="detect" type="detectType" />

<xsd:complexType name="generateType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
```



```
<xsd:element ref="dtmf-type"
  minOccurs="0" maxOccurs="unbounded" />
<xsd:any namespace="##other" minOccurs="0"
  maxOccurs="unbounded" processContents="lax" />
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="generate" type="generateType" />

<xsd:complexType name="passthroughType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="dtmf-type"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="passthrough" type="passthroughType" />

<xsd:complexType name="dtmf-typeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="name" type="dtmf.datatype" use="required" />
      <xsd:attribute name="package" type="xsd:string" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="dtmf-type" type="dtmf-typeType" />

<!-- tones -->

<xsd:complexType name="required-tonesType">
  <xsd:complexContent>
```



```
<xsd:extension base="Tcore">
  <xsd:sequence>
    <xsd:element ref="country-codes"
      minOccurs="0" maxOccurs="1" />
    <xsd:element ref="h248-codes"
      minOccurs="0" maxOccurs="1" />
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="tones" type="required-tonesType" />

<xsd:complexType name="required-country-codesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="country-code"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="country-codes"
  type="required-country-codesType" />

<xsd:complexType name="country-codeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="country-code" type="country-codeType" />

<xsd:complexType name="required-h248-codesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
```



```
<xsd:element ref="h248-code"
  minOccurs="0" maxOccurs="unbounded" />
<xsd:any namespace="##other" minOccurs="0"
  maxOccurs="unbounded" processContents="lax" />
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="h248-codes"
  type="required-h248-codesType" />

<xsd:complexType name="h248-codeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="h248-code" type="h248-codeType" />

<!-- asr-tts -->

<xsd:complexType name="asr-ttsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="asr-support"
          minOccurs="0" maxOccurs="1" />
        <xsd:element ref="tts-support"
          minOccurs="0" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-tts" type="asr-ttsType" />

<xsd:complexType name="asr-supportType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
```



```
<xsd:element ref="language"
  minOccurs="0" maxOccurs="unbounded" />
<xsd:any namespace="##other" minOccurs="0"
  maxOccurs="unbounded" processContents="lax" />
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="asr-support" type="asr-supportType" />

<xsd:complexType name="tts-supportType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="language"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="tts-support" type="tts-supportType" />

<xsd:complexType name="languageType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute ref="xml:lang" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="language" type="languageType" />

<!-- vxml -->

<xsd:complexType name="vxmlType">
  <xsd:complexContent>
```



```
<xsd:extension base="Tcore">
  <xsd:sequence>
    <xsd:element ref="vxml-mode"
      minOccurs="0" maxOccurs="unbounded" />
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="vxml" type="vxmlType" />

<xsd:complexType name="vxml-modeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="package" type="xsd:string" use="required" />
      <xsd:attribute name="require" type="vxml.datatype" use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="vxml-mode" type="vxml-modeType" />

<!-- location -->

<xsd:complexType name="locationType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="ca:civicAddress"
          minOccurs="1" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="location" type="locationType" />
```



```
<!-- encryption -->

<xsd:complexType name="encryptionType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="keying-mechanism"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="encryption" type="encryptionType" />

<xsd:element name="keying-mechanism" type="keying.datatype" />

<!-- application-data -->

<xsd:element name="application-data" type="appdata.datatype" />

<!-- max-prepared-duration -->

<xsd:complexType name="max-prepared-durationType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="max-time" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="max-prepared-duration"
  type="max-prepared-durationType" />

<xsd:complexType name="max-timeType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element name="max-time-package" type="xsd:string" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
        <xsd:any namespace="##other" minOccurs="0"
            maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="max-time-seconds" type="xsd:nonNegativeInteger"
        use="required" />
    <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="max-time" type="max-timeType" />

<!-- stream-mode -->

<xsd:complexType name="file-transfer-modesType">
    <xsd:complexContent>
        <xsd:extension base="Tcore">
            <xsd:sequence>
                <xsd:element ref="file-transfer-mode"
                    minOccurs="0" maxOccurs="unbounded" />
                <xsd:any namespace="##other" minOccurs="0"
                    maxOccurs="unbounded" processContents="lax" />
            </xsd:sequence>
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-transfer-modes"
    type="file-transfer-modesType" />

<xsd:complexType name="file-transfer-modeType">
    <xsd:complexContent>
        <xsd:extension base="Tcore">
            <xsd:sequence>
                <xsd:any namespace="##other" minOccurs="0"
                    maxOccurs="unbounded" processContents="lax" />
            </xsd:sequence>
            <xsd:attribute name="name" type="transfermode.datatype"
                use="required" />
            <xsd:attribute name="package" type="xsd:string" use="required" />
            <xsd:anyAttribute namespace="##other" processContents="lax" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="file-transfer-mode" type="file-transfer-modeType" />
```



```
<!-- mixers -->

<xsd:complexType name="mixerssessionsType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="mix" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixers" type="mixerssessionsType" />

<xsd:complexType name="mixType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="rtp-codec" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="users" type="xsd:nonNegativeInteger"
        use="required" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mix" type="mixType" />

<!-- mixing-modes -->

<xsd:complexType name="mixing-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="audio-mixing-modes"
          minOccurs="0" maxOccurs="1" />
        <xsd:element ref="video-mixing-modes"
          minOccurs="0" maxOccurs="1" />
        <xsd:any namespace="##other" minOccurs="0" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```



```
        maxOccurs="unbounded" processContents="lax" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="mixing-modes" type="mixing-modesType" />

<xsd:complexType name="audio-mixing-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="audio-mixing-mode"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="audio-mixing-modes" type="audio-mixing-modesType" />

<xsd:complexType name="audio-mixing-modeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="audio-mixing-mode" type="audio-mixing-modeType" />

<xsd:complexType name="video-mixing-modesType">
  <xsd:complexContent>
    <xsd:extension base="Tcore">
      <xsd:sequence>
        <xsd:element ref="video-mixing-mode"
          minOccurs="0" maxOccurs="unbounded" />
        <xsd:any namespace="##other" minOccurs="0"
          maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
      <xsd:attribute name="vas" type="boolean.datatype"
        default="false" />
      <xsd:attribute name="activespeakermix" type="boolean.datatype"
```



```
        default="false" />
      <xsd:anyAttribute namespace="##other" processContents="lax" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="video-mixing-modes" type="video-mixing-modesType" />

<xsd:complexType name="video-mixing-modeType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xsd:sequence>
  <xsd:attribute name="package" type="xsd:string" use="required" />
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>

<xsd:element name="video-mixing-mode" type="video-mixing-modeType" />

<!--
#####

DATATYPES

#####
-->

<xsd:simpleType name="version.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="1.0" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="id.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

<xsd:simpleType name="status.datatype">
  <xsd:restriction base="xsd:positiveInteger">
    <xsd:pattern value="[0-9][0-9][0-9]" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="transfermode.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>
```



```
<xsd:simpleType name="action.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="remove" />
    <xsd:enumeration value="update" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="dtmf.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="boolean.datatype">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="true" />
    <xsd:enumeration value="false" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="vxml.datatype">
  <xsd:restriction base="xsd:NMTOKEN"/>
</xsd:simpleType>

<xsd:simpleType name="appdata.datatype">
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>

<xsd:simpleType name="keying.datatype">
  <xsd:restriction base="xsd:NMTOKEN" />
</xsd:simpleType>

</xsd:schema>
```

Figure 14

12. Security Considerations

The MRB network entity has two primary interfaces, Publish and Consumer, that carry sensitive information and must therefore be appropriately protected and secured.

The Publish interface, as defined in, and described in [Section 5.1](#), uses the Media Control Channel Framework [[RFC6230](#)] as a mechanism to connect an MRB to a media server. It is very important that the communication between the MRB and the Media Server is secured: a malicious entity, may change or even delete subscriptions to a Media Server, thus affecting the view the MRB has of the resources actually available on a Media Server, leading it to incorrect selection when media resources are being requested by an Application Server. A malicious entity may even manipulate resources availability on a Media Server, for example, to make the MRB think no resources are available at all. Considering the Publish interface is a CFW Control Package, the same Security Considerations included in the Media Control Channel Framework specification apply here to protect interactions between an MRB and a Media Server.

The Publish interface also allows a Media Server, as explained in [Section 5.1.5.18](#), to provide more or less accurate information about its geographic location, should Application Servers be interested in such kind of details when looking for services at a Media Resource Broker. While the usage of this information is entirely optional and the level of detail to be provided implementation specific, it is important to draw the attention on the potential security issues that the disclosure of such addresses may introduce. As such, it is important to make sure MRB implementations don't disclose this information as is to interested Application Servers, but only exploit those addresses as part of computation algorithms to pick the most adequate resources Application Servers may be looking for.

The Consumer interface, as defined in and described in [Section 5.2](#), conceives transactions based on a session ID. These transactions may be transported either by means of HTTP messages, or SIP dialogs. This means that malicious users could be able to disrupt or manipulate an MRB session should they have access to the above mentioned session ID or replicate it somehow: for instance, a malicious entity could modify an existing session between an Application Server and the MRB, e.g., requesting less resources than originally requested to cause media dialogs to be rejected by the Application Server, or requesting many more resources instead to try and lock as many of (if not all) the resources a MRB can provide, thus making them unavailable to other legitimate Application Servers in subsequent requests. In order to prevent this, it is strongly

advised for MRB implementations to generate very hard to replicate session identifiers, in order to minimize the chances malicious users could gain access to valid ones just guessing or by means of brute force attacks. It is very important, of course, to also secure the way these identifiers are transported by the involved parties, both in requests and responses, in order to prevent network attackers from intercepting Consumer messages and have access to session IDs. The Consumer interface uses either the Hypertext Transfer Protocol (HTTP) or Session Initiation Protocol (SIP) as the mechanism for clients to connect to an MRB to request media resources. In the case of the HTTP use, any binding using the Consumer interface MUST be capable of being transacted over TLS, as described in [RFC 2818](#) [[RFC2818](#)]. In the case of the SIP use, the same security considerations included in the Media Control Channel Framework specification apply here to protect interactions between a client requesting media resources and an MRB.

Should a valid session ID be compromised somehow (that is, intercepted or just guessed by a malicious user), as a further means to prevent disruption, the Consumer interface also prescribes the use of a sequence number in its transactions. This sequence number is to be increased after each successful transaction, starting from a first value randomly generated by the MRB when the session is first created, and must match in every request/response. While this adds complexity to the protocol (implementations must pay attention to those sequence numbers, since wrong values will cause "Wrong sequence number" errors and the failure of the related requests), it is an important added value for security. In fact, considering different transaction related to the same session could be transported in different, unrelated HTTP messages (or SIP INVITEs in case the Inline mode is being used), this sequence number protection prevents the chances of session replication or disruption, especially in cases where the session ID has been compromised: that is, it should make it harder for malicious users to manipulate or remove a session for which they have obtained the Session ID. It is strongly advised that MRB doesn't choose 1 as the first sequence number for a new session, but rather pick a random value to start from. The reaction to out of sequence transactions is left to MRB implementations: a related error code is available, but implementations may decide to enforce further limitations or actions upon the receipt of too many failed attempts in a row, or of what looks like blatant attempts to guess what the current, valid sequence number is.

It is also worth noting that in In-line mode (both IAMB and IUMB) the MRB may act as a Back-to-Back User Agent (B2BUA). This means that, as a B2BUA, the MRB may modify SIP bodies: it is the case, for instance, of the IAMB handling multipart/mixed payloads. This impacts the ability to use any SIP security feature that protects the

body (e.g., [RFC4474](#), s/mime, etc.) unless the MRB intermediates the security association. This should be taken into account when implementing an MRB compliant with this specification.

Both the Publishing and Consumer interface may address the location of a Media Server: the Publishing interface may be used to inform the MRB where a Media Server is located (approximately or precisely), and the Consumer interface to ask for a Media Server located somewhere in particular region (e.g., a conference bridge close to San Francisco). Both Media Server and MRB implementors need to take this into account when deciding whether or not make this location information available, and if so how much bits of information really need to be made available for brokering purposes.

It is worthwhile covering authorization issues related to this specification. Neither the Publishing or the Consumer interface provide an explicit means for implementing authentication, i.e., they do not contain specific protocol interactions to ensure that authorized Application Servers can make use of the services provided by an MRB instance. Considering both the interfaces are transported using well-established protocols (HTTP, SIP, CFW), support for such functionality can be expressed by means of the authentication mechanisms provided by the protocol themselves. Therefore, any MRB-aware entity (Application Servers, Media Servers, Media Resource Brokers themselves) MUST support the HTTP and SIP Digest access authentication. The usage of such Digest access authentications is recommended and not mandatory, which means MRB-aware entities MAY exploit it in deployment.

An MRB may want to enforce further constraints on the interactions between an Application Server/Media Server and an MRB. For example, it may choose to only accept requests associated with a specific session ID from the IP address that originated the first request, or just make use of pre-shared certificates to assess the identity of legitimate Application Server and/or Media Server.

13. IANA Considerations

There are several IANA considerations associated with this specification.

13.1. Media Control Channel Framework Package Registration

This section registers a new Media Control Channel Framework package, per the instructions in [Section 13.1 of \[RFC6230\]](#).

Package Name: mrbs-publish/1.0

Published Specification(s): RFCXXXX

Person and email address to contact for further information: IETF, MEDIACTRL working group, (mediactrl@ietf.org), Chris Boulton (chris@ns-technologies.com). [NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]

13.2. application/mrbs-publish+xml Media Type

To: application

Subject: Registration of media type application/mrbs-publish+xml

Type name: application

Subtype name: mrbs-publish+xml

Required parameters: none

Optional parameters: Same as charset parameter of application/xml as specified in [RFC 3023](#) [[RFC3023](#)].

Encoding considerations: Same as encoding considerations of application/xml as specified in [RFC 3023](#) [[RFC3023](#)].

Security considerations: See [Section 10 of RFC 3023](#) [[RFC3023](#)] and [Section 12](#) of RFCXXXX [[NOTE TO RFC-EDITOR/IANA: Please replace XXXX with the RFC number of this specification.]].

Interoperability considerations: none.

Published specification: [Section 10](#) of RFCXXXX [[NOTE TO RFC-EDITOR/IANA: Please replace XXXX with the RFC number of this specification.]].

Applications which use this media type: This media type is used to support a Media Resource Broker (MRB) entity.

Additional Information:

Magic Number: None

File Extension: .xdf

Macintosh file type code: "TEXT"

Personal and email address for further information: Chris Boulton
<chris at ns-technologies.com>

Intended usage: COMMON

Author/Change controller: The IETF.

13.3. application/mrb-consumer+xml Media Type

To: application

Subject: Registration of media type application/mrb-consumer+xml

Type name: application

Subtype name: mrb-consumer+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter of application/xml as specified in [RFC 3023](#) [[RFC3023](#)].

Encoding considerations: Same as encoding considerations of application/xml as specified in [RFC 3023](#) [[RFC3023](#)].

Security considerations: See [Section 10 of RFC 3023](#) [[RFC3023](#)] and [Section 12](#) of RFCXXXX [[NOTE TO RFC-EDITOR/IANA: Please replace XXXX with the RFC number of this specification.]].

Interoperability considerations: none.

Published specification: [Section 11](#) of RFCXXXX [[NOTE TO RFC-EDITOR/IANA: Please replace XXXX with the RFC number of this specification.]].

Applications which use this media type: This media type is used to support a Media Resource Broker (MRB) entity.

Additional Information:

Magic Number: None

File Extension: .xdf

Macintosh file type code: "TEXT"

Personal and email address for further information: Chris Boulton
<chris at ns-technologies.com>

Intended usage: COMMON

Author/Change controller: The IETF.

[13.4.](#) URN Sub-Namespace Registration for mrb-publish

Please register the URN "urn:ietf:params:xml:ns:mrb-publish", with the ID of "mrb-publish". The schema of the XML namespace named urn:ietf:params:xml:ns:mrb-publish" is [Section 10](#).

[13.5.](#) URN Sub-Namespace Registration for mrb-consumer

Please register the URN "urn:ietf:params:xml:ns:mrb-consumer", with the ID of "mrb-consumer". The schema of the XML namespace named urn:ietf:params:xml:ns:mrb-consumer" is in [Section 11](#).

[13.6.](#) XML Schema Registration for mrb-publish

Please register the schema for mrb-publish:

URI: urn:ietf:params:xml:schema:mrb-publish

ID: mrb-publish

Filename: mrb-publish

Registrant Contact: IETF, MEDIACTRL working group
(mediactrl@ietf.org)

Schema: The XML for the schema is in [Section 10](#) of this document.

[13.7.](#) XML Schema Registration for mrb-consumer

Please register the schema for mrb-consumer:

URI: urn:ietf:params:xml:schema:mrb-consumer

ID: mrb-consumer

Filename: mrb-consumer

Registrant Contact: IETF, MEDIACTRL working group
(mediactrl@ietf.org)

Schema: The XML for the schema is in [Section 11](#) of this document.

14. Changes

Note to RFC Editor: Please remove this whole section.

14.1. Changes from 13 Version

- o Clarified text for offer-less INVITE.
- o General review of text.

14.2. Changes from 12 Version

- o Several changes and clarifications according to the AD review by Robert Sparks.
- o Updated reference for mixer draft ([RFC6505](#)).

14.3. Changes from 11 Version

- o Fixed a wrong reference to [RFC5707](#) (because of a typo this was [RFC5705](#)).
- o Changed the registration in 13.1 to match the template required by [RFC6230](#).
- o Fixed the incorrect URIs for registering the schemas in Sections 13.6 and 13.7.
- o Removed enumeration types for 'dtmf-type', 'vxml-mode' and 'stream-mode' from both the schemas to allow for better extensibility, and clarified values are case insensitive where needed.
- o Clarified that the use of the civic location of a media server is entirely optional, and it's implementation specific to fill it with just the details each implementor deems necessary for any optimization that may be needed.

14.4. Changes from 10 Version

- o Editorial changes as a result of Shepherd review.
- o Added new attribute 'id' to both <mediaResourceRequest> and <mediaResourceResponse> elements in the consumer schema, in order to map a response to a specific request.
- o Renamed 'supported-actions' to 'supported-action' in the Publisher schema.

- o Removed 'support' attribute from both the <vxml-support> element (Publisher schema) and the <vxml> element (Consumer schema): now an empty element means no VXML support is provided/requested.
- o Clarified the scope of the 'application-data' element, and changed its type from xsd:NMTOKEN to xsd:string in the schema.
- o Clarified the use of the <subscription> element in an <mrbresponse>.
- o Clarified the meaning of TCP CONNECTION in sequence diagrams.
- o Removed useless backslashes from XML examples.
- o Updated references for Framework and IVR drafts ([RFC6230](#), [RFC6231](#)).

[14.5.](#) Changes from 09 Version

- o Language changes as a result of Shepherd review.

[14.6.](#) Changes from 08 Version

- o Fixed Nits.
- o Added range for reporting period - as per mailing list.

[14.7.](#) Changes from 07 Version

- o Corrected some errors in the Consumer schema: a few elements were not declared optional as they should have been, and some were incorrectly defined as choices instead of sequences;
- o Corrected examples after validation tests;
- o Fixed a few typos in the text.
- o Clarified language in various places.
- o Added 'Multi-modal MRB Implementations' section.
- o Added 'Relative Merits of Query Mode, IAMM, and IUmm' section.
- o Clarifying text related to IAMM and IUmm.
- o Expanded media-server-address for extra information and to allow multiples.

- o New B2BUA section.
- o Updated Examples.

14.8. Changes from 06 Version

- o Added the missing <encoding> and <decoding> elements to the <rtp-codec> instances, where needed.
- o Fixed a few typos in the text.

14.9. Changes from 05 Version

- o Clarifier that video layouts may refer to either XCON-defined layouts or others.
- o Added [RFC4240](#) as an option for VXML support.
- o Fixed a few typos in the text and in the schemas.

14.10. Changes from 04 Version

- o Corrected some typos and leftovers in both 'session-info' and 'response-session-info' definitions.
- o Clarified that 'response-session-info' is not only included in reply to updates, but also to new requests; besides, clarified that it is an optional element, in the sense that it is mandatory in successful responses (200), while not needed otherwise (any error).
- o Corrected the Query example flow which included a 'session'info' in a new request.

14.11. Changes from 03 Version

- o Addressed comments per the Expert RAI Review by Ben Campbell.
- o Several editorial changes (fixes, typos, nits).
- o Removed the 3xx class responses for the IAMM, per discussion in Anaheim (feature had been added in the -02 version).
- o Clarified that backslashes and XML indentation in the Examples are only provided for readability.
- o Clarified the distinction between 'deactivated' and 'unavailable'.

- o Added text to the status codes in both Publish and Consumer responses, in order to clarify when they are involved.
- o Added some text to better clarify the role of leasing in the Consumer interface.
- o Added additional IANA considerations, that were missing in the previous versions of the document.
- o Added text to the security considerations.

14.12. Changes from 02 Version

- o Added examples in [Section 9](#).
- o Fixed some nits in the schemas (encryption and required mixed=true elements).
- o Completed review nit review comments from Gary Munson.

14.13. Changes from 01 Version

- o Added description of lease mechanism.
- o Added specific HTTP and SIP usage of Consumer interface.
- o Completed Publish interface schema + associated text.
- o Included Consumer interface schema + associated text.
- o Included supported-packages element.
- o Removed announce-var element from doc.
- o Expanded Abstract.
- o General scrub of text - input from Simon Romano.
- o Added IANA Considerations section.
- o Added Security Considerations section.

14.14. Changes from 00 Version

- o Included In-line text based on strawman proposal.
- o Included first attempt at publish interface based on design team work.

15. Acknowledgements

The authors would like to thank the members of the Publish Interface design team who provided valuable input into this document. The design team consisted of Adnan Saleem, Michael Trank, Victor Paulsamy, Martin Dolly, and Scott McGlashan. The authors would also like to thank John Dally, Bob Epley, Simon Romano, Henry Lum, Christian Groves and Jonathan Lennox for input into this specification.

Ben Campbell carried out the RAI expert review on the -03 specification and provided a great deal of invaluable input.

16. References

16.1. Normative References

- [ISO.3166-1]
International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions - Part 1: Country codes", ISO Standard 3166-1:1997, 1997.
- [ISO.639.1988]
International Organization for Standardization, "Code for the representation of names of languages, 1st edition", ISO Standard 639, 1988.
- [ITU-T.Q.1950]
International Telecommunication Union - Telecommunication Standardization Bureau, "Call Bearer Control (CBC) Protocol", ITU-T Recommendation Q.1950, 2002.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", [RFC 2046](#), November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", [RFC 2616](#), June 1999.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", [RFC 3023](#), January 2001.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", [RFC 3261](#), June 2002.
- [RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", [RFC 3311](#), October 2002.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", [RFC 3711](#), March 2004.
- [RFC5139] Thomson, M. and J. Winterbottom, "Revised Civic Location Format for Presence Information Data Format Location

Object (PIDF-L0)", [RFC 5139](#), February 2008.

[W3C.CR-wsdl20-20051215]

Chinnici, R., Moreau, J., Ryman, A., and S. Weerawarana,
"Web Services Description Language (WSDL) Version 2.0 Part
1: Core Language", W3C CR CR-wsdl20-20051215,
December 2005.

[W3C.REC-soap12-part1-20030624]

Gudgin, M., Mendelsohn, N., Moreau, J., Nielsen, H., and
M. Hadley, "SOAP Version 1.2 Part 1: Messaging Framework",
World Wide Web Consortium FirstEdition REC-soap12-part1-
20030624, June 2003,
<<http://www.w3.org/TR/2003/REC-soap12-part1-20030624>>.

[W3C.REC-soap12-part2-20030624]

Hadley, M., Mendelsohn, N., Gudgin, M., Moreau, J., and H.
Nielsen, "SOAP Version 1.2 Part 2: Adjuncts", World Wide
Web Consortium FirstEdition REC-soap12-part2-20030624,
June 2003,
<<http://www.w3.org/TR/2003/REC-soap12-part2-20030624>>.

16.2. Informative References

- [RFC2818] Rescorla, E., "HTTP Over TLS", [RFC 2818](#), May 2000.
- [RFC4240] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network
Media Services with SIP", [RFC 4240](#), December 2005.
- [RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF
Digits, Telephony Tones, and Telephony Signals", [RFC 4733](#),
December 2006.
- [RFC5022] Van Dyke, J., Burger, E., and A. Spitzer, "Media Server
Control Markup Language (MSCML) and Protocol", [RFC 5022](#),
September 2007.
- [RFC5167] Dolly, M. and R. Even, "Media Server Control Protocol
Requirements", [RFC 5167](#), March 2008.
- [RFC5552] Burke, D. and M. Scott, "SIP Interface to VoiceXML Media
Services", [RFC 5552](#), May 2009.
- [RFC5567] Melanchuk, T., "An Architectural Framework for Media
Server Control", [RFC 5567](#), June 2009.
- [RFC5707] Saleem, A., Xin, Y., and G. Sharratt, "Media Server Markup
Language (MSML)", [RFC 5707](#), February 2010.

- [RFC6230] Boulton, C., Melanchuk, T., and S. McGlashan, "Media Control Channel Framework", [RFC 6230](#), May 2011.
- [RFC6231] McGlashan, S., Melanchuk, T., and C. Boulton, "An Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework", [RFC 6231](#), May 2011.
- [RFC6381] Gellens, R., Singer, D., and P. Frojdh, "The 'Codecs' and 'Profiles' Parameters for 'Bucket' Media Types", [RFC 6381](#), August 2011.
- [RFC6505] McGlashan, S., Melanchuk, T., and C. Boulton, "A Mixer Control Package for the Media Control Channel Framework", [RFC 6505](#), March 2012.

Authors' Addresses

Chris Boulton
NS-Technologies

Email: chris@ns-technologies.com

Lorenzo Miniero
Meetecho
Via Carlo Poerio 89
Napoli 80100
Italy

Email: lorenzo@meetecho.com

Gary Munson
AT&T
200 Laurel Avenue South
Middletown
New Jersey 07748
USA

Email: gamunson@att.com

