Media Control System 3.0

IP Control Protocol Specification

Table of Contents

Overview

This document is a reference for the Autonomic Media Control System (MCS) Ethernet and RS-232 control protocol. This control protocol is implemented in the following products:

1. Autonomic Mirage Media Servers
2. Autonomic Media Control System Software (MCS 3.0)
3. NuVo MPS4 Music Server (with optional firmware upgrade)
4. NuVo MPS4 Elite Music Server

This protocol provides two-way communications and control of multiple audio outputs using the AMP (Autonomic Media Playback) Engine

Commands are included for media transport control, media library browsing, and Windows Media Center shell navigation (MCS software only). Feedback is provided for browsing, currently playing media meta-data and album art, and navigation.

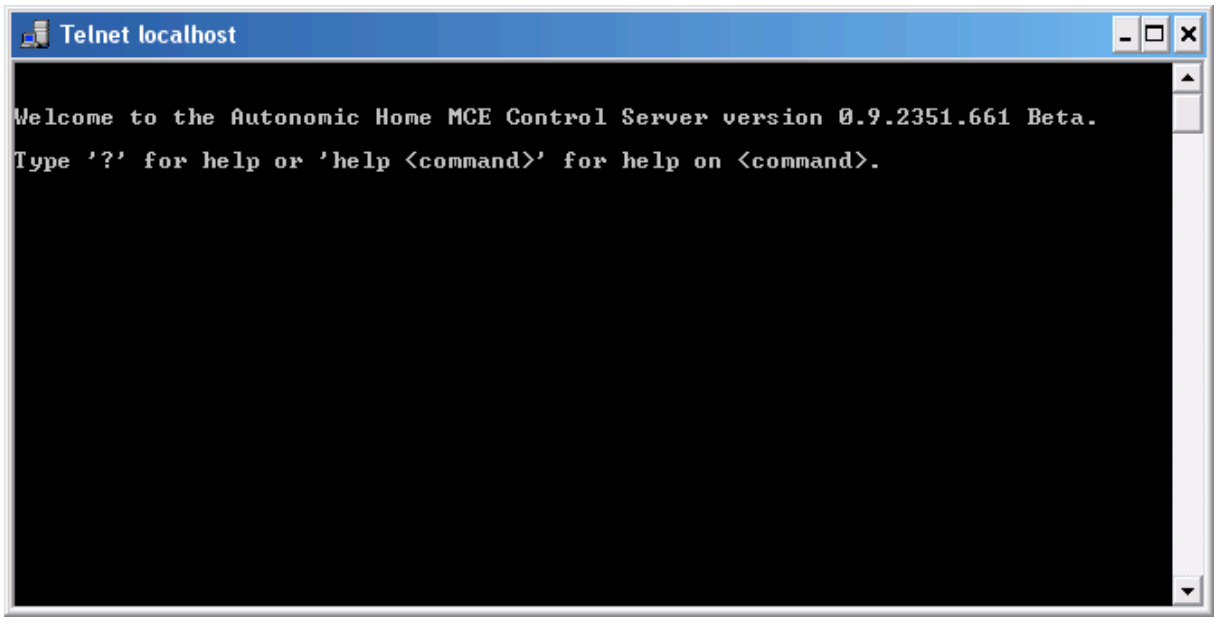
Telnet Access

For testing purposes, the MCS control socket can be reached via telnet on port 5004 of the server.

Local Connection

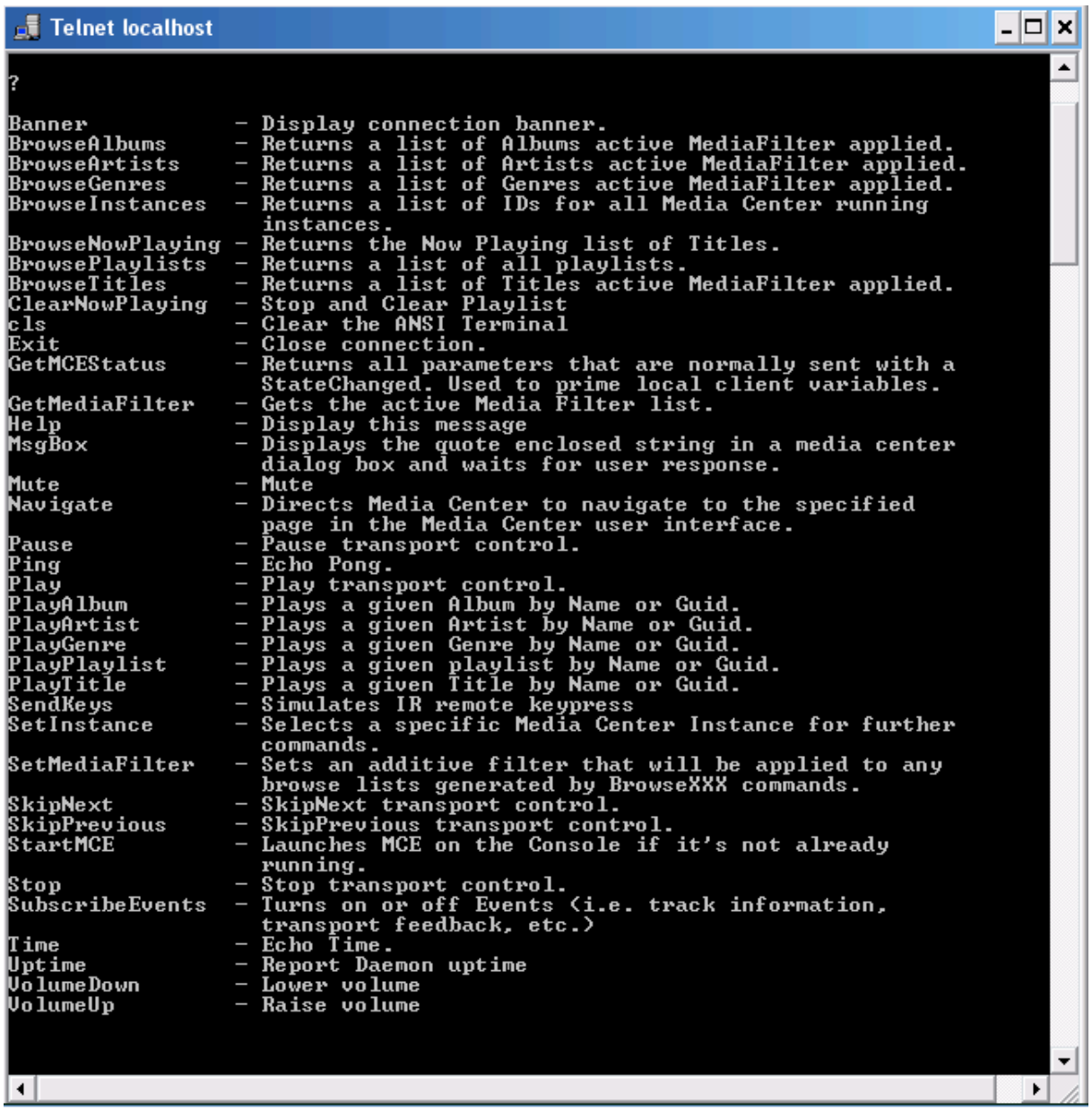
From the Start Menu, click Run, and type "telnet [*serverip]* 5004". If you have changed the server port number in the confutation utility (MCS software only), remember to substitute the number 5004 with the port you have selected.

If everything is working ok, you should see a window that looks something like this:



If you don't get a connection to the service, double check the Server Port setting in the configuration utility (MCS software), or check the server's IP configuration using the Mirage Media Server browser configuration pages.

Once you have a connection to the server, type "?" and hit enter. The Control Server will send a list of valid commands.



Try out a few commands, such as "BrowseAlbums". You should get a listing of Albums from your media library.

If you get an error message stating that the server is not licensed, use the configuration utility to register a valid license key, or obtain a trial license. (MCS Software)

If you are controlling Windows Media Center (MCS Software) and you get an error message stating that no instance is started, you can issue the command "**`StartMCS`**" to start the Media Center Interface. (You'll have to resize it in order to continue using the telnet window.)

MCS Control Specification

Protocol Conventions

Commands are case-insensitive. All commands and their responses are terminated with a carriage return / linefeed pair (CRLF)

Initiating control sessions

The TCP/IP client should initiate the control session with the Control Server by opening a socket to the server port specified in the configuration utility (default of 5004). When a connection has been established, the server will answer with the following string.

```
Welcome to the Autonomic Media Control Server version 3.0.XXXX.XXX Release.
Type '?' for help or 'help <command>' for help on <command>.
```

Starting the MCS Session

The client should take steps to insure that the MCS services are running on the host and get initial feedback.

This can be accomplished by issuing the command **`GetStatus.`** If the server is running and available, the server will respond with a series of **`ReportStatus`** responses:

```
ReportState Administrator@00-00-00-00-00-00 Volume=25
ReportState Administrator@00-00-00-00-00-00 SessionStart=FS_Home
ReportState Administrator@00-00-00-00-00-00 Running=True
```

In addition to the **`Running=True/False`** token, these **`ReportState`** responses may include media information, volume, and transport status. These can be used to initialize the client UI. **`ReportState`** and **`StateChanged`** tokens are fully documented later in this manual.

If using MCS software, it is possible that no instances are running. If there is no MCS session started (i.e. Running=False), the client can begin a session with the command StartMCS.

Command: **`StartMCS`**
Response: **`StartMCS OK`**

Selecting an MCS Instance

On hardware devices, such as the Mirage Media Server or NuVo MPS4 / MPS4 Elite, the multiple audio outputs are defined in the MCS protocol as **Instances**.

The client device is able to enumerate the available **Instances** using the **BrowseInstances** Command.

Example:

Command: **BrowseInstances.**

Response:
```
BeginInstances Total=2
  Media_Player_A
  Media_Player_B
  Media_Player_C
  Media_Player_D
EndInstances NoMore
```

In this example, there are four instances. The client may now select an instance to control, or present a selection list in its user interface to allow selection of control instance.

Command: **SetInstance "Media_Player_A"**

Response: **Instance=Media Player A**

To explicitly select the current instance running on the MCS host computer (MCS software only), use:

Command: **SetInstance ***

Response: **Instance=***

List Processing

Many commands result in the server returning a list of items. Sometimes, these lists can be very long, and will frequently exceed the number of items that can be displayed on a client device. The protocol includes methods for retrieving partial results, and randomly traversing sections of the list.

Lists can be retrieved in two different formats, standard ASCII space and CR/LF delimited lists, or XML based responses.

Standard lists are returned by default. To request XML based lists, issue this command once at the beginning of your session:

```
SetXMLMode Lists
```

Lists are requested by Browse[type] commands which are documented in the command reference later in this document.

Syntax: **Browse[*browsetype*] [*startposition*] [*requestcount*]**

Standard Response:

Header: **Begin[*listtype*] Total=[*count*]**

Items: **[itemtype] [field1] [field2]** …

…

…

Terminator: **End[*listype*] [More]|[NoMore]**

XML Response:

```
<[browsetype] total="[count]" start="[start]" more="true|false" art="true"
alpha="true" displayAs="Thumb" np="1">

   <[itemtype] guid="39654910-d033-4123-8de3-d7c878cae2e3" name="3 Doors
   Down" dna="name" button="1"/>

   …more item nodes…

</browsetype>
```

Where:

browsetype specifies the list being requested.

startposition specifies where the server should start returning list items. Can be an integer or a letter. (ie "C" would position you at the first item in the list that begins with C)

requestcount specified how many items should be returned.

listtype type of list being sent (i.e. **`BeginAlbums, BeginArtists`**, etc..)

count the total number of items in the list. This number may be larger than the number of items that will be returned in one response, as determined by the ***returncount*** parameter in the **`Browse`** command which initiated the list.

itemtype This is the type of list item being sent. (i.e. Album, Artist, Genre, etc.)

field1, field2 The content of these fields are dependant on the type of list. See the command reference for more information.

More|NoMore Indicates the availability of more list items beyond the requested section.

Additional XML Attributes.

art true or false and indicates that the item has album art

alpha true or false. Indicates that the list type can be browsed with an alpha index for the start parameter

dna display name attribute. This indicates which attribute should be used to display the item in a list on the user interface. If not supplied, use the "name" attribute.

name the items name. Use this attribute to display the item on the user interface, unless a dna attribute is supplied.

guid globally unique ID. This value is used to issue referencing commands, for example playing media.

other attributes additional attributes may be supplied depending on the list type.

Picklists

When the hierarchy of a given list is variable (i.e. RadioTime) then the results of a browse request may be a PickList.

```
SetRadioFilter Source="RadioTime"
RadioFilter Ok "RadioTime"

BrowseRadioStations 1 10
BeginPickList Total=8 Start=1 Alpha=0 Caption="RadioTime"
  PickListItem 52f878f1-b2db-1f8e-8de6-01556062a268 "Local Radio"
  PickListItem f69bef9e-b1a4-9bd7-0ab1-362382a9b73d "Music"
  PickListItem c53acb46-5995-d993-e195-4db6203c15a7 "Talk"
  PickListItem 17469649-8723-6fa5-aab1-072c4182a672 "Sports"
  PickListItem 109a2ba7-6927-9e2c-ad25-211225a77d62 "By Location"
  PickListItem c026a092-8fe0-7c7b-af50-2dff0a5afbb4 "By Language"
  PickListItem 34a91c9f-d6cc-eaa0-b295-777646affffa "Podcasts"
  PickListItem c244c390-098e-3361-d433-d2a4fee43fb8 "Settings"
EndPickList NoMore
RadioStations Ok
```

To select a PickListItem use the command **AckPickItem** as described below.

AckPickItem

Syntax: **AckPickItem guid**

Example:
```
AckPickItem 17469649-8723-6fa5-aab1-072c4182a672
BeginPickList Total=5 Start=1 Alpha=0 Caption="Sports"
      …
EndPickList NoMore
```

Use the AckPickItem command to select an item in a PickList

SetPickListCount

Syntax: **SetPickListCount numberOfItemsInAPickList**

Example: **SetPickListCount 10**

Since PickLists can be responded asynchronously as in the request for a context menu (see **AckButton**), the server needs to know the number of items the client expects to display in a list. Use the **SetPickListCount** command at the top of a session to initialize this value.

UI Events

There are times when the server needs to send a client a message and/or needs to collect information from the client. UI Events server this purpose.

UI Events use the same format as **StateChanged** messages (see **Feedback**) and take one of the following forms:

Navigate UI Event

Example: `StateChanged Main UI=<Navigate page ="NowPlaying" />`
Used to: Notify the client of a required page flip.

Valid values for **`page`** are:

NowPlaying	Client should flip to the now playing screen
RefreshList	Client should re-issue the last browse request as the data has changed.

StatusMessage

Example: `StateChanged Main UI=<StatusMessage message="Tuning to Dave Matthews Radio..." />`

Used to: Notify the client of a status message. The client should display this message in an unobtrusive way. This is NOT a pop-up message and should clear itself after a short period of time such as 5 seconds.

Clear

Example: `StateChanged Main UI=<Clear guid="15aa172f-c89d-4722-b970-d3c1f2565650" />`
Used to: Notify the client to clear pop-up messages.

MessageBox

Example: `StateChanged Main UI=`

```
<MessageBox guid="cb0520cb-ba54-4fec-a11d-1eee4a16a361" caption="Edit
Pandora station 'Avril Lavigne Radio'." message="What would you like to do
to this station?" timeout="30">
<Button text="Delete the station" action="AckButton cb0520cb-ba54-4fec-
a11d-1eee4a16a361 &quot;Delete the station&quot;" />
<Button text="Edit the station" action="AckButton cb0520cb-ba54-4fec-a11d-
1eee4a16a361 &quot;Edit the station&quot;" />
<Button text="Cancel" action="AckButton cb0520cb-ba54-4fec-a11d-
1eee4a16a361 &quot;Cancel&quot;" default="true" />
</MessageBox>
```

Used to: Notify client to pop up a message box.

This example has the following attributes:

caption	`Edit Pandora station 'Avril Lavigne Radio'.`	
message	`What would you like to do to this station?`	
timeout	30 (seconds)	At 30 seconds the client should press the default button.

This example has the following buttons:

`Delete the station`	Sends: `AckButton cb0520cb-ba54-4fec-a11d-1eee4a16a361 "Delete the station"` if pressed.	
`Edit the station`	Sends: `AckButton cb0520cb-ba54-4fec-a11d-1eee4a16a361 "Edit the station"` if pressed.	
`Cancel`	Sends: `AckButton cb0520cb-ba54-4fec-a11d-1eee4a16a361 "Cancel"` if pressed.	This is the default button

InputBox

Example: `StateChanged Main UI=<InputBox guid="15aa172f-c89d-4722-b970-d3c1f2565650" caption="Enter an artist or song" message="Type in the name of your favorite artist, song, or composer and Pandora will create a radio station featuring that music and more like it." timeout="120" value="" action="AckButton 15aa172f-c89d-4722-b970-d3c1f2565650 " />`

Used to: Notify client to pop up a input box.

This example has the following attributes:

caption	`Enter an artist or song`	
message	`Type in the name of your favorite artist, song, or composer and Pandora will create a radio station featuring that music and more like it.`	
timeout	120 (seconds)	At 30 seconds the client should press the default button.
action	Sends: `AckButton 15aa172f-c89d-4722-b970-d3c1f2565650` if pressed.	

Asynchronous processing

The MCS protocol is an asynchronous command protocol. This means that clients must be written so as to properly parse and process responses in any order that they may come in.

For example, if the command BrowseArtists is issued to the server, it will begin to send a list of artists. If an external process stops the media transport while this list is being processed, the server will send a StateChanged message in the middle of the list. This is necessary in order to insure that the client can issue responsive feedback to the user, which is vitally important to the control experience:

Example

```
Command:      SetMusicFilter Genre=Jazz
Response:     MusicFilter Genre=Jazz
Command:      BrowseArtists
Response:     BeginArtists Total=6
                Artist {19ef-4880-8064-a79e51ee270c} "Brian McKnight"
                Artist {c502-437a-81f6-3cddbed30059} "Frank Sinatra"
                Artist {6646-4ce2-b255-240c7b8f483a} "Tevin Campbell"
              StateChanged Player_A MediaControl=Stop
                Artist {9bf8-492b-b124-6879a65d414b} "Shakatak"
                Artist {39b7-47ae-bde5-9f9bd728237a} "James Ingram"
                Artist {bd96-4d32-8fbb-75a42d099370} "George Benson"
              EndArtists NoMore
```

The client must be written in a pre-emptive mode in order to properly receive and process messages from the server. Since each message is terminated with a CRLF pair, the client should continually fetch a string from the incoming buffer until a CRLF pair is encountered, search the string for tokens, process the message appropriately, issue appropriate user feedback, and then collect the next message from the buffer, and so on.

In this example, list items could be distinguished from the **`StateChanged`** message by the pair of spaces preceding the list item, and the unique token **`Artist`**.

Displaying Album Art

MCS enables client applications or Ethernet enabled touch panels to display album cover art for any media in the library using the built in MCS web server. The album artwork can requested at variable sizes, skewed at an angle, and displayed with a reflection using parameters in the HTTP request.

The URL to retrieve cover art is the server IP address on port 80 (Mirage Media Server) or port 5005 (MCS Software)

The HTTP request to this web server to retrieve cover art is **albumart**.

All parameters are optional. Simply issuing an **albumart** request with no parameters to the server address will provide you with the now playing art for the current media on the first output of the device.

Example: `http://192.168.1.10/albumart` will display the album art for the currently playing media on the Mirage Server with the static IP address of 192.168.1.10.

If this address pointed to a PC running MCS, the request would look like:
`http://192.168.1.10:5005/albumart`

To enable your client device to browse thumbnails of albums that are not currently playing, you can also request cover art by the GUID that is supplied to the client during list browsing activity. The syntax is:

```
http://[server[:webport]/albumart?album=[guid]
```

For Example:

`http://192.168.1.10:5005/albumart?album={33432-33432-95909-33423-34430}`
would display the album art for the media identified by the GUID.

HTTP Parameters for Album Art

c	constrain 0=size image to fit height and width 1=constrain to dimension and maintain aspect ratio
guid	unique id of the album, artist, genre, or title
fmt	image format. Valid values are **png** or **jpg.**
instance	the MCS instance GUID
h	image height
w	image width
rfle	reflection elevation
rflh	reflection height
rflo	reflection opacity
rz	reflection rotation (z axis)

This example cover was produced using the following HTTP request:

`http://Mirage1/getart?&h=380&w=300&c=1&rfle=3&rflh=30&rflo=70&rz=15&fmt=png`

Command Reference

General Commands

Banner

Command: **banner**

Response: **Welcome to the Autonomic Media Control Server version 3.0**
Type '?' for help or 'help <command>' for help on <command>.

Displays the connection banner including version information.

SetXMLMode

Command: **setxmlmode [none | lists]**

Sets the protocol response mode to XML.

CLS

Command: **cls**

Clears all characters on the ANSI terminal.

Exit

Command: **exit**

Ends the current session and closes the ANSI terminal.

Help

Command: **help [command] or ? [command]**

In the first form, displays a list of all available commands. If the optional [command] parameter is issued, detailed help will be displayed for the command.

GetVersions

Syntax: **GetVersions**

Example:

Command: **GetVersions**

Response:
```
BeginVersions Total=1
     AhEhSrvr 2.0.2398.788
EndVersions NoMore
```

Returns a list of Autonomic Controls component version numbers on the server.

GetLicenseMessage

Syntax: **`GetLicenseMessage`**

Example:

Command: **`GetLicenseMessage`**

Response:
```
Licensed by Autonomic Controls, Inc to Joe Smith
Demo mode in progress:  12 days remaining
Unlicensed
```

Returns the current license message

Time

Syntax: **Time <format>**

Example:

Command: **Time**

Response: **Time: "Saturday, June 10, 2006 4:03:43 PM"**

Echo's the current system time from the MCS computer. The optional <format> parameter can be used to change the format of the return string.

Valid format codes are:

```
"d" : 08/17/2000
"D" : Thursday, August 17, 2000
"f" : Thursday, August 17, 2000 16:32
"F" : Thursday, August 17, 2000 16:32:32
"g" : 08/17/2000 16:32
"G" : 08/17/2000 16:32:32
"m" : August 17
"r" : Thu, 17 Aug 2000 23:32:32 GMT
"s" : 2000-08-17T16:32:32
"U" :Thursday, August 17, 2000 23:32:32
```

Uptime

Syntax: **`Uptime`**

Example:

Command: **`Uptime`**

Response: **`Uptime "1.02:20:25"`**

Echo's the MCS software uptime from the MCS computer in the format *days.hours:minutes:seconds*.

BrowseEncodings

Syntax: **BrowseEncodings**

Example:

Command: **BrowseEncodings**

Response:

```
BeginEncodings Total=95
      37 "IBM EBCDIC (US-Canada)"
      437 "OEM United States"
      737 "Greek (DOS)"
      775 "Baltic (DOS)"
      850 "Western European (DOS)"
      852 "Central European (DOS)"
     …
     …
      861 "Icelandic (DOS)"
      862 "Hebrew (DOS)"
EndEncodings NoMore
```

Allows for browsing the list of valid text encoding id's.

SetEncoding

Syntax: **SetEncoding**

Example:

Command: **SetEncoding 20105**

Response: **Encoding 20105**

Allows for browsing the list of valid text encoding id's. Encoding 20105 recommended for most applications.

Valid format codes are:

```
37  "IBM EBCDIC (US-Canada)"
437 "OEM United States"
500 "IBM EBCDIC (International)"
708 "Arabic (ASMO 708)"
720 "Arabic (DOS)"
737 "Greek (DOS)"
775 "Baltic (DOS)"
850 "Western European (DOS)"
852 "Central European (DOS)"
855 "OEM Cyrillic"
857 "Turkish (DOS)"
858 "OEM Multilingual Latin I"
860 "Portuguese (DOS)"
861 "Icelandic (DOS)"
862 "Hebrew (DOS)"
863 "French Canadian (DOS)"
864 "Arabic (864)"
865 "Nordic (DOS)"
866 "Cyrillic (DOS)"
869 "Greek, Modern (DOS)"
870 "IBM EBCDIC"
874 "Thai (Windows)"
875 "IBM (Greek Modern)"
1026 "IBM (Turkish Latin-5)"
1047 "IBM Latin-1"
1140 "IBM (US-Canada-Euro)"
1141 "IBM (Germany-Euro)"
1142 "IBM (Denmark-Norway-Euro)"
1143 "IBM (Finland-Sweden-Euro)"
1144 "IBM (Italy-Euro)"
1145 "IBM (Spain-Euro)"
1146 "IBM (UK-Euro)"
1147 "IBM (France-Euro)"
1148 "IBM (International-Euro)"
1149 "IBM (Icelandic-Euro)"
1250 "Central European (Windows)"
1251 "Cyrillic (Windows)"
1252 "Western European (Windows)"
1253 "Greek (Windows)"
1254 "Turkish (Windows)"
1255 "Hebrew (Windows)"
1256 "Arabic (Windows)"
1257 "Baltic (Windows)"
1258 "Vietnamese (Windows)"
10000 "Western European (Mac)"
10004 "Arabic (Mac)"
10005 "Hebrew (Mac)"
10006 "Greek (Mac)"
10007 "Cyrillic (Mac)"
10010 "Romanian (Mac)"
10017 "Ukrainian (Mac)"
10021 "Thai (Mac)"
10029 "Central European (Mac)"
10079 "Icelandic (Mac)"
10081 "Turkish (Mac)"
10082 "Croatian (Mac)"
20105 "Western European (IA5)"
20106 "German (IA5)"
20107 "Swedish (IA5)"
20108 "Norwegian (IA5)"
20127 "US-ASCII"
20269 "ISO-6937"
20273 "IBM (Germany)"
20277 "IBM (Denmark-Norway)"
20278 "IBM (Finland-Sweden)"
20280 "IBM (Italy)"
20284 "IBM (Spain)"
20285 "IBM (UK)"
20290 "IBM (Japanese katakana)"
20297 "IBM (France)"
20420 "IBM (Arabic)"
20423 "IBM (Greek)"
20424 "IBM (Hebrew)"
20833 "IBM (Korean Extended)"
20838 "IBM (Thai)"
20866 "Cyrillic (KOI8-R)"
20871 "IBM (Icelandic)"
20880 "IBM (Cyrillic Russian)"
20905 "IBM (Turkish)"
20924 "IBM Latin-1"
21025 "IBM (Serbian-Bulgarian)"
21866 "Cyrillic (KOI8-U)"
28591 "Western European (ISO)"
28592 "Central European (ISO)"
28593 "Latin 3 (ISO)"
28594 "Baltic (ISO)"
28595 "Cyrillic (ISO)"
28596 "Arabic (ISO)"
28597 "Greek (ISO)"
28598 "Hebrew (ISO-Visual)"
28599 "Turkish (ISO)"
28603 "Estonian (ISO)"
28605 "Latin 9 (ISO)"
29001 "Europa"
38598 "Hebrew"
```

MCS Instance Commands

BrowseInstances

Syntax: **BrowseInstances**

Response Syntax:

Header: **BeginInstances**
Items: **[Instance]**
…
…
Terminator: **EndInstances NoMore**

Example:

Command: **BrowseInstances**

Response:
```
BeginInstances Total=2
  FamilyRoom
  XBOX
EndInstances NoMore
```

Returns a list of current instances. If friendly names have been created in the configuration utility, they will be used, otherwise, the server will return :

sessionname@NIC Address as in **Administrator@00-00-00-00-00-00**

Accounts on the MCS host will always be listed. Extender sessions will only be listed when the extender is on and in the Media Center shell. Note that XBOX 360 extenders will not be listed if they are in game or console mode. To start the XBOX 360 in extender mode, press the green button on the XBOX 360 remote control to start in Media Center mode.

SetInstance

Syntax: **`SetInstance`** *`string`***`[instance_id]`**

Example:

Command: **`SetInstance "Family Room"`**

Response: **`Instance=FamilyRoom`**

Selects a specific MCS Instance for further commands and events

If no MCS Instance is selected via this command or if **[instance_id]=*** then the "Current" instance is the instance running on the console.

See the **`BrowseInstances`** Command for information on how to enumerate the current instances.

Interfacing with the Media Center Shell

(MCS Software Only)

MsgBox

Syntax: **`MsgBox <id> <caption> <message> <buttons> <timeout> <image>`**

Example:

Command:
```
MsgBox    1
          "Garage Door"
          "The garage door is open, would it closed?"
          "Yes;No"
          "20"
```

Response: **`MsgBox 1 1`**

Displays the quote enclosed string in a media center dialog box and waits for user response.

<id>	integer, question id – this will help you to match the response with a question.
<caption>	message Caption
<message>	message for display
<buttons>	semicolon delimited button texts
<timeout>	timeout in seconds defaults to 5.
<image>	UNC path or URL to the PNG-format image to display in the dialog box.

The server will respond in the format Msgbox [*id*] [*Button*], where the id is the integer supplied in the MsgBox command and *Button* is the index of the button that the user selected. (1 based). If the message box times out, no response will be sent.

Feedback

GetStatus

Syntax: **GetStatus**

Example:

Command: **GetStatus**

Response:

```
ReportState FamilyRoom TrackName=A Foggy Day
ReportState FamilyRoom MediaControl=Stop
ReportState FamilyRoom SessionStart=StreamingContentAudio
ReportState FamilyRoom Volume=25
ReportState FamilyRoom TrackTime=1
ReportState FamilyRoom TrackDuration=144
ReportState FamilyRoom TotalTracks=50
ReportState FamilyRoom TrackNumber=8
ReportState FamilyRoom RepeatSet=False
ReportState FamilyRoom CD=False
ReportState FamilyRoom ArtistName=Frank Sinatra
ReportState FamilyRoom MediaName=Duets/Duets II
ReportState FamilyRoom Shuffle=True
ReportState FamilyRoom Running=True
```

See ReportState Message

GetStatus returns a list of all parameters that are typically sent with a StateChanged message. This can be used to prime the IP client's feedback status.

This function should also be called before attempting to command the MCS interface to insure that the MCS shell is running. This can be determined by the **Running=True|False token.**

The messages returned in response to this command differ from event driven **StateChanged** messages only in the leading token **ReportState**. Clients can use these tokens to distinguish between an event that has just occurred and a requested update. The rest of the response can be handled by the same parsing routine.

SubscribeEvents

Syntax: **SubscribeEvents** *String* **<Events>**

Example:

Command: **SubscribeEvents True**

Response: **Events=True**

Turns event messages on or off. (Such as track information, track progress, transport feedback, etc.) If <Events> is omitted, a value of "True" is assumed.

Note: as of Version 3.0.6156:
<Events> may be a comma delimited list of events of interest. i.e.

```
SubscribeEvents "MetaData1,MetaData2,MetaData3,MetaData4"
```

Will restrict event notification to those events listed.

StateChanged Message

Syntax: **`StateChanged <instance> <name>=<value>`**

Example:

Command: **`N/A (see SubscribeEvents)`**

Response:
```
StateChanged FamilyRoom MediaControl=Play
StateChanged FamilyRoom TrackTime=77
StateChanged FamilyRoom TrackTime=78
StateChanged FamilyRoom TrackTime=79
StateChanged FamilyRoom TrackTime=80
StateChanged FamilyRoom TrackTime=81
StateChanged FamilyRoom MediaControl=Stop
```

The StateChanged token indicates that the Media Center Control Server is sending a status update to the client. The =<value> field will only be sent if appropriate.

<instance> Indicates the instance of the event being reported.
<name> Indicates the name of the event being reported.
<value> Indicates any value associated with the event.

Valid Name / Values:

ArtistName	The name of the artist of the currently playing media.
CallingPartyName	Caller ID, the name of the calling party.
CallingPartyNumber	Caller ID, the number of the calling party.
CD	CD Playback initiated
CurrentPicture	The name of the current picture displayed (MyPictures)
DiscWriter_ProgressPercentageChanged	Update on the progress of a CD/DVD recording operation
DiscWriter_ProgressTimeChanged	Update on the progress of a CD/DVD recording operation
DiscWriter_SelectedFormat	Selected recording format for a CD/DVD recording operation.
DiscWriter_Start	CD/DVD recording operation has begun.
DiscWriter_Stop	CD/DVD recording operation has concluded.
DVD	DVD Playback has started
Ejecting	The CD/DVD is ejecting
Error	An error occurred in the MCS shell
GuideLoaded	Downloaded a new guide.
MediaControl= Rewind3	Rewind speed 3 initiated

MediaControl=FF1	Fast Forward speed 1 (slow) initiated
MediaControl=FF2	Fast forward speed 2 (medium) initiated
MediaControl=FF3	Fast forward speed 3 (fast) initiated
MediaControl=NextFrame	The next frame transport control was issued.
MediaControl=Pause	Pause transport command issued
MediaControl=Play	Play transport command issued
MediaControl=PrevFrame	The previous frame transport control was issued.
MediaControl=Rewind1	Rewind speed 1 initiated
MediaControl=Rewind2	Rewind speed 2 initiated
MediaControl=SkipNext	The track was skipped forward
MediaControl=SkipPrev	The track was skipped backwards
MediaControl=SlowMotion1	Slow Motion playback (speed 1) has begun
MediaControl=SlowMotion2	Slow motion playback (speed 2) has begun
MediaControl=SlowMotion3	Slow Motion playback (speed 3) has begun
MediaControl=Stop	Stop transport command issued
MediaName	The name of the currently playing media (all media types)
MediaTime	The total duration of the currently playing media (video, music, or TV)
MediaType	The type of the currently playing media
Navigation=Extensibility	Navigating to a hosted HTML application.
Navigation=FS_DVD	Navigating to Play DVD, or the DVD inset was selected.
Navigation=FS_Home	Navigating to Media Center Start Page.
Navigation=FS_TV	Navigating to My TV, or the TV inset was selected.
Navigation=Guide	Navigating to Guide.
Navigation=Music	Navigating to My Music, or the music inset was selected.
Navigation=Photos	Navigating to My Pictures.
Navigation=Radio	Navigating to My Radio.
Navigation=RecordedShows	Navigating to Recorded Shows or scheduled recording pages
Navigation=Unknown	Unknown Media Center status.
Navigation=Videos	Navigating to My Videos, or the video inset was selected.
ParentalAdvisoryRating	The MPAA rating of the current media
PhoneCall	Incoming phone call event.
Radio	The radio has been activated
RadioFrequency	The frequency of the current radio station
Recording	Status of record mode has changed
RepeatSet	Status of repeat mode changed
Running	The MCS Shell is running
SessionEnd	The MCS shell has ended
SessionStart	The MCS shell has started
Shuffle	Status of shuffle mode changed
StreamingContentAudio	Playback of streaming audio has begun
StreamingContentVideo	Playback of streaming video has begun

TitleNumber	The track number of the current media
TotalTracks	The total number of tracks in the current media set (album)
TrackDuration	The total duration of the current track in seconds
TrackName	The name of the current track
TrackNumber	The number of the current track
TrackTime	The progress of track playback in seconds
TransitionTime	The transition time (between pictures in slideshow)
Visualization	The name of the current visualization
Volume	The current volume level (master MCS volume level)

ReportState Message

Syntax: **ReportState <instance> <name>=<value>**

Example:

Command: **See GetMCSStatus command**

Response:
```
StateChanged FamilyRoom MediaControl=Play
StateChanged FamilyRoom TrackTime=77
StateChanged FamilyRoom TrackTime=78
StateChanged FamilyRoom TrackTime=79
StateChanged FamilyRoom TrackTime=80
StateChanged FamilyRoom TrackTime=81
StateChanged FamilyRoom MediaControl=Stop
```

The ReportState message is sent in response to a GetMCSStatus command.

<instance> Indicates the instance of the event being reported.
<name> Indicates the name of the event being reported.
<value> Indicates any value associated with the event.

Set StateChanged command for valid name/value pairs.

Media Center Interface Navigation

(MCS Software Only)

Navigate

Syntax: **`Navigate <screen>`**

Example:

Command: **`Navigate MyPictures`**

Response: **`StateChanged FamilyRoom Navigation=Pictures`**

Instructs the Media Center shell on the host computer to navigate to the specified screen. If the client is subscribed to events (see SubscribeEvents), a StateChanged message will be sent confirming the navigation.

Valid values for screen:

FMRadio	Sets Media Center to FM Radio
InternetRadio	Sets Media Center to Internet Radio
LiveTV	Sets Media Center to Live Television
MorePrograms	Sets Media Center to More Programs
MusicAlbums	Sets Media Center to Music Albums
MusicArtists	Sets Media Center to Music Artists
MusicSongs	Sets Media Center to Music / Songs
MyMusic	Sets Media Center to My Music
MyPictures	Sets Media Center to My Pictures
MyTV	Sets Media Center to My TV
MyVideos	Sets Media Center to MY Videos
RecordedTV	Sets Media Center to Recorded TV
RecorderStorageSettings	Sets Media Center to RecorderStorageSettings
ScheduledTVRecordings	Sets Media Center to Scheduled TV Recordings
SlideShow	Sets Media Center to Slide Show
Start	Sets Media Center to Start Page
TVGuide	Sets Media Center to TV Guide
Visualizations	Sets Media Center to Visualizations
PhotoDetails	Sets Media Center to Photo Details
SlideShow	Initiates Media Center as Slide Show
SlideShowSettings	Sets Media Center Slide Show Settings

Transport Control

Transport Commands

Syntax: **`<command>`**
Response Syntax: **`<command> OK`**

Example:

Command: **`Play`**

Response: **`Play OK`**

Issues the specified transport control

Valid values for <command>:

Play	Instructs Media Player to PLAY the media transport.
Stop	Instructs Media Player to STOP the media transport.
Pause	Instructs Media Player to PAUSE the media transport.
PlayPause	Pause when Playing / Play when Paused
SkipNext	Commands Media Player to move to the next song in the queue. (with wraparound)
SkipPrevious	Commands Media Player to move to the previous song in the queue (with wraparound)
Shuffle [true | false | toggle]	Turns Random Mode on or Off
Repeat [true | false | toggle]	Turns Repeat Mode on or Off

Browse Media Commands

BrowseAlbums

Syntax: **BrowseAlbums <start> <reqcount>**

Response Syntax:

Header: **BeginAlbums Total=[*count*]**
Items: **Album [GUID] [Name]** …
…
…
Terminator: **EndAlbums [More]|[NoMore]**

Example:

Command: **BrowseAlbums 11 10**

Response:

```
BeginAlbums Total=170
  Album {a7ca-47a1-bc2b-f4927bbf2ad8} "Chopin:  Ballades & Scherzos"
  Album {3ce8-4aeb-99f8-43a1a33d30cc} "Chopin: The Complete Nocturnes"
  Album {f6f4-4416-a1c3-af2b7bba0e9c} "Cieli Di Toscana"
  Album {6c61-454e-9f0e-0a28ae8dde95} "Clapton Chronicles"
  Album {5147-4b26-b31f-720230af4278} "Claude Debussy: Preludes"
  Album {ebaa-4b38-b4d6-c18d97ff962e} "Come Away With Me"
  Album {93a1-4c0b-b8c1-51c3447f05c8} "Come on Over [International]"
  Album {0434-49e4-8c2b-8887edeb6258} "Cry Like a Rainstorm"
  Album {f8b8-422a-94e4-923e591fb6b2} "The Dance"
  Album {bc76-4322-8feb-7cf72fc6230b} "Dances with Wolves"
EndAlbums More
```

Allows browsing the media library belonging to the current instance.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the album

If <start> and <reqcount> are omitted, all albums that match the current filter will be returned. (see SetMusicFilter, GetMediaFilter).

Refer to **List Processing** and **Asynchronous Processing** topics at the beginning of the command reference.

BrowseArtists

Syntax: **BrowseArtists <start> <reqcount>**

Response Syntax:

Header: **BeginArtists Total=[*count*]**
Items: **Artist [GUID] [Name]** …
…
…
Terminator: **EndArtists [More]|[NoMore]**

Example:

Command: **BrowseArtists 1 10**

Response:
```
BeginArtists Total=344
  Artist {7d9761cb-ea80-43dd-b63e-90a0b6bd8017} "Unknown"
  Artist {ef504364-e0ac-4f1a-a276-09dc087bfe6e} "NSYNC"
  Artist {e28d9be3-6cbd-4678-9205-eed6b90f714e} "3rd Party"
  Artist {4930f19f-f942-4017-9293-7618c14ef94c} "5ive"
  Artist {9ed91b23-153d-4857-bd35-b72d3f83d8e9} "7 Mile"
  Artist {0af5a33d-d486-4d06-9bee-7f2b0cead68f} "112"
  Artist {5279b9bc-0427-49dc-a1ff-f5e8d17f5717} "Aaron Copland"
  Artist {87c1d18b-079e-4d10-8564-0a7a5e49b65d} "Aaron Hall"
  Artist {902e1012-710d-46eb-861d-7a75ca81b96e} "Aaron Neville"
  Artist {bd3c4dde-b07c-4123-908d-f00d76311ae8} "Abbey Simon"
EndArtists More
```

Allows for browsing the media library belonging to the current instance.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the artist

If <start> and <reqcount> are omitted, all artists that match the current filter will be returned. (see SetMusicFilter, GetMediaFilter).

Refer to **List Processing** and **Asynchronous Processing** topics at the beginning of the command reference.

BrowseGenres

Syntax: **BrowseGenres <start> <reqcount>**

Response Syntax:

Header: **BeginGenres Total=[*count*]**
Items: **Genre [GUID] [Name]** …
…
…
Terminator: **EndGenres [More]|[NoMore]**

Example:

Command: **BrowseGenres 11 10**

Response:

```
BeginGenres Total=10
  Genre {90c4469e-fa42-4b3d-b9e1-88f0bfe02df9} "Classical"
  Genre {1f75c29d-cd2f-4dde-881b-6ba855222afb} "Country"
  Genre {161ebefe-bc24-41d4-8e97-8b338f93ec05} "Dance / Electronic"
  Genre {69d2c275-e25f-4b7c-a2ca-e2975960636c} "Hip-Hop"
  Genre {e71ed659-e54b-4c5d-99c6-35a1487727c7} "Jazz"
  Genre {33d4d6d6-a3fa-4797-80c9-755ac5c79eae} "New Age"
  Genre {ba3d9dff-55f1-41e3-9660-3ee7054c9a52} "Pop"
  Genre {2e2c338d-24d2-4860-8be6-ccbe8fdfa119} "R&B"
  Genre {9c5b5efe-3934-437b-8a83-7bb903be6d25} "Rock"
  Genre {d61f8edd-d7bf-4d79-8aa5-a91c857ffd2c} "Soundtrack"
EndGenres NoMore
```

Allows browsing the media library belonging to the current instance.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the genre

If <start> and <reqcount> are omitted, all genres that match the current filter will be returned. (see SetMusicFilter, GetMediaFilter).

Refer to **List Processing** and **Asynchronous Processing** topics at the beginning of the command reference.

BrowseNowPlaying

Syntax: **BrowseNowPlaying <start> <reqcount>**

Response Syntax:

Header: **BeginNowPlaying Total=[*count*]**
Items: **Title [GUID] [Name] [time]**
…
…
Terminator: **EndNowPlaying [More]|[NoMore]**

Example:

Command: **BrowseNowPlaying 1 10**

Response:

```
BeginNowPlaying Total=98
  Title {3216-457f-87c2-b5da6541b895} "A Foggy Day" "00:02:25"
  Title {92ca-4ccf-983c-7b2760cca26d} "All or Nothing at All" "00:04:00"
  Title {4306-8067-6154c7eb5d7b} "All the Way" "00:03:54"
  Title {f3f7-4d65-b616-6d98d3e442a6} "All the Way/One for My Baby" "00:06:04"
  Title {40bc-48c6-be87-4586684d95c1} "Bewitched" "00:03:32"
  Title {c950-4521-acb7-026c9dc0ed8b} "Come Fly with Me" "00:03:09"
  Title {13a3-4f30-8980-4a6ad5fa9569} "Come Rain or Come Shine" "00:04:05"
  Title {c24f-42d9-a19e-98826e66c747} "Embraceable You" "00:03:46"
  Title {94ed-4b7d-b419-ea06d0d21436} "Fly Me to the Moon" "00:03:07"
  Title {104f-46e6-8fc4-6371aa86e14a} "Fly Me to the Moon" "00:02:32"
EndNowPlaying More
```

Allows browsing the queue for the current instance.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the track
<time> the length of the track

If <start> and <reqcount> are omitted, all titles in the queue will be returned.

Refer to **List Processing** and **Asynchronous Processing** topics at the beginning of the command reference.

BrowsePlaylists

Syntax: **BrowsePlaylists <start> <reqcount>**

Response Syntax:

Header: **BeginPlaylists Total=[*count*]**
Items: **Playlist [GUID] [Name]** …
…
…
Terminator: **EndPlaylists [More]|[NoMore]**

Example:

Command: **BrowsePlaylists**

Response:
```
BeginPlaylists Total=5
  Playlist {90e56f8e-c900-44fc-8cd7-fdac81b6f215} "Diana"
  Playlist {5fd3175e-2688-4617-bcf7-575c71b1e0bc} "Napster Tracks"
  Playlist {efdd1f28-63ff-4dfd-b244-b2f6868af4a6} "Popular"
  Playlist {c386107f-5e7e-45c4-a270-42c6de8aeb84} "Soft Background"
  Playlist {039a8699-506f-4567-a0ac-90fc2778a2f5} "The Movies!"
EndPlaylists NoMore
```

Allows browsing play lists in the current instance.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the play list

If <start> and <reqcount> are omitted, all play lists that match the current filter will be returned. (see SetMusicFilter, GetMediaFilter).

Refer to **List Processing** and **Asynchronous Processing** topics at the beginning of the command reference.

BrowseRadioGenres

Syntax: **BrowseRadioGenres <start> <reqcount>**

Response Syntax:

Header: **BeginRadioGenres Total=[*count*]**
Items: **RadioGenre [GUID] [Name]** …
…
…
Terminator: **EndRadioGenres [More]|[NoMore]**

Example:

Command: **BrowseRadioGenres 1 10**

Response:

```
BeginRadioGenres Total=3
  RadioGenre {90c4469e-fa42-4b3d-b9e1-88f0bfe02df9} "Comedy"
  RadioGenre {1f75c29d-cd2f-4dde-881b-6ba855222afb} "Pop"
  RadioGenre {161ebefe-bc24-41d4-8e97-8b338f93ec05} "Rock"
EndRadioGenres NoMore
```

Allows browsing the media library belonging to the current instance for Radio Genres.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the radio Genre

If <start> and <reqcount> are omitted, all radio sources that match the current filter will be returned. (see SetRadioFilter).

Refer to **List Processing**, **Asynchronous Processing,** and **PickLists** topics at the beginning of the command reference.

BrowseRadioStations

Syntax: **BrowseRadioStations <start> <reqcount>**

Response Syntax:

Header: **BeginRadioStations Total=[*count*]**
Items: **RadioStation [GUID] [Name]** …
…
…
Terminator: **EndRadioStations [More]|[NoMore]**

Example:

Command: **BrowseRadioStations 1 5**

Response:

```
BeginRadioStations Total=136 Start=1 Alpha=1 Caption="Radio Stations"
  RadioStation {d47f4e8e-040d-46e1-bf36-1edcf645f575} "1st Wave"
  RadioStation {a84d89c6-47b0-4d17-aa60-0edf88f11901} "20 on 20"
  RadioStation {bca387a0-04b9-443b-b01c-f5a72903b93e} "40s on 4"
  RadioStation {44690865-3c22-4635-9db5-68e297977a4b} "50s on 5"
  RadioStation {c63303a9-deec-457d-b09e-d2ae260ae033} "60s on 6"
EndRadioStations More
```

Allows browsing the media library belonging to the current instance for Radio Stations.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the radio Genre

If <start> and <reqcount> are omitted, all radio sources that match the current filter will be returned. (see SetRadioFilter).

Refer to **List Processing**, **Asynchronous Processing,** and **PickLists** topics at the beginning of the command reference.

BrowseRadioSources

Syntax: **BrowseRadioSources <start> <reqcount>**

Response Syntax:

Header: **BeginRadioSources Total=[*count*]**
Items: **RadioSource [GUID] [Name]** …
…
…
Terminator: **EndRadioSources [More]|[NoMore]**

Example:

Command: **BrowseRadioSources 1 10**

Response:
```
BeginRadioSources Total=4
  RadioSource {90c4469e-fa42-4b3d-b9e1-88f0bfe02df9} "Pandora"
  RadioSource {1f75c29d-cd2f-4dde-881b-6ba855222afb} "Sirius"
  RadioSource {161ebefe-bc24-41d4-8e97-8b338f93ec05} "RadioTime"
EndRadioRouces NoMore
```

Allows browsing the media library belonging to the current instance for Radio Sources.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the radio source

If <start> and <reqcount> are omitted, all radio sources that match the current filter will be returned. (see SetRadioFilter).

Refer to **List Processing**, **Asynchronous Processing,** and **PickLists** topics at the beginning of the command reference.

BrowseTitles

Syntax: **BrowseTitles <start> <reqcount>**

Response Syntax:

Header: **BeginTitles Total=[*count*]**
Items: **Title [GUID] [Name] [time]**
…
…
Terminator: **EndTitles [More]|[NoMore]**

Example:

Command: **BrowseTitles 1 10**

Response:

```
BeginTitles Total=98
  Title {3216-457f-87c2-b5da6541b895} "A Foggy Day" "00:02:25"
  Title {92ca-4ccf-983c-7b2760cca26d} "All or Nothing at All" "00:04:00"
  Title {4306-8067-6154c7eb5d7b} "All the Way" "00:03:54"
  Title {f3f7-4d65-b616-6d98d3e442a6} "All the Way/One for My Baby" "00:06:04"
  Title {40bc-48c6-be87-4586684d95c1} "Bewitched" "00:03:32"
  Title {c950-4521-acb7-026c9dc0ed8b} "Come Fly with Me" "00:03:09"
  Title {13a3-4f30-8980-4a6ad5fa9569} "Come Rain or Come Shine" "00:04:05"
  Title {c24f-42d9-a19e-98826e66c747} "Embraceable You" "00:03:46"
  Title {94ed-4b7d-b419-ea06d0d21436} "Fly Me to the Moon" "00:03:07"
  Title {104f-46e6-8fc4-6371aa86e14a} "Fly Me to the Moon" "00:02:32"
EndTitles More
```

Allows browsing titles in the media library of current instance.

<start> specifies where to start the listing.
<reqcount> specifies the total items requested

<GUID> a globally unique ID used for playback commands and further browsing.
<name> the name of the track
<time> the length of the track

If <start> and <reqcount> are omitted, all genres that match the current filter will be returned. (see SetMusicFilter, GetMediaFilter).

Refer to **List Processing** and **Asynchronous Processing** topics at the beginning of the command reference.

Play Media Commands

PlayAlbum

Syntax: **`PlayAlbum [guid or album] [enqueue]`**

Example:

Command:
```
PlayAlbum {ab3794df-30a8-4a19-b5cf-c75740743ffa} True
PlayAlbum Duets True
PlayAlbum "All For You" False
```

Response: **`PlayAlbum OK`**

Plays all tracks in the specified Album. You may specify a GUID or an Album Name enclosed in quotes. If you specify an album name that contains embedded spaces, you must also enclose the album name in quotes. The <guid> resource can be obtained with a BrowseAlbums command.

<guid> a globally unique ID obtained with BrowseAlbums.
Note: This may optionally be a Title guid in which case the Album for that title is queued and playback begins with that title within the album.

<album> the name of an album to play or queue.

<enqueue> If "true", the tracks will be added to the queue without interrupting playback
If "false", the queue will be cleared before the tracks are added.

If there are no songs in the current queue, or if <enqueue> is "true", then playback of the first track will begin automatically.

PlayArtist

Syntax: **`PlayArtist [guid or Artist] [enqueue]`**

Example:

Command:
```
PlayArtist {ab3794df-30a8-4a19-b5cf-c75740743ffa} True
PlayArtist Seal True
PlayArtist "Frank Sinatra" True
```

Response: **`PlayArtist OK`**

Plays all tracks of the specified Artist. You may specify a GUID or an Artist Name. If you specify an artist name that contains embedded spaces, you must also enclose the name in quotes. The <guid> resource must be obtained with a BrowseArtists command.

<guid> a globally unique ID obtained with **`BrowseArtists`**.
<artist> the name of an artist to play or queue.
<enqueue> If "true", the tracks will be added to the queue without interrupting playback
If "false", the queue will be cleared before the tracks are added.

If there are no songs in the current queue, or if <enqueue> is "true", then playback of the first track will begin automatically.

PlayGenre

Syntax: **`PlayGenre [guid or genre] [enqueue]`**

Example:

Command:
```
PlayGenre {ab3794df-30a8-4a19-b5cf-c75740743ffa} True
PlayGenre Jazz True
PlayGenre "R & B" True
```
Response: **`PlayGenre OK`**

Plays all tracks of the specified Genre. You may specify a GUID or an genre name. If you specify an genre name that contains embedded spaces, you must also enclose the name in quotes. The <guid> resource must be obtained with a BrowseGenres command.

<guid> a globally unique ID obtained with **`BrowseGenres`**.
<genre> the name of a genre to play or queue.
<enqueue> If "true", the tracks will be added to the queue without interrupting playback
If "false", the queue will be cleared before the tracks are added.

If there are no songs in the current queue, or if <enqueue> is "true", then playback of the first track will begin automatically.

PlayPlaylist

Syntax: **`PlayPlaylist [guid or Playlist] [enqueue] [startGuid]`**

Example:

Command:
```
PlayPlaylist {ab3794df-30a8-4a19-b5cf-c75740743ffa} True
PlayPlaylist Party True
PlayPlaylist "My Favorites" True
```

Response: **`PlayPlaylist OK`**

Plays all tracks of the specified Playlist. You may specify a guid or an playlist Name. If you specify an playlist name that contains embedded spaces, you must also enclose the name in quotes. The <guid> resource must be obtained with a BrowsePlaylists command.

<guid>	a globally unique ID obtained with **`BrowsePlaylists`**.
<playlist>	the name of a playlist to play or queue
<enqueue>	If "true", the tracks will be added to the queue without interrupting playback If "false", the queue will be cleared before the tracks are added
<startGuid>	When specified, playback will start with this track

If there are no songs in the current queue, or if <enqueue> is "true", then playback of the first track will begin automatically.

PlayTitle

Syntax: **`PlayTitle [guid or title] [enqueue]`**

Example:

Command:
```
PlayTitle {ab3794df-30a8-4a19-b5cf-c75740743ffa} True
PlayTitle Summertime True
PlayTitle "Yellow Brick Road" False
```

Response: **`PlayTitle OK`**

Plays all tracks of the specified Title. You may specify a guid or a song name. If you specify a song name that contains embedded spaces, you must also enclose the name in quotes. The <guid> resource must be obtained with a BrowseTitles command.

<guid> a globally unique ID obtained with **`BrowseTitles`**.
<title> the name of a song to play or queue
<enqueue> If "true", the tracks will be added to the queue without interrupting playback
If "false", the queue will be cleared before the tracks are added.

If there are no songs in the current queue, or if <enqueue> is "true", then playback of the first track will begin automatically.

JumpToNowPlayingItem

Syntax: **`JumpToNowPlayingItem [guid or index]`**

Example:

Command: **`JumpToNowPlayingItem {ab3794df-30a8-4a19-b5cf-c75740743ffa}`**
`JumpToNowPlayingItem 10`

Response: **`JumpToNowPlayingItem OK`**

Jumps directly to a title in the now playing list and begins playback.

Form 1 **`JumpToNowPlayingItem [guid]`**

Jumps to the title specified by [guid]. The [guid] resource is provided in response to the BrowseNowPlaying command.

Form 2 **`JumpToNowPlayingItem [index]`**

Jumps to the title specified by [index], which is the 1 based count from the top of the queue.

RemoveNowPlayingItem

Syntax: **`RemoveNowPlayingItem [guid or index]`**

Example:

Command: **`RemoveNowPlayingItem {ab3794df-30a8-4a19-b5cf-c75740743ffa}`**
`RemoveNowPlayingItem 10`

Response: **`RemoveNowPlayingItem OK`**

Jumps directly to a title in the now playing list and begins playback.

Form 1 **`RemoveNowPlayingItem [guid]`**

Removes the title specified by [guid] from the now playing queue. The [guid] resource is provided in response to the Browse commands.

Form 2 **`RemoveNowPlayingItem [index]`**

Removes the title specified by [index] from the now playing queue, which is the 1 based count from the top of the queue.

PlayRadioStation

Syntax: **`PlayRadioStation [guid or title]`**

Example:

Command: **`PlayRadioStation {ab3794df-30a8-4a19-b5cf-c75740743ffa}`**
`PlayRadioStation "40s on 4"`

Response: **`PlayRadioStation OK`**

You may specify a guid or a station name. If you specify a station name that contains embedded spaces, you must also enclose the name in quotes. The <guid> resource must be obtained with a BrowseRadioTitles command.

<guid> a globally unique ID obtained with **`BrowseRadioStations`**.
<title> the name of a radio station to play or queue

Playing a radio station always clears the now playing queue.

Filter Media Library Commands

SetMusicFilter

Syntax: `SetMusicFilter [tag]=([guid] | serarch=[string]) | [searchstring] | Clear`

Example:

Command:
```
SetMusicFilter Album={75ecb109-df1b-4e0f-8cbb-584017ef28da}
SetMusicFilter Artist="Peter Frampton"
SetMusicFilter Search="*Diana*"
SetMusicFilter Clear
```

Response: `MusicFilter Album={75ecb109-df1b-4e0f-8cbb-584017ef28da}`

SetMusicFilter Filters future list requests. Text strings may be substituted for [guid], but this practice is not recommended for filters other than Search, as there is no guarantee that the text string will be unique within the media library, in which case the server will return the first matching entry.

Issuing successive SetMusicFilter commands are additive to the filter. This is useful for providing users with a browsing interface with drill down capabilities.

Form 1 `SetMusicFilter [tag] = [guid]`
The [guid] resource must be obtained from one of the Browse commands. [guid] strings are not guaranteed to persist across sessions.

Valid Values for [tag] are: Artist, Album, Genre, Playlist, or Title

Form 2 `SetMusicFilter [tag] = [string]`
Finds exact matches for tag=string. Case sensitive.

Form 3 `SetMusicFilter Search=[searchstring]`
Entering a search string will filter all subsequent Browse commands to items matching the string. The [*] wildcard character is allowed, so `"*Diana*"`, will find all items with the workd "Diana" in them, while `"Diana*"` will find all items that *begin* with "Diana". Not Case Sensitive.

Form 4 `SetMusicFilter Clear`
Accumulated filters can be cleared with a single `SetMusicFilter Clear` command

SetRadioFilter

Syntax: `SetRadioFilter ([tag]=[guid]) | Clear`

Example:

Command:
```
SetRadioFilter Source={fbbcedb1-af64-4c3f-bfe5-000000000020}
SetRadioFilter Genre={a24751f5-2d38-4b63-abf0-b4892c126e83}
SetMusicFilter Clear
```

Response: `RadioFilter Ok "RadioTime"`

SetRadioFilter Filters future list requests. Issuing successive SetRadioFilter commands are additive to the filter. This is useful for providing users with a browsing interface with drill down capabilities.

Form 1 `SetMusicFilter [tag] = [guid]`
The [guid] resource must be obtained from one of the Browse commands. [guid] strings are not guaranteed to persist across sessions.

Valid Values for [tag] are: Source, Genre

Form 2 `SetMusicFilter Clear`
Accumulated filters can be cleared with a single `SetRadioFilter Clear` command

IR Commands

SendKeys

(MCS Software Only)

Syntax: **`SendKeys [irkey]`**

Example:

Command: **`SendKeys 1`**

`Response:` **`NA`**

Sends the specified IR key to the server to be executed on the MCS instance as though the user had sent the command from the hand held remote control.

These commands are fundamentally different from other commands in the protocol that seemingly overlap. The action taken by MCS in response to an **SendKeys** command will be determined by the current MCS application.

For example, an MCS add-in application might use the next and previous buttons on the remote to allow the user to navigate a list. In this example, issuing a **SendKeys Replay** would be interpreted by the add-in as a list navigation and would have a different effect than issuing the **SkipPrev** command listed in the **Transport** section of this protocol, which will always move to the next track in the current media queue.

Navigation	Transport:	AV and Power Control	Data Entry:
Home	Play	Volume+	0
Up	Pause	Volume-	1
Down	Stop	Chan/Page+	2
Left	Record	Chan/Page-	3
Right	FastForward	Mute	4
OK	Rewind	DVDMenu	5
Back	Skip	Standby	6
Details	Replay		7
Guide			8
Jump			9
MoreInfo			Clear
			Enter