Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 26, 2008

C. Boulton
Avaya
T. Melanchuk
Rain Willow Communications
S. McGlashan
Hewlett-Packard

February 23, 2008

A Basic Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework
draft-boulton-ivr-control-package-06

Abstract

This document defines a Media Control Channel Framework Package for basic Interactive Voice Response (IVR) interaction.

Table of Contents

1. Introduction

The Media Control Channel Framework [MCCF] provides a generic approach for establishment and reporting capabilities of remotely initiated commands. The Framework utilizes many functions provided by the Session Initiation Protocol [RFC3261] (SIP) for the rendezvous and establishment of a reliable channel for control interactions. The Control Framework also introduces the concept of a Control Package. A Control Package is an explicit usage of the Control Framework for a particular interaction set. This document defines a Control Package for basic IVR dialogs.

This package has been designed to satisfy the IETF MediaCtrl requirements ([I-D.ietf-mediactrl-requirements]) by building upon two major approaches to IVR dialog design. These approaches address a wide range of IVR use cases and are used in many applications which are extensively deployed today.

First, the package is designed to provide the major IVR functionality of SIP Media Server languages such as netann ([RFC4240]), MSCML ([RFC5022]) and MSML ([MSML]) which themselves build upon more traditional non-SIP languages ([H.248.9], [RFC2897]). A key differentiator is that this package provides the functionality using the Media Control Channel Framework.

Second, its design is aligned with key concepts in W3C Voice Browser languages. The key dialog management mechanism is closely aligned with CCXML ([CCXML10]). The basic dialog functionality in this package can be seen as a subset of VoiceXML ([VXML20], [VXML21]): where possible, basic prompting, DTMF collection and media recording features are incorporated, but not any advanced VoiceXML constructs (such as <form>, its interpretation algorithm, or a dynamic data model). As W3C develops VoiceXML 3.0, we expect to see further alignment, especially in providing a set of basic independent primitive elements (such as prompt, collect and record) which can be re-used in different dialog languages.

By reusing and building upon design patterns from these approaches to IVR languages, this package is intended to provide a foundation which is familiar to current IVR developers and sufficient for 'basic' IVR applications, as well as a path to other languages - including control packages which extend this package - which address more advanced applications.

The scope of this package is 'basic' IVR functionality of a media server which includes:

- playing one or more media resources as a prompt to the user
- collecting DTMF input from the user according to a grammar
- recording user media input

Out of scope of this package are more advanced functions including ASR (Automatic Speech Recognition), TTS (Text-to-Speech), VoiceXML, fax and media transformation. Such functionality may be addressed by extension packages.

This functionality of this package is defined by messages, containing XML [XML] elements, transported using the Media Control Channel Framework. The XML elements can be divided into two types: dialog management elements; and carried within those, elements which define the specific IVR operations for the dialog.

Dialog management elements are designed to manage the general lifecycle of a dialog. Elements are provided for preparing a dialog, starting the dialog on a conference or connection, and terminating execution of a dialog. Each of these elements is contained in a Media Control Channel Framework CONTROL message sent to the media server. When the appropriate action has been executed, the media server sends a REPORT message (or a 200 if it can execute in time) with a response element indicating whether the operation was successful or not (e.g. if the dialog cannot be started, then the error is reported in this response). Once a dialog has been successfully started, the media server may send further event notifications in a framework CONTROL message. This package defines one event notification, namely a dialogexit event indicating that the dialog has exited. If the dialog has executed successful, the dialogexit event includes information collected during the dialog. If an error occurs during execution (e.g. a media resource failed to play, no recording resource available, etc), then error information is reported in the dialogexit event. Once a dialogexit event is sent, the dialog lifecycle is terminated.

Specific dialog types are referenced or contained within dialog management elements for preparing and starting dialogs. This package defines a basic IVR dialog type which contains child elements for playing prompts to the user, collects DTMF input from the user and recording media input from the user. The child elements can co-occur in the basic IVR dialog type so as to provide 'play announcement', 'prompt and collect' as well as 'prompt and record' functionality.

Implementation of this control package MUST adhere to the syntax and semantics of XML elements described in this document. In cases where there is a difference in constraints between the XML schema and the

textual description of elements, the textual definition takes priority.

Other control packages MAY extend this package. Two extension points are RECOMMENDED:

1. Retaining the dialog management capability but extending the basic IVR dialog type by adding new capabilities. For example, adding speech recognition capability.

2. Extending the dialog management capability with alternative dialog types. For example, using the VoiceXML dialog type rather than the basic IVR dialog type ([VXMLCP]).

Otherwise, extension control packages MUST respect the syntax and semantics of this control package.

This document specifies how the basic IVR control package fulfills the requirements of the Media Control Channel Framework control packages (Section 3), the dialog management elements defined by the package (Section 4), the basic IVR dialog type definitions (Section 5) as well as providing examples (Section 6, Section 7), type definitions (Section 8) and XML schema (Section 9).

2. Conventions and Terminology

In this document, BCP 14/RFC 2119 [RFC2119] defines the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL". In addition, BCP 15 indicates requirement levels for compliant implementations.

The following additional terms are defined for use in this document:

Dialog: A dialog performs media interaction with a user. A dialog is specified as inline XML, or via a URI reference to an external XML document type. Dialogs typically feature basic capabilities such as playing audio prompts, collecting DTMF input and recording audio input from the user. More advanced dialogs may also feature synthesized speech, recording and playback of video, recognition of spoken input, and mixed initiative conversations.

Application server: A SIP [RFC3261] application server (AS) hosts and executes services such as interactive media and conferencing in an operator's network. An AS influences and impacts the SIP session, in particular by terminating SIP sessions on a media server, which is under its control.

Media Server: A media server (MS) processes media streams on behalf of an AS by offering functionality such as interactive media, conferencing, and transcoding to the end user. Interactive media functionality is realized by way of dialogs which are initiated by the application server.

3. Control Package Definition

This section fulfills the mandatory requirement for information that MUST be specified during the definition of a Control Framework Package, as detailed in Section 8 of [MCCF].

3.1. Control Package Name

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

The name and version of this Control Package is "msc-ivr-basic/1.0" (Media Server Control - Interactive Voice Response - Basic - version 1.0). Its IANA registration is specified in Section 11.1.

3.2. Framework Message Usage

The Control Framework requires a Control Package to explicitly detail the control messages that can be used as well as provide an indication of directionality between entities. This will include which role type is allowed to initiate a request type.

This package defines XML elements for dialog management in Section 4; these elements specify dialog requests, responses and event notifications. The package also defined elements for the basic IVR dialog type in Section 5. The basic IVR dialog elements are contained inside dialog management elements. XML Schema for all XML elements defined in this package are specified in Section 9.

In this package, the MS operates as a Control Framework Server in receiving requests from, and sending responses to, the AS (operating as Control Framework Client). These requests are carried in CONTROL message bodies from AS to MS; <dialogprepare>, <dialogstart> and <dialogterminate> elements are defined as package requests. Responses to these requests are carried either in REPORT messages or Control Framework 200 response bodies with <response> element payload.

Note that package responses are different from framework response codes. Framework error response codes (see Section 8 of [MCCF]) are used when the request or event notification is invalid; for example, a request is invalid XML (400), or not understood (500). Package responses are carried in 200 response or REPORT message bodies. This package's response codes are defined in Section 4.2.1.

The MS also operates as a Control Framework Client in sending event notifications to the AS (Control Framework Server). Event notifications are carried in CONTROL message bodies; the <event>

element is defined for package event notifications. AS responses are carried in Control Framework responses or, if the transaction is extended, in REPORT messages.

3.3. Common XML Support

The Control Framework requires a Control Package definition to specify if the attributes for media dialog or conference references are required.

This package requires that the XML Schema in Section 16.1 of [MCCF] MUST be supported for media dialogs and conferences.

The package schema uses "connectionid" and "conferenceid" attributes which are imported from the XML schema.

3.4. CONTROL Message Body

The Control Framework requires a Control Package to define the control body that can be contained within a CONTROL command request and to indicate the location of detailed syntax definitions and semantics for the appropriate body types.

When operating as Control Framework Server, the MS receives CONTROL messages with a body containing a valid <dialogprepare>, <dialogstart> or <dialogterminate> request element. The syntax and semantics of these elements are defined in Section 4.1.

When operating as Control Framework Client, the MS sends CONTROL messages with a body containing a valid notification <event> element. The syntax and semantics of this element is defined in Section 4.3.

3.5. REPORT Message Body

The Control Framework requires a control package definition to define the REPORT body that can be contained within a REPORT command request, or that no report package body is required. This section should indicate the location of detailed syntax definitions and semantics for the appropriate body types.

When operating as Control Framework Server, the MS sends REPORT bodies containing a valid <response> element. The syntax and semantics of this element is specified in Section 4.2.

When operating as Control Framework Client, the MS receives REPORT messages which SHOULD NOT contain a body.

[Editors Note:IVR01 need to clarify the behavior when the MS receives

a REPORT in response to sending an event notification CONTROL.]

4. Dialog Management Element Definitions

This section defines the dialog management XML messages for this control package. These messages are divided into requests (Section 4.1), responses (Section 4.2) and notifications (Section 4.3).

4.1. Requests

Requests are send to the MS (operating as a Control Framework Server) in the body of CONTROL messages.

The following request elements are defined:

<dialogprepare>: prepare an IVR dialog for later execution

<dialogstart>: start an IVR dialog on a connection or conference

<dialogterminate>: terminate an IVR dialog

These elements control the life cycle of a dialog. Each dialog has the following states:

IDLE: the dialog is uninitialized.

PREPARING: the dialog is being prepared. If an error occurs the dialog transitions to the TERMINATED state.

PREPARED: the dialog has been successfully prepared and has a valid dialog identifier.

STARTING: the dialog is being started. If an error occurs the dialog transitions to the TERMINATED state.

STARTED: the dialog has been successfully started and is now active. When the dialog exits (normally or terminated), or due to an error, a dialogexit notification event is sent and the dialog transitions to the TERMINATED state.

TERMINATED: the dialog is terminated and its dialogid is no longer valid. No further dialog notifications are sent for this dialog.

As a consequence, it is an error if the same dialog is prepared or started more than once.

4.1.1. <dialogprepare>

The <dialogprepare> request is sent to the MS to request preparation of an IVR dialog. Dialog preparation consists of (a) retrieving an external dialog document (if specified), and (b) validating the document both syntactically and semantically.

A prepared dialog is executed when the AS sends a <dialogstart> request referencing the prepared dialog (see Section 4.1.2).

A <dialogprepare> element has the following attributes:

src: specifies the location of an external dialog document to prepare. A valid value is a URI (see Section 8.9). It is an error if the document cannot be retrieved or processed (e.g. URI protocol not supported). The attribute is optional.

type: specifies the type of the external dialog document indicated in the 'src' attribute. A valid value is a MIME type (see Section 8.10). The attribute is optional.

dialogid: string indicating a unique name for the dialog. If this attribute is not specified, the MS creates a unique name for the dialog. The value is used in subsequent references to the dialog (e.g. as dialogid in a <response>). It is an error if a dialog with the same name already exists on the MS. The attribute is optional.

The <dialogprepare> element in this package is extended with the the child element <basicivr> for inline Basic IVR dialog (see Section 5.1). Other packages which extend this package MAY use alternative dialog types.

The dialog to prepare can either be specified inline as a child element or externally using the src attribute (but not both). It is an error if both an inline dialog element and a src attribute are specified.

When an MS receives a <dialogprepare> request, the dialog is in a PREPARING state. If dialog preparation was successful, the dialog is in a PREPARED state; otherwise the dialog is in a TERMINATED state. The MS MUST reply to <dialogprepare> request with a <response> element (Section 4.2), reporting whether dialog preparation was successful or not.

Example: in CONTROL message from AS to MS, the following request prepares a basic IVR dialog where a single prompt is played once:

```
<dialogprepare>
  <basicivr>
     <prompt>
      <media src="http://www.example.com/prompt1.wav"/>
     </prompt>
  </basicivr>
</dialogprepare>
```

Alternatively, the same dialog could be referenced:

```
<dialogprepare type="application/msc-ivr-basic+xml"
   src="http://example.org/basic1.xml"/>
```

if the dialog is prepared successful, then a Control Framework 200 or a REPORT message for the same transaction would contain a response such as:

```
<response status="200" dialogid="d100"/>
```

The Control Framework transaction is then complete.

See Section 6 and Section 7 for further examples.

4.1.2. <dialogstart>

The <dialogstart> element is sent by the AS to start a dialog. The dialog may be specified in the <dialogstart> request itself, or reference a previously prepared dialog.

Starting a dialog consists of (a) retrieving any resources (e.g. media, grammar, etc) associated with the dialog, and (b) activating these resources according to the dialog type. For example, with <basicivr>, retrieving media resources and playing them to the user.

The <dialogstart> element has the following attributes:

src: specifies the location of an external dialog document to start. A valid value is a URI (see Section 8.9). It is an error if the document cannot be retrieved or processed (e.g. URI protocol not supported). The attribute is optional.

type: specifies the type of the external dialog document indicated in the 'src' attribute. A valid value is a MIME type (see Section 8.10). The attribute is optional.

dialogid: string indicating a unique name for the dialog. If this attribute is not specified, the MS creates a unique name for the dialog. The value is used in subsequent references to the dialog (e.g. as dialogid in a <response>). It is an error if a dialog with the same name already exists on the MS. The attribute is optional.

prepareddialogid: string identifying a dialog previously prepared using a dialogprepare request. The attribute is optional.

connectionid: string identifying the SIP dialog connection on which this dialog is to be started (see Section 16.1 of [MCCF]). The attribute is optional.

conferenceid: string identifying the conference on which this dialog is to be started (see Section 16.1 of [MCCF]). The attribute is optional.

Exactly one of the connectionid or conferenceid attributes MUST be specified. It is an error to specify both connectionid and conferenceid attributes or neither.

The <dialogstart> element has the following child elements defined:

<stream>: contains the following attributes:

media: a string indicating the type of media associated with the stream. It is strongly RECOMMENDED that the following values are used for common types of media: "audio" for audio media, and "video" for video media. The attribute is mandatory.

label: a string indicating the SDP label associated with a media stream ([RFC4574]). The attribute is optional.

direction: a string indicating the direction of the media flow between a dialog and its end point conference or connection. Defined values are: "sendrecv" (media can be sent and received), "sendonly" (media can only be sent), and "recvonly" (media can only be received). The default value is "sendrecv". The attribute is optional.

One or more <stream> elements may be specified so that individual media streams can be controlled independently; for example, audio only for transmission, but video only for reception. The <stream> element is optional. If no <stream> elements are specified, then the default is the media configuration of the connection or conference. It is an error if a <stream> element is in conflict with (a) another <stream> element, (b) with dialog, connection or

conference media capabilities, or (c) with a SDP label value as part of the connectionid (see Section 16.1 of [MCCF]).

The <dialogstart> element in this package is extended with the the child element <basicivr> for inline Basic IVR dialog (see Section 5.1). Other packages which extend this package MAY use alternative dialog types.

The dialog to start can be specified either inline, or externally, or reference a previously prepared dialog. Exactly one of the src attribute, the prepareddialogid or a dialog type child element MUST be specified. If the prepareddialogid is specified, it is an error to specify the src attribute, the dialogid attribute or a dialog type child element. If the src attribute is specified, it is an error to specify the prepareddialogid attribute, or a dialog type child element. If a dialog type child element is specified, it is an error to specify the src attribute or the prepareddialogid attribute.

When an MS receives a <dialogstart> request, the dialog is either in an IDLE or PREPARED state (if a previously prepared dialog is being started). If in the IDLE state, then the dialog is first prepared and if successful, transitions to the PREPARED state. When the prepared dialog is started it transitions to the STARTING state and, if the dialog is started successfully, the dialog then transition to the STARTED state. If a error occurs during dialog preparation or starting, then the error MUST be reported, and the dialog transitions to a TERMINATED state. The MS MUST reply to <dialogstart> request with a <response> element (Section 4.2), reporting whether the dialog was started successful or not.

[Editors Note:IVR02 This specification does not defined the behavior when more than one dialog is started on the same connection or conference. Various models could be applied:

reject: only one dialog per connection or conference

replace: the current dialog is stopped and the new one started

queue: the new dialog is queued for starting after current dialog exits.

mix: multiple dialogs are permitted with the same connection or conference: input is passed to both dialog; and dialog output is mixed.

We also need to take into account that the <stream> element can be used to specify that different dialogs affect different media streams of the same connection or conference. For example, one dialog that

only generates output, and another dialog on the same connection only receives input. Workgroup input on this issue is required.]

Example: in CONTROL message from AS to MS, the following request starts a basic ivr recording dialog on a conference:

```
 <dialogstart conferenceid="conference11">
    <basicivr>
      <record maxtime="384000s"/>
    </basicivr>
 </dialogstart>
```

Alternatively, the same dialog could be referenced:

```
 <dialogstart conferenceid="conference11"
       type="application/msc-ivr-basic+xml"
       src="http://example.org/basic2.xml"/>
```

if the dialog is started successfully, then a Control Framework 200 or a REPORT message for the same transaction would contain a response such as:

```
<response status="200" dialogid="d101" conferenceid="conference11"/>
```

The Control Framework transaction is then complete.

See Section 6 and Section 7 for further examples.

4.1.3. <dialogterminate>

A dialog that has been successfully prepared or started can be terminated by sending a <dialogterminate> request element to the MS.

The <dialogterminate> element has the following attributes:

dialogid: string identifying the dialog to terminate. The attribute is mandatory.

immediate: indicates whether the dialog is to be terminated immediately or not. A valid value is a boolean (see Section 8.1). A value of true indicates that the dialog is terminated and a dialogexit <event> notification MUST be sent immediately. A value of false indicates that the dialog terminates normally, sending a dialogexit <event> notification is sent when the dialog has exited. The attribute is optional. The default value is false.

It is an error if the dialogid is invalid.

When an MS receives a <dialogterminate> request, the dialog MUST be in a PREPARED, STARTING or STARTED states. If it is in a PREPARED state, then it transitions immediately to the TERMINATED state. If it is in STARTING state, then any further starting (or preparation) of the dialog is canceled, the response of the dialogstart command reporting that the dialog is terminated, and the dialog transitions to the TERMINATED state. If the dialog is in a STARTED state, then the dialog is terminated according to the value of the immediate attribute, and when the dialogexit notification is sent, the dialog moves into the TERMINATED state.

The MS MUST reply to <dialogterminate> request with a <response> element (Section 4.2), reporting whether the dialog was stopped successful or not.

Example: in CONTROL message from AS to MS, the following request terminate a dialog with dialogid "vxi1":

```
<dialogterminate dialogid="vxi1" immediate="true"/>
```

if the dialog is terminated successfully, then a Control Framework 200 or a REPORT message for the same transaction would contain a response such as:

```
<response status="200" dialogid="d100"/>
```

The Control Framework transaction is then complete.

See Section 6 and Section 7 for further examples.

4.2. Responses

Responses are generated in response to <dialogprepare>, <dialogstart> and <dialogterminate> requests. Responses are specified in a <response> element.

4.2.1. <response>

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

The <response> element has following attributes:

status: numeric code indicating the response status. The attribute is mandatory.

reason: string specifying a reason for the response status. The attribute is optional.

dialogid: string identifying the dialog. The attribute is optional.

connectionid: string identifying the SIP dialog connection associated with the dialog (see Section 16.1 of [MCCF]). The attribute is optional.

conferenceid: string identifying the conference associated with the dialog (see Section 16.1 of [MCCF]). The attribute is optional.

The following status codes are defined:

code	description
200	OK
401	dialogid already exists
402	dialogid does not exist
403	connectionid does not exist
404	conferenceid does not exist
405	Unknown or unsupported element
406	Element required
407	Unknown or unsupported attribute
408	Attribute required
409	dialog type not supported
410	Retrieving document resource failed
411	Invalid attribute value
499	other error

Table 1: <response> status codes

[Editors Note:IVR03 more status codes may need to be defined.]

For example, a response when a dialog was prepared successfully:

```
<response status="200" dialogid="vxi1"/>
```

The response if dialog preparation failed due to an unsupported dialog type:

```
 <response status="409" dialogid="vxi1"
    reason="Unsupported dialog type: application/basicirv+xml"/>
```

For example, a response when the dialog was started successfully.

```
 <response status="200" dialogid="vxi1"
              connectionid="7HDY839~HJKSkyHS"/>
```

See Section 6 and Section 7 for further examples.

4.3. Notifications

Event notifications are specified in an <event> element.

Event notifications MUST NOT be sent unless the dialog is in a STARTED state.

4.3.1. <event>

When a dialog generates a notification event, the MS sends the event to the AS using an <event> element.

The <event> element has the following attributes:

dialogid: string identifying the dialog which generated the event. The attribute is mandatory.

The <event> element has the following child elements:

<dialogexit>: indicates that the dialog has exited. The element is optional.

Extension packages may define other child elements for notifications which they support.

See Section 6 and Section 7 for examples.

4.3.2. <dialogexit>

The <dialogexit> event indicates that an active dialog has exited because it is complete, has been terminated, or because an error

occurred during execution (for example, a media resource cannot be played). This event MUST be sent by the MS when the dialog exits. Once the dialogexit event is sent, the dialog transitions from the STARTED state to the TERMINATED state. The MS MUST NOT send any further notifications for a dialog in the TERMINATED state.

The <dialogexit> element has the following attributes:

status: a status code indicating success or failure of the dialog. A valid value is a non-negative integer (see Section 8.4). A value of 0 indicates that the dialog has been terminated externally. A value of 1 indicates success. Any other value indicate an error. The attribute is mandatory.

reason: a textual description providing a reason for the status code; e.g. details about an error. A valid value is a string (see Section 8.6). The attribute is optional. There is no default value.

[Editors Note:IVR04 do we need to specify specific error codes?]

The <dialogexit> element in this package is extended with child elements for reporting prompt, collect and record information (see Section 5.5) of the basic IVR dialog type. Other packages which extend this package MAY use alternative child elements.

Example: when a STARTED <basicivr> dialog exits normally the MS sends a dialogexit <event>:

```
<event dialogid="vxi1"/>
 <dialogexit status="1">
   <collectinfo dtmf="1234"  termmode="match"/>
 </dialogexit>
```

See Section 6 and Section 7 for further examples.

5. Basic IVR Dialog Element Definitions

The basic IVR dialog is an instance of a dialog described in Section 4.1. The MS MUST support the Basic IVR dialog type in this package.

A basic IVR dialog is specified inline or by reference in <dialogprepare> (Section 4.1.1) or <dialogstart> (Section 4.1.2) elements. Inline basic ivr dialogs are specified in a <basicivr> child of these elements.

The basic IVR dialog is a simple XML container - <basicivr> - for executing basic IVR operations of playing prompts (<prompt> - see Section 5.2), collecting DTMF (<collect> - see Section 5.3),and recording user input (<record> - see Section 5.4. Results of the dialog execution are reported in a dialogexit notification event.

Three execution models are defined:

playannouncements: only a <prompt> element is specified in the container. The prompt media resources are played in sequence.

promptandcollect: a <collect> element is specified and, optionally, a <prompt> element. If a <prompt> element is specified and bargein is enabled, playing of the prompt is terminated when bargein occurs, and DTMF collection is initiated; otherwise, the prompt is played to completion before DTMF collection is initiated. If no prompt element is specified, DTMF collection is initiated immediately.

promptandrecord: a <record> element is specified and, optionally, a <prompt> element. If a <prompt> element is specified and bargein is enabled, playing of the prompt is terminated when bargein occurs, and recording is initiated; otherwise, the prompt is played to completion before recording is initiated. If no prompt element is specified, recording is initiated immediately.

Each of the core elements - <prompt>, <collect> and <record> - are specified so that their execution and reporting is largely self-contained. This facilitates their re-use in other dialog container elements.

Execution results are reported in the <dialogexit> notification event whose definition is extended with the elements defined in Section 5.5. If the dialog terminated normally (i.e. not due to an error or to a <dialogterminate> request), then the MS MUST report the results for at least the core operation:

<prompt> only: <promptinfo> (see Section 5.5.1) with at least the termmode attribute specified.

<collect>: <collectinfo> (see Section 5.5.2) with the dtmf and termmode attributes specified.

<record>: <recordinfo> (see Section 5.5.3) with at least the recording, type and termmode attributes specified.

The media format requirements for basic IVR dialogs are undefined. This package is agnostic to the media types and codecs for media resources and recording which need to be supported by an implementation. For example, a MS implementation may choose to support only audio and in particular the 'audio/basic' codec for media playback and recording. However, if an MS encounters when executing a dialog a media type or codec which it cannot process, the MS MUST stop further processing and report an error to the AS using the dialogexit notification.

The implementation of basic IVR dialogs MUST adhere to the syntax and semantics described in this section. Implementations MAY support additional (non-basicivr namespace) attributes and elements of these dialogs. If an implementation encounters additional (non-basicivr namespace) elements or attributes which it cannot process, it MUST ignore them and continue processing.

[Editors Note:IVR05 One use case which is not covered by this specification is that of attaching a dialog to a connection, which itself is attached to a conference, and allowing that dialog to repeatedly collect and return matching DTMF without terminating. The current specification requires that the dialog exits at the end of its DTMF collection. For the next collection cycle, a new dialog would need to be started. If the use case is in scope for this package, then the definition of <basicivr> could be modified as follows:

1. A signalmode attribute/element is added to <basicivr> indicating the conditions under which the dialog sends a (non-terminating) notification event (TBD). If any of its child elements completes with a termmode value matching a condition (e.g. a 'match' from <collect>), then information collected by the children is returned to the AS in a notification event. The dialog is always restarted and it needs to be terminated by the AS (unless an error occurs).

2. If the signalmode attribute/element is not specified, then the behavior is as currently defined.

3. Note that if media prompts were specified, the same prompts would play repeatedly - a more sophisticated approach would be required to play different prompts on different termmodes like nomatch, noinput, etc

Working Group input is needed to determined if this use case is in scope for this package and whether this type of solution is adequate.]

5.1. <basicivr>

A dialog to play prompts to the user, collect DTMF or record input. The dialog is specified using a <basicivr> element with the namespace defined in Section 11.2.

A <basicivr> element has the following attributes:

version: a string specifying the version of the basicivr package. The attribute is optional. The default value is '1.0'.

The <basicivr> element has the following child elements:

<prompt>: defines media resources to play in sequence (see Section 5.2). The element is optional.

<collect>: defines how DTMF is collected (see Section 5.3). The element is optional.

<record>: defines how recording takes place (see Section 5.4). The element is optional.

It is an error if no child element is specified. The behavior is not defined if both <collect> and <record> are specified.

5.2. <prompt>

The <prompt> element specifies a sequence of media resources to play.

A <prompt> element has the following attributes:

xml:base: A string declaring the base URI from which relative URIs in child elements are resolved. A valid value is a URI (see Section 8.9). The attribute is optional.

bargein: Indicates whether user input stops prompt playback. A valid value is a boolean (see Section 8.1). A value of true indicates that bargein is permitted and prompt playback is stopped. A value of false indicates that bargein is not

permitted: user input does not terminate prompt playback. The attribute is optional. The default value is true.

iterations: number of times the prompt is to be played. A valid value is a non-negative integer (see Section 8.4). A value of 0 indicates that the prompt is repeated until halted by other means. The attribute is optional. The default value is 1.

duration: maximum duration for the prompt playback. A valid value is a Time Designation (see Section 8.7). If no value is specified, then there is no limit on the duration of the prompt. The attribute is optional. There is no default value.

volume: playback volume for the prompt. A valid value is a percentage (see Section 8.4). The value indicates increase or decrease relative to the original recorded volume of the media. A value of 100% (the default) plays the media at its recorded volume, a value of 200% will play the media twice recorded volume, 50% at half its recorded volume, and so on. The attribute is optional. The default value is 100%.

offset: offset to begin playing the prompt. A valid value is a Time Designation (see Section 8.7). The offset is measured in normal media playback time from the beginning of the media resource. The attribute is optional. The default value is 0s.

The duration attribute takes priority over the iterations attribute in determining maximum duration of the prompt.

[Editors Note:IVR06 iterations could be replaced with 'repeatCount', duration with 'repeatDur', volume could be replaced with 'soundLevel' and offset with 'clipBegin' for better alignment with SMIL and possibly VoiceXML.]

[Editors Note:IVR07 should volume use a linear or logarithmic scaling?]

The <prompt> element has the following child elements:

<media>: media resource (see Section 5.2.1) to play. Multiple instances of this element are permitted. The element is optional.

<variable>: specifies a variable media announcement (see Section 5.2.2) to play. Multiple instances of this element are permitted. The element is optional.

<dtmf>: generates one or more DTMF tones (see Section 5.2.3) to play. Multiple instances of this element are permitted. The element is optional.

It is an error if no child element is specified.

Prompt playing has the following execution model upon initialization:

1. If an error occurs during execution, then playback terminates and the error is reported in the status attribute (see Section 5.5) of the <dialogexit> event.

2. A iterations counter is initialized to 0.

3. A timer is started for the value of the duration attribute. If the timer expires before playback is complete, then playback terminates and the dialogexit result contains the <promptinfo> termmode attribute set to maxduration (see Section 5.5.1).

4. A playback cycle is initiated playing each <media>, <variable> and <dtmf> in document order. The value of the volume attribute is applied to each <media> and <variable> if they are rendered as audio media. If the offset attribute is specified, playback begins at that offset. If the offset is greater than the cumulative duration of the child elements, then no prompts are played in the cycle.

5. If the value of the bargein attribute is true, then user input causes playback to terminate and the dialogexit result contains the <promptinfo> termmode attribute is set to bargein (see Section 5.5.1).

6. If the playback cycle completes successfully, then the iterations counter is updated. If the counter reaches the value of the iterations attribute, then playback is terminated and the dialogexit result contains a <promptinfo> termmode attribute set to completed (see Section 5.5.1). Otherwise, another playback cycle is initiated.

[Editors Note:IVR08 Need to clarify that DTMF used for VCR controls does not cause playback to be stopped, but causes the appropriate operation to be applied to the playing prompts.]

5.2.1. <media>

The <media> element specifies a media resource to play.

A <media> element has the following attributes:

src: specifies the location of the media resource. A valid value is a URI (see Section 8.9). The attribute is mandatory.

type: specifies the type of the media resource indicated in the 'src' attribute. The type value may include additional parameters for guiding playback; for example, [RFC4281] defines a 'codec' parameter for 'bucket' media types like video/3gpp. A valid value is a MIME type (see Section 8.10). The attribute is optional. There is no default value.

The <media> element has no children.

It is an error if the media resource cannot be retrieved or played.

5.2.2. <variable>

The <variable> element specifies variable announcements using predefined media resources. Each variable has at least a type (e.g. date) and a value (e.g. 2008-02-25). The value is rendered according to the variable type (e.g. 25th February 2008) as well as other defined attributes.

A <variable> element has the following attributes:

value: specifies the string to be rendered. A valid value is a string (see Section 8.6). The attribute is mandatory.

type: specifies the type to use for rendering. A valid value is a string (see Section 8.6). The attribute is mandatory.

format: specifies format information to use in conjunction with the type for the rendering. A valid value is a string (see Section 8.6). The attribute is optional. There is no default value.

xml:lang: specifies the language to use when rendering the variable. A valid value is a language identifier (see Section 8.11). The attribute is optional. The default value is platform-specific.

[Editors Note:IVR09 do we need a gender attribute for variable announcements?]

The <variable> element has no children.

This package is agnostic to which <variable> values, types and formats are supported by an implementation. However it is RECOMMENDED that an implementation support the following type/format combinations:

type=date Supported formats: "mdy", "ymd", "dym"

type=time Supported formats: "t12", "t24"

type=digits Supported formats: "gen", "ndn", "crn", "ord"

[Editors Note:IVR10 Further work needed on these definitions. Which other types and formats need to, or can, be specified?]

This specification is agnostic to the type and codec of media resources into which variable are rendered. For example, an MS choosing to support audio may render the <variable> into one or more audio media resources.

It is an error if a <variable> element cannot be rendered successfully.

Execution of this element can seen, at least logically, as conversion of a <variable> into a list of <media> elements. For example,

```
<variable value="2008-02-25" type="date" format="dmy"
xml:lang="en"/>
```

could be transformed into audio saying "twenty-fifth of February two thousand and eight" using a list of <media> resources:

```
<media src="voicebase/en/25th.wav"/>
<media src="voicebase/en/of.wav"/>
<media src="voicebase/en/february.wav"/>
<media src="voicebase/en/2000.wav"/>
<media src="voicebase/en/and.wav"/>
<media src="voicebase/en/8.wav"/>
```

5.2.3. <dtmf>

The <dtmf> element specifies a sequence of DTMF tones for output.

DTMF tones could be generated using <media> resources where the output is transported as RTP audio packets. However, <media> resources are not sufficient for cases where DTMF tones are to be transported as DTMF RTP ([RFC2833]) or in event packages.

A <dtmf> element has the following attributes:

digits: specifies the DTMF sequence to output. A valid value is a DTMF string (see Section 8.3). The attribute is mandatory.

level: used to define the power level for which the DTMF tones will be generated. Values are expressed in dBm0. A valid value is an integer in the range of 0 to -96 (dBm0). Larger negative values express lower power levels. Note that values lower than -55 dBm0 will be rejected by most receivers (TR-TSY-000181, ITU-T Q.24A). The attribute is optional. The default value is -6 (dBm0).

duration: specifies the duration for which each DTMF tone is generated. A valid value is a time designation (see Section 8.7). Implementations may round the value if they only support discrete durations. The attribute is optional. The default value is 100ms.

interval: specifies the duration of a silence interval following each generated DTMF tone. A valid value is a time designation (see Section 8.7). Implementations may round the value if they only support discrete durations. The attribute is optional. The default value is 100ms.

The <dtmf> element has no children.

It is an error if a <dtmf> element cannot be processed successfully.

5.3. <collect>

The <collect> element defines how DTMF input is collected.

The <collect> element has the following attributes:

cleardigitbuffer: indicates whether the digit buffer is to be cleared. A valid value is a boolean (see Section 8.1). A value of true indicates that the digit buffer is to be cleared. A value of false indicates that the digit buffer is not to be cleared. The attribute is optional. The default value is true.

timeout: indicates the time to wait for user input. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 5s.

interdigittimeout: indicates inter-digit timeout value to use when recognizing DTMF input. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 2s.

termtimeout: indicates the terminating timeout value to use when recognizing DTMF input. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 0s.

escapekey: specifies a DTMF key that indicates the DTMF collection is to be re-initiated. A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

termchar: specifies a DTMF character for terminating DTMF input collection using the internal grammar. A valid value is a DTMF character (see Section 8.2). To disable termination by a conventional DTMF character, set the parameter to an unconventional character like 'A'. The attribute is optional. The default value is '#'.

maxdigits: The maximum number of digits to collect using an internal digits (0-9 only) grammar. A valid value is a positive integer (see Section 8.5). The attribute is optional. The default value is 5.

skipinterval: indicates how far a MS should skip backwards or forwards through prompt playback when the rewind (rwkey) of fast forward key (ffkey) is pressed. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 6s.

ffkey: maps a DTMF key to a fast forward operation equal to the value of 'skipinterval'. A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

rwkey: maps a DTMF key to a rewind operation equal to the value of 'skipinterval'. A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

pauseinterval: indicates how long a MS should pause prompt playback when the pausekey is pressed. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 10s.

pausekey: maps a DTMF key to a pause operation equal to the value of 'pauseinterval'. A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

resumekey: maps a DTMF key to a resume operation. A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

volumeinterval: indicates the increase or decrease in playback volume (relative to the current volume) when the volupkey or voldnkey is pressed. A valid value is a percentage (see Section 8.8). The attribute is optional. The default value is 10%.

volupkey: maps a DTMF key to a volume increase operation equal to the value of 'volumeinterval'. A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

voldnkey: maps a DTMF key to a volume decrease operation equal to the value of 'volumeinterval'.A valid value is a DTMF Character (see Section 8.2). The attribute is optional. There is no default value.

It is an error if any VCR key (ffkey, rwkey, pausekey, resumekey, volupkey, voldnkey) is specified with the same value except that the pausekey and resumekey may have the same value.

[Editors Note:IVR11 Do we need a speed VCR control as well?]

[Editors Note:IVR12 it has been proposed that termchar should be replaced by termkeys (DTMF string value) to allow termination by multiple characters (e.g. '**'). An additional attribute would probably also be need to control interdigit timing within this string. Is this functionality required for this package?]

The <collect> element has the following child elements:

<grammar>: indicates a custom grammar format (see Section 5.3.1). The element is optional.

The custom grammar takes priority over the internal grammar. If a <grammar> element is specified, it MUST be used for DTMF collection.

DTMF collection has the following execution model upon initialization:

1. If an error occurs during execution, then collection terminates and the error is reported in the status attribute (see Section 5.5) of the <dialogexit> event.

2. The digit buffer is cleared if the value of the cleardigitbuffer attribute is true.

3. A timer with the duration of the value of the timeout attribute is activated. If the timer expires before DTMF input collection

has completed, then collection execution terminates and the dialogexit result contains a <collectinfo> (see Section 5.5.2) with the termmode attribute set to noinput.

4. If DTMF input matches any VCR keys (for example the ffkey), then the appropriate operation is applied to the active prompts; this is a no-op if a prompt is not specified, or there are no active prompts. If a seek operation (ffkey, rwkey) attempts to go beyond the beginning or end of the prompt queue, then it is automatically truncated to the prompt beginning or end respectively. If the pause operation attempts to pause output when it is already paused, then the operation is ignored. If the resume operation attempts to resume when the prompts are not paused, then the operation is ignored. If a volume operations attempts to go beyond the minimum or maximum volume supported by the platform, then the operation is ignored. DTMF input matching VCR controls is ignored for grammar matching.

5. If DTMF input matches the value of the escapekey attribute, then the timer is canceled and DTMF collection is re-initialized.

6. Other DTMF input is matched to the grammar. Valid DTMF patterns are either a simple digit string where the maximum length is determined by the maxdigits attribute and may be terminated by the character in the termchar attribute; or a custom DTMF grammar specified with the <grammar> element. The attributes interdigittimeout and termtimeout control interdigit timeout and the terminating timeout respectively.

7. If the input completely matches the grammar, the timer is canceled, collection execution terminates and the dialogexit result contains a <collectinfo> (see Section 5.5.2) with the termmode attribute set to match.

8. If the input does not match the grammar, the timer is canceled, collection execution terminates and the dialogexit result contains a <collectinfo> (see Section 5.5.2) with the termmode attribute set to nomatch.

5.3.1. <grammar>

The <grammar> element allows a custom grammar, inline or external, to be specified. Custom grammars permit the full range of DTMF characters including '*' and '#' to be specified for DTMF pattern matching.

The <grammar> element has the following attributes:

src: specifies the location of an external grammar. A valid value is a URI (see Section 8.9). The attribute is optional. There is no default value.

type: identifies the preferred type of the grammar document identified by the src attribute. A valid value is a MIME type (see Section 8.10). The attribute is optional. There is no default value.

The <grammar> element allows inline grammars to be specified. XML grammar formats MUST use a namespace other than the one used in this specification. Non-XML grammar formats MAY use a CDATA section.

The MS MUST support the [SRGS] grammar format and MS MAY support KPML ([RFC4730]) or other grammar formats.

It is an error if a grammar format is specified which is not supported by the MS.

[Editors Note:IVR13 One drawback of using <grammar> with an inline custom grammar is that nested <grammar> elements are needed for SRGS: i.e. custom <grammar> element and inside that a <grammar> element in the SRGS namespace. The alternative would be (a) <grammar> element is only for external grammars, and (b) inline grammars are specified as children of <collect> (with suitable co-occurrence restrictions). Feedback is required to determined if the nesting is confusing or not.]

Example: in CONTROL message from AS to MS, the following request starts a dialog where DTMF collection is determined by an inline SRGS grammar:

```
<dialogstart connectionid="con1">
  <basicivr>
   <collect cleardigitbuffer="false" timeout="20s"
        interdigittimeout="1s" skipinterval="10s"
    rwkey="4" ffkey="6">
        <grammar>
                <grammar xmlns="http://www.w3.org/2001/06/grammar"
                               version="1.0" mode="dtmf">
                  <rule id="digit">
                   <one-of>
                        <item>0</item>
                        <item>1</item>
                        <item>2</item>
                        <item>3</item>
                        <item>4</item>
                        <item>5</item>
                        <item>6</item>
                        <item>7</item>
                        <item>8</item>
                        <item>9</item>
                   </one-of>
                </rule>

                <rule id="pin" scope="public">
                 <one-of>
                        <item>
                         <item repeat="4">
                          <ruleref uri="#digit"/>
                                </item>#</item>
                                <item>* 9</item>
                 </one-of>
                </rule>

       </grammar>
         </grammar>
        </collect>
   </basicivr>
</dialogstart>
```

The same grammar could also be referenced externally (and take advantage of HTTP caching):

```
 <dialogstart connectionid="con1">
    <basicivr>
                <collect cleardigitbuffer="false" timeout="20s"
                        interdigittimeout="1s" skipinterval="10s"
            rwkey="4" ffkey="6">
                        <grammar type=""application/srgs+xml"
                     src="http://example.org/pin.grxml/>
                </collect>
    </basicivr>
 </dialogstart>
```

See Section 6 and Section 7 for further examples.

5.4. <record>

The <record> element defines how media input is recorded.

The <record> element has the following attributes:

timeout: indicates the time to wait for user input. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 5s.

type: specifies the type of the resulting recording format. The type value may include additional parameters for guiding recording; for example, [RFC4281] defines a 'codec' parameter for 'bucket' media types like video/3gpp. A valid value is a MIME type (see Section 8.10). The attribute is optional. There is no default value (recording format is MS-specific).

dest: specifies the location of the resulting recording. The MS MAY stream the recorded data to the location as recording occurs. A valid value is a URI (see Section 8.9). The attribute is optional. If not specified, the MS MUST provide a recording location URI; this recording is available until the connection or conference associated with the dialog is destroyed.

vadinitial: Control whether voice activity detection can be used to initiate the recording operation. A valid value is a boolean (see Section 8.1). A value of true indicates recording may be initiated using voice activity detection. A value of false indicates that recording must not be initiated using voice activity detection. The attribute is optional. The default value is true.

vadfinal: Control whether voice activity detection can be used to terminate the recording operation. A valid value is a boolean (see Section 8.1). A value of true indicates recording may be terminated using voice activity detection. A value of false indicates that recording must not be terminated using voice activity detection. The attribute is optional. The default value is true.

dtmfterm: Indicates whether the recording operation is terminated by DTMF input. A valid value is a boolean (see Section 8.1). A value of true indicates that recording is terminated by DTMF input. A value of false indicates that recording is not terminated by DTMF input. The attribute is optional. The default value is true.

maxtime: indicates The maximum duration of the recording. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 15s.

beep: indicates whether a 'beep' should be played immediately prior to initiation of the recording operation. A valid value is a boolean (see Section 8.1). The attribute is optional. The default value is false.

finalsilence: indicates the interval of silence that indicates end of speech. This parameter is ignored if vadfinal (vadfinal) has the value false. A valid value is a Time Designation (see Section 8.7). The attribute is optional. The default value is 5s.

It is an error if a dest attribute is specified and the MS does not understand the URI protocol, cannot locate the URI, or cannot send recorded data successfully to the location.

The <record> element has no child elements.

Recording has the following execution model upon initialization:

1. If an error occurs during execution, then recording terminates and the error is reported in the status attribute (see Section 5.5) of the <dialogexit> event.

2. If a beep attribute with the value of true is specified, then a beep tone is played.

3. A timer with the duration of the value of the timeout attribute is activated. If the timer expires before the recording operation is completed, then recording execution terminates and

the dialogexit result contains a <recordinfo> (see Section 5.5.3) with the termmode attribute set to noinput.

4. Initiation of the recording operation depends on the value of the vadinitial attribute. If vadinitial has the value false, then the recording operation is initiated immediately. Otherwise, the recording operations is initiated when voice activity is detected.

5. When the recording operation is initiated, a timer is started for the value of the maxtime attribute. If the timer expires before the recording operation is complete, then recording execution terminates and the dialogexit result contains a <recordinfo> (see Section 5.5.3) with the termmode attribute set to maxtime.

6. During the record operation user media input is recording in the format specified by the value of the type attribute. If the dest attribute is specified, then recorded input is sent to that location. Otherwise, MS uses an internal location.

7. If the dtmfterm attribute has the value true and DTMF input is detected during the record operation, then the recording terminates and and the dialogexit result contains a <recordinfo> (see Section 5.5.3) with the termmode attribute set to dtmf.

8. If vadfinal attribute has the value true, then the recording operation is terminated when a period of silence, with the duration specified by the value of the finalsilence attribute, is detected. The dialogexit result contains a <recordinfo> (see Section 5.5.3) with the termmode attribute set to finalsilence.

5.5. Exit Information

When a basic ivr dialog exits, information is reported in a <dialogexit> notification event.

The content model of the <dialogexit> element defined in Section 4.3.2 is extended with the following child elements:

<promptinfo>: reports information about the prompt execution (see Section 5.5.1). The element is optional.

<collectinfo>: reports information about the collect execution (see Section 5.5.2). The element is optional.

<recordinfo>: reports information about the record execution (see Section 5.5.3). The element is optional.

5.5.1. <promptinfo>

The <promptinfo> element reports the information about prompt execution. It has the following attributes:

duration: indicates the cumulative duration of prompt playing in milliseconds. A valid value is a non-negative integer (see Section 8.4). The attribute is optional. There is no default value.

iterations: indicates the number of iterations played completely. A valid value is a positive integer (see Section 8.5). The attribute is optional. There is no default value.

termmode: indicates how playback was terminated. Valid values are: 'completed', 'maxduration' or 'bargein'. The attribute is mandatory.

The <promptinfo> element has no child elements.

5.5.2. <collectinfo>

The <collectinfo> element reports the information about collect execution. It has the following attributes:

dtmf: DTMF input collected from the user. A valid value is a DTMF string (see Section 8.3 with no space between characters. The attribute is optional. There is no default value.

termmode: indicates how collection was terminated. Valid values are: 'match', 'noinput' or 'nomatch'. The attribute is mandatory.

The <collectinfo> element has no child elements.

5.5.3. <recordinfo>

The <recordinfo> element reports the information about record execution. It has the following attributes:

recording: references the location to which media is recorded. A valid value is a URI (see Section 8.9). The attribute is optional. There is no default value.

type: indicates the format of the recording. A valid value is a MIME type (see Section 8.10). The attribute is optional. There is no default value.

duration: indicates the duration of the recording in milliseconds. A valid value is a non-negative integer (see Section 8.4). The attribute is optional. There is no default value.

size: indicates the size of the recording in bytes. A valid value is a non-negative integer (see Section 8.4). The attribute is optional. There is no default value.

termmode: indicates how recording was terminated. Valid values are: 'noinput', 'dtmf', 'maxtime', and 'finalsilence'. The attribute is mandatory.

The <recordinfo> element has no child elements.

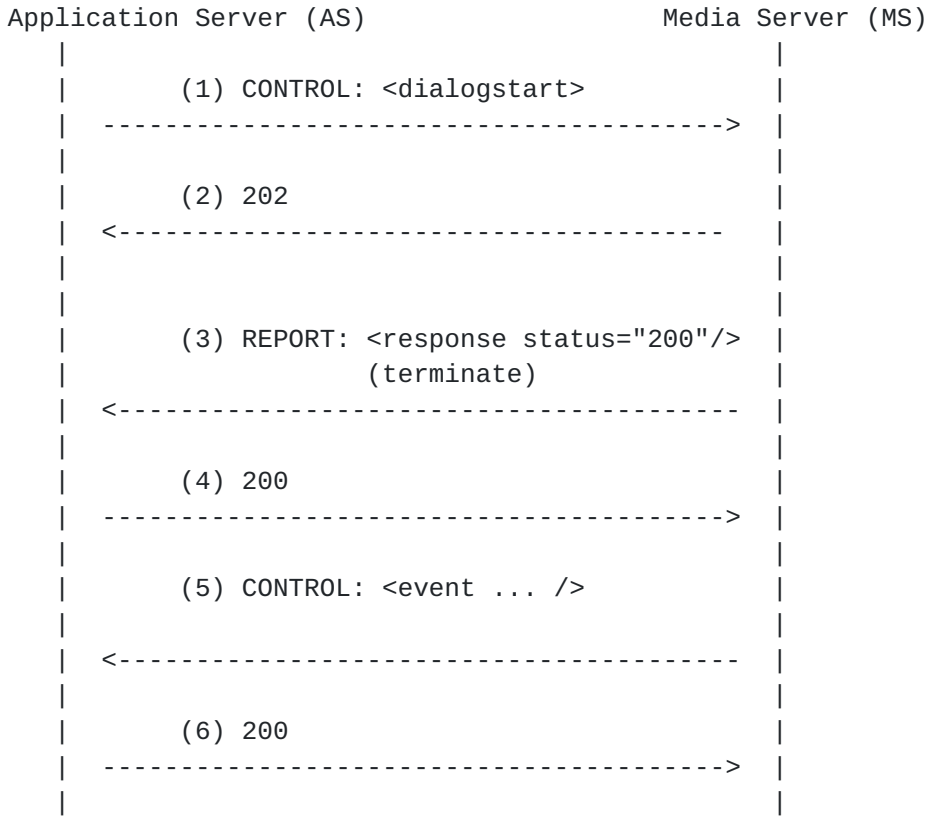
6. AS-MS Dialog Interaction Examples

The following example assume a control channel has been established as described in the Media Control Channel Framework ([MCCF]).

The XML messages are in angled brackets; the REPORT status is in round brackets. Other aspects of the protocol are omitted for readability.

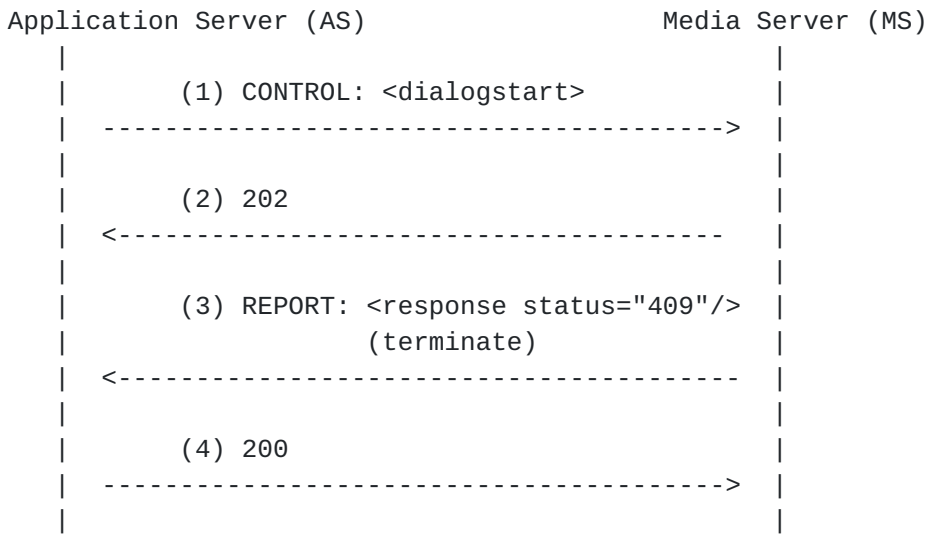
6.1. Starting an IVR dialog

An IVR dialog is started successfully, and dialogexit notification <event> is sent from the MS to the AS when the dialog exits normally.



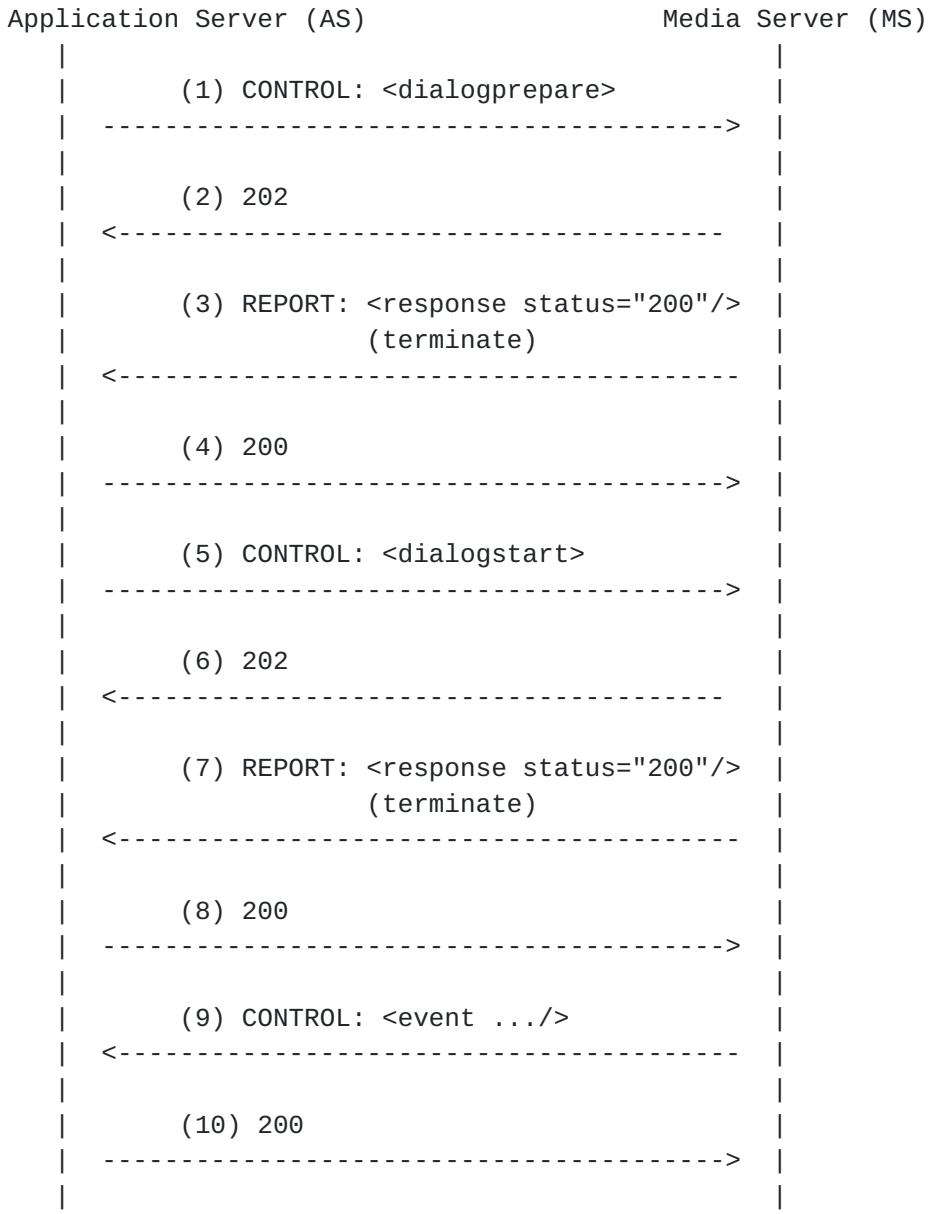
6.2. IVR dialog fails to start

An IVR dialog fails to start due to an unknown dialog type.



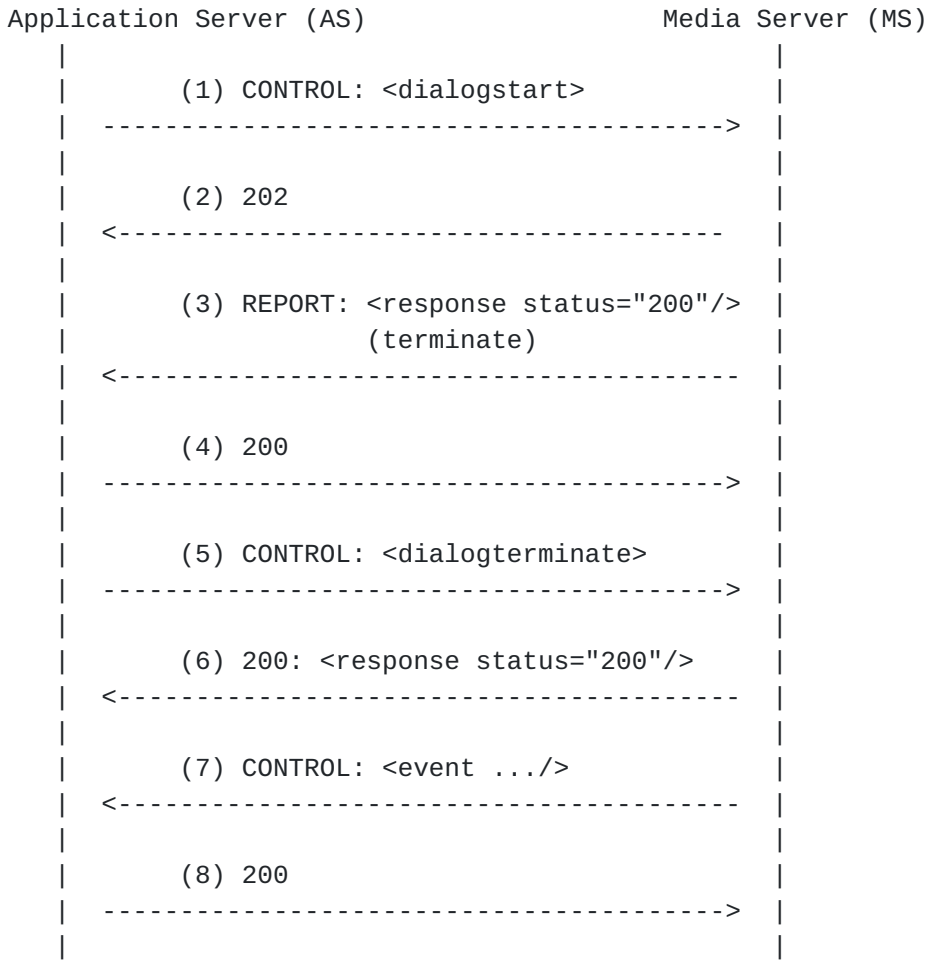
6.3. Preparing and starting an IVR dialog

An IVR dialog is prepared and started successfully, and then the dialog exits normally.



6.4. Terminating a dialog

An IVR dialog is started successfully, and then terminated by the AS. The dialogexit event is sent by to the AS when the dialog exits.



Note that in (6) the <response> payload to the <dialogterminate/> request is carried on a framework 200 response since it could complete the requested operation before the transaction timeout.

7. Basic IVR Dialog Examples

The following examples show how <basicivr> is used with <dialogprepare>, <dialogstart> and <event> elements to play prompts, collect DTMF input and record user input.

The examples do not specify all messages between the AS and MS.

7.1. Playing announcements

This example prepares an announcement composed of two prompts.

```
<dialogprepare>
   <basicivr>
    <prompt>
      <media src="http://www.example.com/media/Number_09.wav"/>
      <media src="http://www.example.com/media/Number_11.wav"/>
    </prompt>
   </basicivr>
</dialogprepare>
```

If the dialog is prepared successfully, a <response> with status 200 is returned:

```
<response status="200" dialogid="vxi78"/>
```

The prepared dialog is then started on a conference playing the prompts twice:

```
<dialogstart prepareddialogid="vxi78" conferenceid="conference11"/>
```

In the case of a successful dialog, the output is provided in <event>; for example

```
<event dialogid="vxi78">
    <dialogexit status="1">
       <promptinfo termmode="completed"/>
    </dialogexit>
</event>
```

7.2. Prompt and collect

This example plays no prompts and just waits for DTMF input from the user:

```
<dialogstart connectionid="7HDY839~HJKSkyHS~HUwkuh7ns">
 <basicivr>
  <collect/>
 </basicivr>
</dialogstart>
```

If the dialog is successful, then "dialogexit" <event> contains the dtmf collected in its result parameter:

```
<event dialogid="vxi80">
    <dialogexit status="1">
       <collectinfo dtmf="12345" termmode="match"/>
    </dialogexit>
</event>
```

In this example, a prompt is played and then we wait for 3 hours for a two digit sequence:

```
<dialogstart connectionid="7HDY839~HJKSkyHS~HUwkuh7ns">
 <basicivr>
  <prompt>
   <media src="http://www.example.com/prompt1.wav"/>
  </prompt>
  <collect timeout="1080s" maxdigits="2"/>
 </basicivr>
</dialogstart>
```

If no user input is collected within 3 hours, then following would be returned:

```
<event dialogid="vxi81">
    <dialogexit status="1" >
       <collectinfo termmode="noinput"/>
    </dialogexit>
</event>
```

And finally in this example, one of the input parameters is invalid:

```
<dialogstart connectionid="7HDY839~HJKSkyHS~HUwkuh7ns">
 <basicivr>
   <prompt iterations="two">
     <media src="http://www.example.com/prompt1.wav"/>
   </prompt>
   <collect cleardigitbuffer="true" bargein="true"
   timeout="4s" interdigittimeout="2s"
   termtimeout="0s maxdigits="2"/>
 </basicivr>
</dialogstart>
```

The error is reported in a the response:

```
<response status="411" dialogid="vxi82"
  reason="iterations value invalid: two"/>
```

7.3. Prompt and record

In this example, the user is prompted, then their input is recorded for a maximum of 30 seconds.

```
<dialogstart connectionid="7HDY839~HJKSkyHS~HUwkuh7ns">
    <basicivr>
      <prompt>
       <media src="http://www.example.com/media/sayname.wav"/>
      </prompt>
      <record dtmfterm="false" maxtime="30s" beep="true"/>
    </basicivr>
</dialogstart>
```

If successful, the following is returned in a dialogexit <event>:

```
<event dialogid="vxi83">
 <dialogexit status="1">
  <recordinfo recording="http://www.example.com/recording1.wav"
              termmode="dtmf"/>
 </dialogexit>
</event>
```

8. Type Definitions

This section defines types referenced in attribute definitions.

8.1. Boolean

The value space of boolean is the set {true, false}.

8.2. DTMFChar

A DTMF character. The value space is the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, #, *, A, B, C, D}.

8.3. DTMFString

A String composed of one or more DTMFChars.

8.4. Non-Negative Integer

The value space of non-negative integer is the infinite set {0,1,2,...}.

8.5. Positive Integer

The value space of positive integer is the infinite set {1,2,...}.

8.6. String

A string in the character encoding associated with the XML element.

8.7. Time Designation

A time designation consists of a non-negative real number followed by a time unit identifier.

The time unit identifiers are: "ms" (milliseconds) and "s" (seconds).

Examples include: "3s", "850ms", "0.7s", ".5s" and "+1.5s".

8.8. Percentage

A percentage consists of a Positive Integer followed by "%".

Examples include: "100%", "500%" and "10%".

8.9. URI

Uniform Resource Indicator as defined in [RFC3986].

8.10. mimetype

A string formated as a IANA mimetype.

8.11. Language Identifier

A language identifier labels information content as being of a particular human language variant. Following the XML specification for language identification [XML], a legal language identifier is identified by a RFC3066 ([RFC3066]) code where the language code is required and a country code or other subtag identifier is optional.

9. Formal Syntax

This package defines two XML schema:

1. msc-ivr-common.xsd: defines the common datatypes, attributes and dialog management elements. It does not use a namespace.

2. msc-ivr-basic.xsd: defines the basic ivr elements and uses schema redefinition to extend dialog management elements. All elements in this schema, including the imported dialog management elements, are in the urn:ietf:params:xml:ns:msc-ivr-basic namespace.

The schema are dependent upon the schema (framework.xsd) defined in Section 16.1 of the Control Framework [MCCF]

Both schema are fully extensible: attributes and elements from other namespaces are permitted, and extensions can be specified using the <redefine> mechanism.

The schema of extension control packages MUST imported one of these schema and specify how they extend the elements. Extensions to the <basicivr> element use msc-ivr-basic.xsd. Extensions to the dialog management elements use msc-ivr-common.xsd.

[Editors Note:IVR14 A later version of this document may provide a RelaxNG schema which provides better support for co-occurrence constraints than XML schema. Does anyone want/need such a schema?]

9.1. msc-ivr-common.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified"
 xmlns:fw="urn:ietf:params:xml:ns:control:framework-attributes"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
  <xsd:documentation>
   IETF MediaCtrl IVR Common 1.0 (20080225)

   This is the common core schema of the IVR packages defined
   as part of the Basic IVR Control Package. It specifies common
   datatypes, attributes and dialog control element definitions.

  </xsd:documentation>
 </xsd:annotation>
 <!--
  ##################################################
```

```
   SCHEMA IMPORTS

   ##################################################
  -->
  <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="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

     See http://www.w3.org/2001/xml.xsd for latest version
    </xsd:documentation>
   </xsd:annotation>
  </xsd:import>
  <!--
   ######################################################

   COMMON ELEMENTS

   ######################################################
  -->
  <!--  dialogprepare -->
  <xsd:attributeGroup name="msc.ivr.common.dialogprepare.attlist">
   <xsd:attribute name="src" type="xsd:anyURI" />
   <xsd:attribute name="type" type="mime.datatype" />
   <xsd:attribute name="dialogid" type="dialogid.datatype" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.dialogprepare.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.dialogprepare.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.common.dialogprepare.mix" minOccurs="0"
```

```
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.dialogprepare.type">
   <xsd:group ref="msc.ivr.common.dialogprepare.content" />
   <xsd:attributeGroup ref="msc.ivr.common.dialogprepare.attlist" />
  </xsd:complexType>
  <xsd:element name="dialogprepare"
   type="msc.ivr.common.dialogprepare.type" />
  <!--  dialogstart -->
  <xsd:attributeGroup name="msc.ivr.common.dialogstart.attlist">
   <xsd:attribute name="src" type="xsd:anyURI" />
   <xsd:attribute name="type" type="mime.datatype" />
   <xsd:attribute name="dialogid" type="dialogid.datatype" />
   <xsd:attribute name="prepareddialogid" type="dialogid.datatype" />
   <xsd:attributeGroup ref="fw:framework-attributes" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.dialogstart.mix">
   <xsd:choice>
    <xsd:element ref="stream" minOccurs="0" maxOccurs="unbounded" />
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.dialogstart.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.common.dialogstart.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.dialogstart.type">
   <xsd:group ref="msc.ivr.common.dialogstart.content" />
   <xsd:attributeGroup ref="msc.ivr.common.dialogstart.attlist" />
  </xsd:complexType>
  <xsd:element name="dialogstart"
   type="msc.ivr.common.dialogstart.type" />
  <!--  dialogterminate -->
  <xsd:attributeGroup name="msc.ivr.common.dialogterminate.attlist">
   <xsd:attribute name="dialogid" type="dialogid.datatype"
    use="required" />
   <xsd:attribute name="immediate" type="boolean.datatype"
    default="false" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.dialogterminate.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
```

```
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.dialogterminate.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.common.dialogterminate.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.dialogterminate.type">
   <xsd:group ref="msc.ivr.common.dialogterminate.content" />
   <xsd:attributeGroup ref="msc.ivr.common.dialogterminate.attlist" />
  </xsd:complexType>
  <xsd:element name="dialogterminate"
   type="msc.ivr.common.dialogterminate.type" />
  <!--  response -->
  <xsd:attributeGroup name="msc.ivr.common.response.attlist">
   <xsd:attribute name="status" type="status.datatype" use="required" />
   <xsd:attribute name="reason" type="xsd:string" />
   <xsd:attribute name="dialogid" type="dialogid.datatype" />
   <xsd:attributeGroup ref="fw:framework-attributes" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.response.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.response.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.common.response.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.response.type">
   <xsd:group ref="msc.ivr.common.response.content" />
   <xsd:attributeGroup ref="msc.ivr.common.response.attlist" />
  </xsd:complexType>
  <xsd:element name="response" type="msc.ivr.common.response.type" />
  <!--  event -->
  <xsd:attributeGroup name="msc.ivr.common.event.attlist">
   <xsd:attribute name="dialogid" type="dialogid.datatype"
    use="required" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.event.mix">
   <xsd:choice>
```

```
    <xsd:element ref="dialogexit" />
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.event.content">
   <xsd:choice>
    <xsd:group ref="msc.ivr.common.event.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.event.type">
   <xsd:group ref="msc.ivr.common.event.content" />
   <xsd:attributeGroup ref="msc.ivr.common.event.attlist" />
  </xsd:complexType>
  <xsd:element name="event" type="msc.ivr.common.event.type" />
  <!--  dialogexit-->
  <xsd:attributeGroup name="msc.ivr.common.dialogexit.attlist">
   <xsd:attribute name="status" type="xsd:positiveInteger"
    use="required" />
   <xsd:attribute name="reason" type="xsd:string" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.dialogexit.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.dialogexit.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.common.dialogexit.mix"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.dialogexit.type">
   <xsd:group ref="msc.ivr.common.dialogexit.content" />
   <xsd:attributeGroup ref="msc.ivr.common.dialogexit.attlist" />
  </xsd:complexType>
  <xsd:element name="dialogexit"
   type="msc.ivr.common.dialogexit.type" />
  <!--  stream -->
  <xsd:attributeGroup name="msc.ivr.common.stream.attlist">
   <xsd:attribute name="media" type="media.datatype" use="required" />
   <xsd:attribute name="label" type="label.datatype" />
   <xsd:attribute name="direction" type="direction.datatype"
    default="sendrecv" />
   <xsd:attributeGroup ref="msc.common.attribs" />
```

```
  </xsd:attributeGroup>
  <xsd:group name="msc.ivr.common.stream.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>
  <xsd:group name="msc.ivr.common.stream.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.common.stream.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="msc.ivr.common.stream.type">
   <xsd:group ref="msc.ivr.common.stream.content" />
   <xsd:attributeGroup ref="msc.ivr.common.stream.attlist" />
  </xsd:complexType>
  <xsd:element name="stream" type="msc.ivr.common.stream.type" />
  <!--
   ######################################################

   COMMON ATTRIBUTES AND ELEMENTS

   ######################################################
  -->
  <xsd:attributeGroup name="msc.common.attribs">
   <xsd:annotation>
    <xsd:documentation>
     group allowing attributes from other namespaces
    </xsd:documentation>
   </xsd:annotation>
   <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:attributeGroup>
  <xsd:group name="msc.common.content">
   <xsd:annotation>
    <xsd:documentation>
     group allowing elements from other namespaces
    </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>
  <!--
   ######################################################

   COMMON DATATYPES
```

```
   ##################################################
  -->
  <xsd:simpleType name="mime.datatype">
   <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
  <xsd:simpleType name="dialogid.datatype">
   <xsd:restriction base="xsd:string" />
  </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="status.datatype">
   <xsd:restriction base="xsd:NMTOKEN">
    <xsd:pattern value="[0-9][0-9][0-9]" />
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="media.datatype">
   <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
  <xsd:simpleType name="label.datatype">
   <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
  <xsd:simpleType name="direction.datatype">
   <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="sendrecv" />
    <xsd:enumeration value="sendonly" />
    <xsd:enumeration value="recvonly" />
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="timedesignation.datatype">
   <xsd:annotation>
    <xsd:documentation>
     Time designation following Time in CSS2
    </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
    <xsd:pattern value="(\+)?([0-9]*\.)?[0-9]+(ms|s)" />
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="dtmfchar.datatype">
   <xsd:annotation>
    <xsd:documentation>DTMF character [0-9#*A-D]</xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9#*A-D]" />
```

```
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="dtmfstring.datatype">
   <xsd:annotation>
    <xsd:documentation>DTMF sequence [0-9#*A-D]</xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
    <xsd:pattern value="([0-9#*A-D])+" />
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="percentage.datatype">
   <xsd:annotation>
    <xsd:documentation>
     whole integer followed by '%'
    </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
    <xsd:pattern value="([0-9])+%" />
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:schema>
```

9.2. msc-ivr-basic.xsd

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema targetNamespace="urn:ietf:params:xml:ns:msc-ivr-basic"
  elementFormDefault="qualified"
  xmlns="urn:ietf:params:xml:ns:msc-ivr-basic"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
   <xsd:documentation>
    IETF MediaCtrl IVR Basic 1.0 (20080225)

    This is the schema of the Basic IVR control package. It imports
    the msc-ivr-common schema (dialogprepare, etc) and specifies
    basicivr, prompt, collect and record elements.

    The schema namespace is urn:ietf:params:xml:ns:msc-ivr-basic

   </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

     See http://www.w3.org/2001/xml.xsd for latest version
    </xsd:documentation>
   </xsd:annotation>
  </xsd:import>



  <xsd:redefine schemaLocation="msc-ivr-common.xsd">
   <xsd:annotation>
    <xsd:documentation>
     This import brings in the IVR common package and redefines some
     elements as follows: [1] adds basicivr element to dialogprepare
     [2] adds basicivr element to dialogstart [3] adds promptinfo,
     collectinfo and recordinfo as children of dialogexit element
    </xsd:documentation>
   </xsd:annotation>


   <xsd:group name="msc.ivr.common.dialogprepare.mix">
    <xsd:choice>
     <xsd:group ref="msc.ivr.common.dialogprepare.mix" />
     <xsd:element ref="basicivr" minOccurs="0" maxOccurs="1" />
    </xsd:choice>
   </xsd:group>

   <xsd:group name="msc.ivr.common.dialogstart.mix">
    <xsd:choice>
     <xsd:group ref="msc.ivr.common.dialogstart.mix" />
     <xsd:element ref="basicivr" minOccurs="0" maxOccurs="1" />
    </xsd:choice>
   </xsd:group>

   <xsd:group name="msc.ivr.common.dialogexit.mix">
    <xsd:choice>
     <xsd:group ref="msc.ivr.common.dialogexit.mix" />
     <xsd:element ref="promptinfo" minOccurs="0" maxOccurs="1" />
     <xsd:element ref="collectinfo" minOccurs="0" maxOccurs="1" />
```

```
     <xsd:element ref="recordinfo" minOccurs="0" maxOccurs="1" />
    </xsd:choice>
   </xsd:group>

  </xsd:redefine>



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

   BASIC IVR ELEMENTS

   ################################################################
  -->


  <!-- basicivr -->

  <xsd:attributeGroup name="msc.ivr.basicivr.basicivr.attlist">
   <xsd:attribute name="version" type="xsd:string" default="1.0" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.basicivr.mix">
   <xsd:choice>
    <xsd:element ref="prompt" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="collect" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="record" minOccurs="0" maxOccurs="1" />
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.basicivr.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.basicivr.mix" minOccurs="1"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>


  <xsd:complexType name="msc.ivr.basicivr.basicivr.type">
   <xsd:group ref="msc.ivr.basicivr.basicivr.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.basicivr.attlist" />
  </xsd:complexType>

  <xsd:element name="basicivr" type="msc.ivr.basicivr.basicivr.type" />
```

```
  <!--  prompt  -->

  <xsd:attributeGroup name="msc.ivr.basicivr.prompt.attlist">
   <xsd:attribute ref="xml:base" />
   <xsd:attribute name="bargein" type="boolean.datatype"
    default="true" />
   <xsd:attribute name="iterations" type="xsd:nonNegativeInteger"
    default="1" />
   <xsd:attribute name="duration" type="timedesignation.datatype" />
   <xsd:attribute name="volume" type="percentage.datatype"
    default="100%" />
   <xsd:attribute name="offset" type="timedesignation.datatype"
    default="0s" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.prompt.mix">
   <xsd:choice>
    <xsd:element ref="media" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="variable" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="dtmf" minOccurs="0" maxOccurs="unbounded" />
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.prompt.content">
   <xsd:choice>
    <xsd:group ref="msc.ivr.basicivr.prompt.mix"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.prompt.type">
   <xsd:group ref="msc.ivr.basicivr.prompt.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.prompt.attlist" />
  </xsd:complexType>

  <xsd:element name="prompt" type="msc.ivr.basicivr.prompt.type" />

  <!--  media -->

  <xsd:attributeGroup name="msc.ivr.basicivr.media.attlist">
   <xsd:attribute name="src" type="xsd:anyURI" use="required" />
   <xsd:attribute name="type" type="mime.datatype" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
```

```
  <xsd:group name="msc.ivr.basicivr.media.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.media.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.media.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.media.type">
   <xsd:group ref="msc.ivr.basicivr.media.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.media.attlist" />
  </xsd:complexType>

  <xsd:element name="media" type="msc.ivr.basicivr.media.type" />


  <!--  variable -->

  <xsd:attributeGroup name="msc.ivr.basicivr.variable.attlist">
   <xsd:attribute name="value" type="xsd:string" use="required" />
   <xsd:attribute name="type" type="xsd:string" use="required" />
   <xsd:attribute name="format" type="xsd:string" />
   <xsd:attribute ref="xml:lang" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.variable.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.variable.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.variable.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.variable.type">
   <xsd:group ref="msc.ivr.basicivr.variable.content" />
```

```
   <xsd:attributeGroup ref="msc.ivr.basicivr.variable.attlist" />
  </xsd:complexType>

  <xsd:element name="variable" type="msc.ivr.basicivr.variable.type" />

  <!-- dtmf -->

  <xsd:attributeGroup name="msc.ivr.basicivr.dtmf.attlist">
   <xsd:attribute name="digits" type="dtmfstring.datatype"
    use="required" />
   <xsd:attribute name="level" type="xsd:integer" default="-6" />
   <xsd:attribute name="duration" type="timedesignation.datatype"
    default="100ms" />
   <xsd:attribute name="interval" type="timedesignation.datatype"
    default="100ms" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.dtmf.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.dtmf.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.dtmf.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.dtmf.type">
   <xsd:group ref="msc.ivr.basicivr.dtmf.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.dtmf.attlist" />
  </xsd:complexType>

  <xsd:element name="dtmf" type="msc.ivr.basicivr.dtmf.type" />



  <!-- collect -->

  <xsd:attributeGroup name="msc.ivr.basicivr.collect.attlist">
   <xsd:attribute name="cleardigitbuffer" type="boolean.datatype"
    default="true" />
   <xsd:attribute name="timeout" type="timedesignation.datatype"
    default="5s" />
```

```
   <xsd:attribute name="interdigittimeout"
    type="timedesignation.datatype" default="2s" />
   <xsd:attribute name="termtimeout" type="timedesignation.datatype"
    default="0s" />
   <xsd:attribute name="escapekey" type="dtmfchar.datatype" />
   <xsd:attribute name="termchar" type="dtmfchar.datatype"
    default="#" />
   <xsd:attribute name="maxdigits" type="xsd:positiveInteger"
    default="5" />
   <xsd:attribute name="skipinterval" type="timedesignation.datatype"
    default="6s" />
   <xsd:attribute name="ffkey" type="dtmfchar.datatype" />
   <xsd:attribute name="rwkey" type="dtmfchar.datatype" />
   <xsd:attribute name="pauseinterval" type="timedesignation.datatype"
    default="10s" />
   <xsd:attribute name="pausekey" type="dtmfchar.datatype" />
   <xsd:attribute name="resumekey" type="dtmfchar.datatype" />
   <xsd:attribute name="volumeinterval" type="percentage.datatype"
    default="10%" />
   <xsd:attribute name="volupkey" type="dtmfchar.datatype" />
   <xsd:attribute name="voldnkey" type="dtmfchar.datatype" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.collect.mix">
   <xsd:choice>
    <xsd:element ref="grammar" minOccurs="0" maxOccurs="1" />
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.collect.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.collect.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.collect.type">
   <xsd:group ref="msc.ivr.basicivr.collect.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.collect.attlist" />
  </xsd:complexType>

  <xsd:element name="collect" type="msc.ivr.basicivr.collect.type" />


  <!--  grammar -->
```

```
  <xsd:attributeGroup name="msc.ivr.basicivr.grammar.attlist">
   <xsd:attribute name="src" type="xsd:anyURI" />
   <xsd:attribute name="type" type="mime.datatype" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.grammar.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.grammar.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.grammar.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.grammar.type"
   mixed="true">
   <xsd:group ref="msc.ivr.basicivr.grammar.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.grammar.attlist" />
  </xsd:complexType>

  <xsd:element name="grammar" type="msc.ivr.basicivr.grammar.type" />

  <!-- record -->

  <xsd:attributeGroup name="msc.ivr.basicivr.record.attlist">
   <xsd:attribute name="timeout" type="timedesignation.datatype"
    default="5s" />
   <xsd:attribute name="type" type="mime.datatype" />
   <xsd:attribute name="dest" type="xsd:anyURI" />
   <xsd:attribute name="beep" type="boolean.datatype" default="false" />
   <xsd:attribute name="vadinitial" type="boolean.datatype"
    default="true" />
   <xsd:attribute name="vadfinal" type="boolean.datatype"
    default="true" />
   <xsd:attribute name="dtmfterm" type="boolean.datatype"
    default="true" />
   <xsd:attribute name="maxtime" type="timedesignation.datatype"
    default="15s" />
   <xsd:attribute name="finalsilence" type="timedesignation.datatype"
    default="5s" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>
```

```
  <xsd:group name="msc.ivr.basicivr.record.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.record.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.record.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.record.type">
   <xsd:group ref="msc.ivr.basicivr.record.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.record.attlist" />
  </xsd:complexType>

  <xsd:element name="record" type="msc.ivr.basicivr.record.type" />

  <!-- promptinfo -->

  <xsd:attributeGroup name="msc.ivr.basicivr.promptinfo.attlist">
   <xsd:attribute name="duration" type="xsd:nonNegativeInteger" />
   <xsd:attribute name="iterations" type="xsd:positiveInteger" />
   <xsd:attribute name="termmode" type="prompt_termmode.datatype"
    use="required" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.promptinfo.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.promptinfo.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.promptinfo.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>


  <xsd:complexType name="msc.ivr.basicivr.promptinfo.type">
   <xsd:group ref="msc.ivr.basicivr.promptinfo.content" />
```

```
   <xsd:attributeGroup ref="msc.ivr.basicivr.promptinfo.attlist" />
  </xsd:complexType>

  <xsd:element name="promptinfo"
   type="msc.ivr.basicivr.promptinfo.type" />


  <!--  collectinfo -->

  <xsd:attributeGroup name="msc.ivr.basicivr.collectinfo.attlist">
   <xsd:attribute name="dtmf" type="dtmfstring.datatype" />
   <xsd:attribute name="termmode" type="collect_termmode.datatype"
    use="required" />
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.collectinfo.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.collectinfo.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.collectinfo.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.collectinfo.type">
   <xsd:group ref="msc.ivr.basicivr.collectinfo.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.collectinfo.attlist" />
  </xsd:complexType>

  <xsd:element name="collectinfo"
   type="msc.ivr.basicivr.collectinfo.type" />


  <!-- recordinfo -->

  <xsd:attributeGroup name="msc.ivr.basicivr.recordinfo.attlist">
   <xsd:attribute name="recording" type="xsd:anyURI" />
   <xsd:attribute name="type" type="mime.datatype" />
   <xsd:attribute name="duration" type="xsd:nonNegativeInteger" />
   <xsd:attribute name="size" type="xsd:nonNegativeInteger" />
   <xsd:attribute name="termmode" type="record_termmode.datatype"
    use="required" />
```

```
   <xsd:attributeGroup ref="msc.common.attribs" />
  </xsd:attributeGroup>

  <xsd:group name="msc.ivr.basicivr.recordinfo.mix">
   <xsd:choice>
    <xsd:group ref="msc.common.content" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:choice>
  </xsd:group>

  <xsd:group name="msc.ivr.basicivr.recordinfo.content">
   <xsd:sequence>
    <xsd:group ref="msc.ivr.basicivr.recordinfo.mix" minOccurs="0"
     maxOccurs="unbounded" />
   </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="msc.ivr.basicivr.recordinfo.type">
   <xsd:group ref="msc.ivr.basicivr.recordinfo.content" />
   <xsd:attributeGroup ref="msc.ivr.basicivr.recordinfo.attlist" />
  </xsd:complexType>

  <xsd:element name="recordinfo"
   type="msc.ivr.basicivr.recordinfo.type" />



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

   BASIC IVR DATATYPES

   ################################################################
  -->



  <xsd:simpleType name="prompt_termmode.datatype">
   <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="completed" />
    <xsd:enumeration value="maxduration" />
    <xsd:enumeration value="bargein" />
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="collect_termmode.datatype">
   <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="match" />
```

```
    <xsd:enumeration value="noinput" />
    <xsd:enumeration value="nomatch" />
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="record_termmode.datatype">
   <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="noinput" />
    <xsd:enumeration value="dtmf" />
    <xsd:enumeration value="maxtime" />
    <xsd:enumeration value="finalsilence" />
   </xsd:restriction>
  </xsd:simpleType>

 </xsd:schema>
```

10. Security Considerations

As this control package uses XML markup, implementation MUST address the security considerations of [RFC3023].

11. IANA Considerations

This specification instructs IANA to register a new Media Control Channel Framework Package, a new XML namespace and a new mime type.

11.1. Control Package Registration

Control Package name: msc-ivr-basic/1.0

11.2. URN Sub-Namespace Registration

XML namespace: urn:ietf:params:xml:ns:msc-ivr-basic

11.3. Mime Type Registration

MIME type: application/msc-ivr-basic+xml

12. Change Summary

Note to RFC Editor: Please remove this whole section.

The following are the major changes between the -06 of the draft and the -05 version.

- Event notifications are sent in CONTROL messages with the MS acting as Control Framework Client. Compared with the previous approach, this means that a <dialogstart> transaction is now complete when the MS sends a <response>. A new transaction is initiated by the MS each time the MS sends a notification <event> to the AS.

- Changed conf-id to conferenceid and connection-id to connectionid.

- Clarification of the state model for dialogs

- <dialogprepare>: modified definition of src attribute to allow reference to external dialog documents; added (MIME) type attribute; removed <data> child element.

- <dialogstart>: modified definition of src attribute to allow reference to external dialog documents; added (MIME) type attribute; removed <data> child element;

- <dialogterminate>: modified so that a dialogexit event is always sent for active dialogs (i.e. the dialogexit event is a terminating notification)

- <event> notification simplified and make more extensible. Manual notifications (via <subscribe> element) are removed from the basic package. A <dialogexit> event is defined as <event> child and it can be extended with additional child elements

- <data> element is removed.

- <subscribe> element removed.

- Replaced dialog templates with a general <basicivr> element. It has child elements for playing media resource (<prompt>), collecting DTMF (<collect>) and recording (<record>). The functionality is largely unchanged.

- <dialogprepare> and <dialogstart> are extended with <basicivr> child element.

- <event> is extended with a <dialogexit> element which contains status and reported information (replacement for output parameters in template dialogs)

- Prompts: now structured as a <prompt> element with <media>, <variable> and <dtmf> children. The <prompt> element has xml:base attribute, bargein, iterations, duration, volume and offset attributes. The speed attribute is removed. A <media> element has src and type attributes. The maxage and maxstale attributes are removed.

- DTMF input: parameters now specified as attributes of a <collect> element. Custom grammar specified with a <grammar> element as child of <collect> element. Added 'escapekey' to allow the dialog to be retried. Added 'pauseinterval', 'pausekey' and 'resumekey' to allow the prompts to paused/resumed. Added 'volumeinterval', 'volupkey' and voldnkey' to add prompt volume to be increased/ decreased. Moved 'bargein' attribute to prompt.

- Recording: parameters now specified as attributes of <record> element. Added 'dest' and 'beep' attributes.

The following are the major changes between the -05 of the draft and the -04 version.

- Mainly an alignment/evaluation exercise with requirements produced by MEDIACTRL IVR design team.

- playannouncement parameters from Table 7 of '04' version are now reflected in text - schema to be updated.

- Added VCR commands based on MSCML.

The following are the major changes between the -04 of the draft and the -03 version.

- None.

The following are the major changes between the -03 of the draft and the -02 version.

- added "basicivr:" protocol to template dialog types which must be supported as values of the "src" attribute in <dialogprepare> and <dialogstart>. Note alternative: "/basicivr/playannouncement" offered in [RFC4240].

- added "basicivr:" URI schema to IANA considerations

- Added mimetype, vadinitial and vadfinal parameters to 'promptandrecord' dialog type
- updated references

The following are the major changes between the -02 of the draft and the -01 version.

- added version 1.0 to package name
- separate section for element definitions
- dialogterminate treated as request rather than notification
- simplified responses: single element <response>
- removed response elements: <dialogprepared>, <dialogstarted>, <errordialognotprepared>, <errordialognotstarted>
- simplified event notifications to single <event> element carried in a REPORT
- <dialogexit> element replaced with <event name="dialogexit">
- removed <dialoguser> element
- added <stream> element as child of <dialogstart>
- removed 'type' attribute from <dialogprepare> and <dialogstart>
- added dialogid attribute to <dialogprepare> and <dialogstart>
- removed template "Sample Implementation" section
- renamed <namelist> to <data>
- re-organized so that template details after general package framework and element description.

The following are the primary changes between the -01 of the draft and the -00 version.

- Removed requirement for VoiceXML dialog support
- Added requirement for template dialog support

13. Contributors

Asher Shiratzky from Radvision provided valuable support and contributions to the early versions of this document.

The authors would like to thank the IVR design team consisting of Roni Even, Lorenzo Miniero, Adnan Saleem and Diego Besprosvan who provided valuable feedback, input and text to this document.

14. Acknowledgments

The authors would like to thank Adnan Saleem of Radisys, Gene Shtirmer of Intel, Dave Burke of Google and Dan York of Voxeo for useful reviews of this work.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

15.2. Informative References

[CCXML10] Auburn, R J., "Voice Browser Call Control: CCXML Version 1.0", W3C Working Draft (work in progress), January 2007.

[H.248.1] "Gateway control protocol: Version 3", ITU-T Recommendation H.248.1.

[H.248.9] "Gateway control protocol: Advanced media server packages", ITU-T Recommendation H.248.9.

[I-D.ietf-mediactrl-requirements] Dolly, M. and R. Even, "Media Server Control Protocol Requirements", draft-ietf-mediactrl-requirements-03 (work in progress), December 2007.

[MCCF] Boulton, C., Melanchuk, T., McGlashan, S., and A. Shiratzky, "Media Control Channel Framework", draft-ietf-mediactrl-sip-control-framework-01 (work in progress), February 2008.

[MSML] Saleem, A., Xin, Y., and G. Sharratt, "Media Session Markup Language (MSML)", draft-saleem-msml-06 (work in progress), February 2008.

[RFC2833] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[RFC2897] Cromwell, D., "Proposal for an MGCP Advanced Audio Package", RFC 2897, August 2000.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", RFC 3066, 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.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4240] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[RFC4281] Gellens, R., Singer, D., and P. Frojdh, "The Codecs Parameter for "Bucket" Media Types", RFC 4281, November 2005.

[RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.

[RFC4730] Burger, E. and M. Dolly, "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)", RFC 4730, November 2006.

[RFC5022] Van Dyke, J., Burger, E., and A. Spitzer, "Media Server Control Markup Language (MSCML) and Protocol", RFC 5022, September 2007.

[SRGS] Hunt, A. and S. McGlashan, "Speech Recognition Grammar Specification Version 1.0", W3C Recommendation, March 2004.

[VXML20] McGlashan, S., Burnett, D., Carter, J., Danielsen, P., Ferrans, J., Hunt, A., Lucas, B., Porter, B., Rehor, K., and S. Tryphonas, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Recommendation, March 2004.

[VXML21] Oshry, M., Auburn, RJ., Baggia, P., Bodell, M., Burke, D., Burnett, D., Candell, E., Carter, J., McGlashan, S., Lee, A., Porter, B., and K. Rehor, "Voice Extensible Markup Language (VoiceXML) Version 2.1", W3C Recommendation, June 2007.

[VXMLCP] Boulton, C., Melanchuk, T., and S. McGlashan, "A VoiceXML Interactive Voice Response (IVR) Control Package for the Media Control Channel Framework", draft-boulton-ivr-vxml-control-package-04 (work in progress), February 2008.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation, February 2004.

Authors' Addresses

Chris Boulton
Avaya
Building 3
Wern Fawr Lane
St Mellons
Cardiff, South Wales CF3 5EA

Email: cboulton@avaya.com

Tim Melanchuk
Rain Willow Communications

Email: tim.melanchuk@gmail.com

Scott McGlashan
Hewlett-Packard
Gustav III:s boulevard 36
SE-16985 Stockholm, Sweden

Email: scott.mcglashan@hp.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).