

*Multimedia Terminal Framework
Reference Guide*



RADVISION

Delivering the Visual Experience

NOTICE

© 2001-2007 RADVISION Ltd. All intellectual property rights in this publication are owned by RADVISION Ltd. and are protected by United States copyright laws, other applicable copyright laws and international treaty provisions. RADVISION Ltd. retains all rights not expressly granted.

The publication is RADVISION confidential. No part of this publication may be reproduced in any form whatsoever or used to make any derivative work without prior written approval by RADVISION Ltd.

No representation of warranties for fitness for any purpose other than what is specifically mentioned in this guide is made either by RADVISION Ltd. or its agents.

RADVISION Ltd. reserves the right to revise this publication and make changes without obligation to notify any person of such revisions or changes. RADVISION Ltd. may make improvements or changes in the product(s) and/or the program(s) described in this documentation at any time.

If there is any software on removable media described in this publication, it is furnished under a license agreement included with the product as a separate document. If you are unable to locate a copy, please contact RADVISION Ltd. and a copy will be provided to you.

Unless otherwise indicated, RADVISION registered trademarks are registered in the United States and other territories. All registered trademarks recognized.

For further information contact RADVISION or your local distributor or reseller.

Multimedia Terminal Framework version 2.5.1.54, November, 2007

Publication 9

<http://www.radvision.com>

CONTENTS

PART 1: MDM CONTROL

1 *RvMdmDigitMap Module*

What's in this Chapter	3
RvMdmDigitMap General Control Functions	4
rvMdmDigitMapConstruct()	5
rvMdmDigitMapConstructA()	6
rvMdmDigitMapCopy()	7
rvMdmDigitMapDestruct()	8
RvMdmDigitMap Accessor Functions	9
rvMdmDigitMapAddPattern()	10
rvMdmDigitMapSetLongTimeout()	11
rvMdmDigitMapSetShortTimeout()	12
rvMdmDigitMapSetStartTimeout()	13
RvMdmDigitMap Type Definitions	14
RvMdmDigitMapMatchType	15

2 *RvMdmDigitPosition Module*

What's in this Chapter	17
RvMdmDigitPosition General Control Functions	18
rvMdmDigitPositionConstruct()	19
rvMdmDigitPositionDestruct()	20
RvMdmDigitPosition Accessor Functions	21
rvMdmDigitPositionAddEvents()	22
rvMdmDigitPositionSetMultipleFlag()	23
rvMdmDigitPositionSetTimerMode()	24

	RvMdmDigitPosition Type Definitions	25
	RvMdmDigitPositionTimerMode	26
3	<i>RvMdmDigitString Module</i>	
	What's in this Chapter	27
	RvMdmDigitString General Control Functions	28
	rvMdmDigitStringConstruct()	29
	rvMdmDigitStringDestruct()	30
	RvMdmDigitString Accessor Functions	31
	rvMdmDigitStringAddElement()	32
4	<i>RvMdmEvent Module</i>	
	What's in this Chapter	33
	RvMdmEvent Functions	34
	RvMdmEvent	35
	rvMdmEventGetParameterList()	36
	rvMdmEventGetName()	37
5	<i>RvMdmMediaStreamDescr Module</i>	
	What's in this Chapter	39
	RvMdmMediaStreamDescr Accessor Functions	40
	rvMdmMediaStreamDescrGetControlParameters()	41
	rvMdmMediaStreamDescrGetLocalDescr()	42
	rvMdmMediaStreamDescrGetMode()	43
	rvMdmMediaStreamDescrGetRemoteDescr()	44
	rvMdmMediaStreamDescrGetReserveGroupMode()	45
	rvMdmMediaStreamDescrGetReserveValueMode()	46
	rvMdmMediaStreamDescrReportControlParameters()	47
	rvMdmMediaStreamDescrReportLocalDescr()	48
	rvMdmMediaStreamDescrReportRemoteDescr()	49

6 *RvMdmPackageItem Module*

What's in this Chapter	51
RvMdmPackageItem General Control Functions	52
rvMdmPackageItemConstruct()	53
rvMdmPackageItemConstructA()	54
rvMdmPackageItemConstructCopy()	55
rvMdmPackageItemCopy()	56
rvMdmPackageItemDestruct()	57
rvMdmPackageItemEqual()	58
RvMdmPackageItem Accessor Functions	59
rvMdmPackageItemGetItem()	60
rvMdmPackageItemGetPackage()	61

7 *RvMdmParameterList Module*

What's in this Chapter	63
RvMdmParameterList General Control Functions	64
rvMdmParameterListConstruct()	65
rvMdmMdmParameterListConstructA()	66
rvMdmParameterListConstructCopy()	67
rvMdmParameterListCopy()	68
rvMdmParameterListDestruct()	69
RvMdmParameterList Accessor Functions	70
rvMdmParameterListForEach()	71
rvMdmParameterListGet()	72
rvMdmParameterListGet2()	73
rvMdmParameterListIsEmpty()	74
rvMdmParameterListSet()	75

8 *RvMdmParameterValue Module*

What's in this Chapter	77
RvMdmParameterValue General Control Functions	78
rvMdmParameterValueConstruct()	79
rvMdmParameterValueConstructA()	80
rvMdmParameterValueConstructCopy()	81

rvMdmParameterValueConstructList()	82
rvMdmParameterValueConstructListA()	83
rvMdmParameterValueDestruct()	84
RvMdmParameterValue Accessor Functions	85
rvMdmParameterValueAddToList()	86
rvMdmParameterValueGetListSize()	87
rvMdmParameterValueGetListValue()	88
rvMdmParameterValueGetType()	89
rvMdmParameterValueGetValue()	90

9 *RvMdmSignal Module*

What's in this Chapter	91
RvMdmSignal Accessor Functions	92
rvMdmSignalGetId()	93
rvMdmSignalGetPkg()	94
rvMdmSignalGetArguments()	95
RvMdmSignal List of Packages and Events	96
DISPLAY PACKAGE (dis)	97
KEY PACKAGE (kp)	98
FUNCTION KEY PACKAGE (kf)	100
INDICATOR PACKAGE (ind)	101
BASIC DTMF GENERATOR PACKAGE (dg)	102
DTMF DETECTION PACKAGE (dd)	103
CALL PROGRESS TONES GENERATOR PACKAGE (cg)	105
ANALOG LINE SUPERVISION PACKAGE (al)	106
RADVISION CALL CONTROL PACKAGE (rvcc)	107

10 *RvMdmTerm Module*

What's in this Chapter	109
RvMdmTerm Functions	110
rvMdmTermGetId()	111
rvMdmTermGetType()	112
rvMdmTermGetUserData()	113
rvMdmTermProcessEvent()	114

rvMdmTermSetUserData()	116
rvMdmTermPropertiesSetPresentationInfo()	117
rvMdmTermPropertiesSetPhoneNumbers()	118
rvMtfTerminalMakeCall()	119
rvMdmTermModifyMedia()	120

11 *RvMdmTermClass Module*

What's in this Chapter	123
RvMdmTermClass Functions	124
rvMdmTermClassAddMediaCapabilities()	125
rvMdmTermClassAddSupportedPkg()	127
rvMdmTermClassRegisterCreateMediaCB()	128
rvMdmTermClassRegisterDestroyMediaCB()	129
rvMdmTermClassRegisterModifyMediaCB()	130
rvMdmTermClassRegisterPlaySignalCB()	131
rvMdmTermClassRegisterStartSignalCB()	132
rvMdmTermClassRegisterStopSignalCB()	133
rvMdmTermClassRegisterMapDialStringToAddressCB()	134
rvMdmTermClassRegisterMatchDialStringCB()	135
rvMdmTermClassClearMediaCapabilities()	136
rvMdmTermClassRegisterRegisterPhysTermCompletedCB()	137
rvMdmTermClassRegisterUnregisterTermCompletedCB()	138
rvMdmTermClassRegisterModifyMediaCompletedCB()	139

12 *RvMdmTermDefaultProperties Module*

What's in this Chapter	141
RvMdmTermDefaultProperties General Control Functions	142
rvMdmTermDefaultPropertiesConstruct()	143
rvMdmTermDefaultPropertiesDestruct()	144
RvMdmTermDefaultProperties Accessor Functions	145
rvMdmTermDefaultPropertiesSetDigitMap()	146
RvMdmTermDefaultPropertiesSetPassword()	147
RvMdmTermDefaultPropertiesSetUsername()	148

13 *RvMdmTermMgr Module*

What's in this Chapter	149
RvMdmTermMgr General Control Functions	150
rvMdmTermMgrDestruct()	151
rvMdmTermMgrRegisterAllTermsToNetwork()	152
rvMdmTermMgrRegisterTermToNetwork()	153
rvMdmTermMgrUnregisterTermFromNetwork()	154
rvMdmTermMgrRegisterEphemeralTermination()	155
rvMdmTermMgrRegisterPhysicalTermination()	157
rvMdmTermMgrStart()	159
rvMdmTermMgrStop()	160
rvMdmTermMgrUnregisterTermination()	161
rvMdmTermMgrRegisterPhysicalTerminationAsync()	162
rvMdmTermMgrUnregisterTerminationAsync()	164
RvMdmTermMgr Accessor Functions	165
rvMdmTermMgrFindTermination()	166
rvMdmTermMgrForEachPhysicalTerm()	167
rvMdmTermMgrGetIdleTermination()	168
rvMdmTermMgrGetPackage()	169
rvMdmTermMgrGetUserData()	170
rvMdmTermMgrRegisterConnectCB()	171
rvMdmTermMgrRegisterDeleteEphTermCB()	172
rvMdmTermMgrRegisterDisconnectCB()	173
rvMdmTermMgrRegisterSelectTermCB()	174
rvMdmTermMgrSetUserData()	175
rvMdmTermMgrCreateTermClass()	176
RvMdmTermMgr Callback Functions	177
RvMdmTermRegisterPhysTermCompletedCB()	178
RvMdmTermUnregisterTermCompletedCB()	179
RvMdmTermUnregisterTermFromNetworkCompletedCB()	180

14 *Call Forward Module*

What's in this Chapter	181
Call Forward Functions	182
rvCCCfwGetTypeNumber()	183

Call Forward Callbacks	184
rvlppCfwActivateCompletedCB()	185
rvlppCfwDeactivateCompletedCB()	187
Call Forward Structures	188
RvlppCfwCBs	189
RvlppCfwCfg	190
Call Forward Type Definitions	191
RvlppCfwType	192
RvlppCfwReturnReasons	193

15 *Logger Module*

What's in this Chapter	195
Logger Functions	196
lppLogInit()	197
lppLogReload()	198
lppLogEnd()	199
lppLogMessage()	200
Logger Type Definitions	201
lppLogSourceFilterElm	202

16 *Enumerated Types*

What's in this Chapter	203
Enumerated Types	204
RvMdmSignalType	205
RvMdmStreamDirection	206
RvMdmStreamMode	207
RvMdmTermType	209
RvMdmRelation	210
RvCCTerminalState	211
RvCCConnState	212
RvCCTermConnState	216
RvCCMediaState	218
RvCCConnType	219
RvCCCallState	220

RvCCTerminalType	221
RvCCTerminalEvent	223
RvCCEventCause	230

PART 2: SIP CONTROL

17 *SIP Control General*

What's in this Chapter	237
SIP Control General Functions	238
rvlppSipSystemInit()	239
rvlppSipSystemEnd()	240
rvlppSipRegisterExtCbks	241
rvlppSipInitConfig()	242
rvlppSipStackInitialize()	243
rvlppSipPhoneConstruct()	244
rvlppSipControlGetRegistrarAddress()	245
rvlppSipControlGetUserDomain()	246
rvlppSipControlGetOutboundProxyAddress()	247
rvlppSipControlGetExtUserData()	248
rvlppSipControlGetTransportType()	249
rvlppSipControlGetStackHandle()	250
rvlppSipControlGetStackCallbacks()	251
rvlppSipControlMsgSetSdpBody()	252
RvlppSipStackCallbacks	253
SIP Control General Type Definitions	255
RvlppSipPhoneCfg	256
RvlppSipControlHandle	263

18 *STUN Module*

What's in this Chapter	265
STUN Functions	266
RvlppStunMgrCreate()	267
RvlppStunAddressResolveComplete()	268

RvIppStunMgrDelete()	269
RvIppStunMgrGetSendMethod()	270
STUN Callbacks	271
RvIppStunIsAddressResolveNeededCB()	272
RvIppStunAddressResolveStartCB()	273
RvIppAddressResolveReplyCB()	274
STUN Type Definitions	275
RvIppStunMgrParam	276
RvIppStunAddrData	277

19 *TLS Module*

What's in this Chapter	279
TLS Functions	280
rvIppSipTlsInitConfig()	281
rvIppSipTlsRegisterExtCbks()	282
TLS Callbacks	283
RvIppTlsGetBufferCB()	284
RvIppSipTlsPostConnectionAssertionCB()	285
TLS Type Definitions	287
RvIppTransportTlsCfg	288
RvIppSipTlsExtCbks	290
RvIppTlsBufferType	291

PART 3: USER CALLBACKS

20 *User Callbacks*

What's in this Chapter	295
User Callbacks	296
RvMdmTermMapDialStringToAddressCB()	297
RvMdmTermMatchDialStringCB()	299
RvMdmTermCreateMediaCB()	300
RvMdmTermDestroyMediaCB()	302

RvMdmTermModifyMediaCB()	303
RvMdmTermPlaySignalCB()	305
RvMdmTermSetStateCB()	307
RvMdmTermStartSignalCB()	308
RvMdmTermStopSignalCB()	310
RvMdmTermMgrConnectCB()	311
RvMdmTermMgrDeleteEphTermCB()	313
RvMdmTermMgrDisconnectCB()	314
RvMdmTermMgrSelectTerminationCB()	316
RvMdmTermModifyMediaCompletedCB()	317

PART 4: EXTENSIBILITY

21 *SIP Control Extension*

What's in this Chapter	321
SIP Control Extension Functions	322
rvlppSipControlGetMdmTerminal()	323
RvlppSipExtCbks	324
SIP Control Extension Type Definitions	326
RvlppSipStackConfigCB()	327
RvlppSipRegisterStackEventsCB()	328
RvlppSipPreCallLegCreatedIncomingCB()	329
RvlppSipPostCallLegCreatedIncomingCB()	331
RvlppSipPreCallLegCreatedOutgoingCB()	332
RvlppSipPostCallLegCreatedOutgoingCB()	334
RvlppSipPreStateChangedCB()	335
RvlppSipPostStateChangedCB()	337
RvlppSipPreMsgToSendCB()	339
RvlppSipPostMsgToSendCB()	341
RvlppSipPreMsgReceivedCB()	343
RvlppSipPostMsgReceivedCB()	345
RvlppSipPreRegClientStateChangedCB()	347
RvlppProviderHandle	349
RvlppTerminalHandle	350

22 *MDM Control Extension*

What's in this Chapter	353
MDM Control Extension Functions	354
rvIppMdmRegisterExtCbks()	355
MDM Control Extension Provider API	356
rvIppMdmProviderGetDialToneDuration()	357
rvIppMdmProviderGetDisplayData()	358
rvIppMdmProviderFindTerminalByTermId()	359
rvIppMdmProviderFindTerminalByAddress()	360
rvIppMdmProviderFindAnyTerminal()	361
rvIppMdmProviderFindTerminalByNumber()	362
MDM Control Extension Terminal API	363
rvIppMdmTerminalGetProvider()	365
rvIppMdmTerminalGetId()	366
rvIppMdmTerminalGetType()	367
rvIppMdmTerminalGetLastEvent()	368
rvIppMdmTerminalGetMediaCaps()	369
rvIppMdmTerminalGetMediaStream()	370
rvIppMdmTerminalIsFirstDigit()	371
rvIppMdmTerminalGetLastDigit()	372
rvIppMdmTerminalGetPhoneNumber()	373
rvIppMdmTerminalGetActiveAudioType()	374
rvIppMdmTerminalGetActiveAudioTerm()	375
rvIppMdmTerminalStopSignals()	376
rvIppMdmTerminalStartUserSignalUI()	377
rvIppMdmTerminalStartUserSignalAT()	378
rvIppMdmTerminalStartDialToneSignal()	379
rvIppMdmTerminalStartRingingSignal()	380
rvIppMdmTerminalStopRingingSignal()	381
rvIppMdmTerminalStartRingbackSignal()	382
rvIppMdmTerminalStartCallWaitingSignal()	383
rvIppMdmTerminalStartCallWaitingCallerSignal()	384
rvIppMdmTerminalStartBusySignal()	385
rvIppMdmTerminalStartWarningSignal()	386

rvlppMdmTerminalStartDTMFTone()	387
rvlppMdmTerminalStopDTMFTone()	388
rvlppMdmTerminalSetHoldInd()	389
rvlppMdmTerminalSetLineInd()	390
rvlppMdmTerminalSendLineActive()	391
rvlppMdmTerminalSetDisplay()	392
rvlppMdmTerminalClearDisplay()	393
rvlppMdmTerminalSendCallerId()	394
rvlppMdmTerminalSetWaitForDigits()	396
rvlppMdmTerminalGetDialString()	397
rvlppMdmTerminalResetDialString()	398
rvlppMdmTerminalIsOutOfBandDtmfEnabled()	399
rvlppMdmTerminalIsDtmfPlayEnabled()	400
rvlppMdmTerminalGetMaxConnections()	401
rvlppMdmTerminalGetNumActiveConnections()	402
rvlppMdmTerminalGetActiveConnection()	403
rvlppMdmTerminalGetCurDisplayRow()	404
rvlppMdmTerminalGetCurDisplayColumn()	405
rvlppMdmTerminalOtherHeldConnExist()	406
rvlppMdmTerminalGetHoldConn()	407
rvlppMdmTerminalSetState()	408
rvlppMdmTerminalGetState()	409
rvlppMdmTerminalGetMdmTerm()	410
rvlppMdmTerminalGetHandle	411
MDM Control Extension Connection API	412
rvlppMdmConnGetTerminal()	413
rvlppMdmConnGetLineId()	414
rvlppMdmConnGetState()	415
rvlppMdmConnGetTermState()	416
rvlppMdmConnGetMediaState()	417
rvlppMdmConnGetType()	418
rvlppMdmConnGetConnectParty()	419
rvlppMdmConnGetLocalMedia()	420
rvlppMdmConnGetMediaCaps()	421
rvlppMdmConnSetUserData()	422
rvlppMdmConnGetUserData()	423
rvlppMdmConnGetCallerName()	424
rvlppMdmConnGetCallerNumber()	425

rvlppMdmConnGetCallerNumberByIndex()	427
rvlppMdmConnGetCallerAddress()	429
rvlppMdmConnGetCallerId()	430
rvlppMdmConnGetRemotePresentationInfo()	431
rvlppMdmConnGetCallState()	432
MDM Control Extension Type Definitions	433
RvlppMdmPreProcessEventCB()	434
RvlppMdmPostProcessEventCB()	436
RvlppMdmConnectionCreatedCB()	437
RvlppMdmConnectionDestructedCB()	438
RvlppMdmMapUserEventCB()	439
RvlppMdmDisplayCB()	441
rvlppMdmExtCbks	443
<i>Index</i>	445

ABOUT THIS MANUAL

This Reference Guide describes the Media Device Manager (MDM) API. The MDM API hides the syntax and semantics of underlying protocol messages and provides an abstract interface to a Media Gateway User Application.

When working with the MDM API, application developers do not need to know specific protocol messages, and therefore the User Application is not aware of these messages. The API interacts with the Media Gateway at a low level, in terms of signals, events and media streams. The MDM resides on top of a layer which processes the protocol commands and breaks the semantics into protocol-independent concepts, such as start/stop signaling and create/modify/delete a stream.

This Reference Guide is divided into the following parts:

- [Part 1: MDM Control](#)
- [Part 2: SIP Control](#)
- [Part 3: User Callbacks](#)
- [Part 4: Extensibility](#)

PART 1: MDM CONTROL

1

RvMdmDigitMap MODULE

WHAT'S IN THIS CHAPTER

This chapter contains functions and type definitions used to operate on RvMdmDigitMap objects.

The RvMdmDigitMap Type holds a dialing plan.

This chapter includes:

- RvMdmDigitMap General Control Functions
- RvMdmDigitMap Accessor Functions
- RvMdmDigitMap Type Definitions

RvMdmDigitMap General Control Functions

RvMDMDIGITMAP GENERAL CONTROL FUNCTIONS

- `rvMdmDigitMapConstruct()`
- `rvMdmDigitMapConstructA()`
- `rvMdmDigitMapCopy()`
- `rvMdmDigitMapDestruct()`

rvMdmDigitMapConstruct()

DESCRIPTION

Constructs a digit map object.

SYNTAX

```
RvMdmDigitMap* rvMdmDigitMapConstruct (  
    IN RvMdmDigitMap* x);
```

PARAMETERS

x

The digit map object.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

rvMdmDigitMapConstructA()

DESCRIPTION

Constructs a digit map object, using a user-specific memory allocator.

SYNTAX

```
RvMdmDigitMap* rvMdmDigitMapConstructA(  
    IN RvMdmDigitMap*    x,  
    IN RvAlloc*           alloc);
```

PARAMETERS

x

The digit map object.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

RvAlloc

rvMdmDigitMapCopy()

DESCRIPTION

Copies the value of one digit map object to another.

SYNTAX

```
RvMdmDigitMap* rvMdmDigitMapCopy(  
    IN RvMdmDigitMap*      d,  
    IN const RvMdmDigitMap* s);
```

PARAMETERS

d

The destination digit map object.

s

The digit map object to copy.

RETURN VALUES

A pointer to the destination object, or NULL if the copy failed.

rvMdmDigitMapDestruct()

DESCRIPTION

Destroys a digit map object.

SYNTAX

```
void rvMdmDigitMapDestruct (  
    IN RvMdmDigitMap* x);
```

PARAMETERS

x

The digit map object.

RETURN VALUES

None.

RvMdmDigitMap ACCESSOR FUNCTIONS

This section includes:

- `rvMdmDigitMapAddPattern()`
- `rvMdmDigitMapSetLongTimeout()`
- `rvMdmDigitMapSetShortTimeout()`
- `rvMdmDigitMapSetStartTimeout()`

RvMdmDigitMap Accessor Functions

rvMdmDigitMapAddPattern()

rvMdmDigitMapAddPattern()

DESCRIPTION

Adds a digit string object to a digit map object.

SYNTAX

```
void rvMdmDigitMapAddPattern(  
    IN RvMdmDigitMap*          x,  
    IN const RvMdmDigitString* s);
```

PARAMETERS

x

The digit map object.

s

The digit string object.

RETURN VALUES

None.

SEE ALSO

[RvMdmDigitString Module](#)

rvMdmDigitMapSetLongTimeout()

DESCRIPTION

Sets the long timeout value of the digit map object.

SYNTAX

```
void rvMdmDigitMapSetLongTimeout (  
    IN RvMdmDigitMap*    x,  
    IN RvUInt32           val) ;
```

PARAMETERS

x

The digit map object.

val

The long timeout value, in seconds.

RETURN VALUES

None.

RvMdmDigitMap Accessor Functions

rvMdmDigitMapSetShortTimeout()

rvMdmDigitMapSetShortTimeout()

DESCRIPTION

Sets the short timeout value of the digit map object.

SYNTAX

```
void rvMdmDigitMapSetShortTimeout (
    IN RvMdmDigitMap*    x,
    IN RvUInt32           val);
```

PARAMETERS

x

The digit map object.

val

The short timeout value, in seconds.

RETURN VALUES

None.

rvMdmDigitMapSetStartTimeout()

DESCRIPTION

Sets the start timeout value of the digit map object.

SYNTAX

```
void rvMdmDigitMapSetStartTimeout(  
    IN RvMdmDigitMap    x,  
    IN RvUInt32 int      val);
```

PARAMETERS

x

The digit map object.

val

The start timeout value, in seconds.

RETURN VALUES

None.

RvMdmDigitMap Type Definitions
rvMdmDigitMapSetStartTimeout()

**RvMDMDIGITMAP
TYPE DEFINITIONS**

- RvMdmDigitMapMatchType

RvMdmDigitMapMatchType

DESCRIPTION

Digit map matching return type. This enumeration is used as a return parameter from [RvMdmTermMatchDialStringCB\(\)](#) callback.

Each enumeration value incurs an action that the Multimedia Terminal Framework will make accordingly:

Value	Description	Action
NOMATCH	The dial string does not match any legal pattern.	1. The Multimedia Terminal Framework stops collecting digits. 2. The Multimedia Terminal Framework starts playing a warning tone.
PARTIALMATCH	The dial string may match a legal pattern.	The Multimedia Terminal Framework will continue collecting digits.
FULLMATCH	The dial string matches a legal pattern, but might be ambiguous (i.e., more digits can be collected).	The Multimedia Terminal Framework will continue collecting digits.
UNAMBIGUOUSMATCH	The dial string matches a legal pattern.	1. The Multimedia Terminal Framework stops collecting digits. 2. The Multimedia Terminal Framework tries to map this dial string to a destination address by invoking <code>RvMdmTermMgrMapDialStringToAddressCB()</code> .

SYNTAX

```
typedef enum  
{
```

RvMdmDigitMap Type Definitions

RvMdmDigitMapMatchType

```
RV_MDMDIGITMAP_NOMATCH,  
RV_MDMDIGITMAP_PARTIALMATCH,  
RV_MDMDIGITMAP_FULLMATCH,  
RV_MDMDIGITMAP_UNAMBIGUOUSMATCH  
} RvMdmDigitMapMatchType;
```

SEE ALSO

[RvMdmTermMatchDialStringCB\(\)](#)

[RvMdmTermMgrMapDialStringToAddressCB\(\)](#)

2

RvMdmDigitPosition MODULE

WHAT'S IN THIS CHAPTER

This chapter contains functions and type definitions used to operate on RvMdmDigitPosition objects.

The RvMdmDigitPosition Type holds the events to be matched at one point in time during dialing.

This chapter includes:

- [RvMdmDigitPosition General Control Functions](#)
- [RvMdmDigitPosition Accessor Functions](#)
- [RvMdmDigitPosition Type Definitions](#)

RvMdmDigitPosition General Control Functions

RvMdmDigitPosition General Control Functions

This section includes:

- `rvMdmDigitPositionConstruct()`
- `rvMdmDigitPositionDestruct()`

rvMdmDigitPositionConstruct()

DESCRIPTION

Constructs a digit position object.

SYNTAX

```
RvMdmDigitPosition* rvMdmDigitPositionConstruct(  
    IN RvMdmDigitPosition* x);
```

PARAMETERS

x

The digit position object.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

rvMdmDigitPositionDestruct()

DESCRIPTION

Destroys a digit position object.

SYNTAX

```
void rvMdmDigitPositionDestruct(  
    IN RvMdmDigitPosition* x);
```

PARAMETERS

x

The digit position object.

RETURN VALUES

None.

RvMdmDigitPosition Accessor Functions

This section includes:

- rvMdmDigitPositionAddEvents()
- rvMdmDigitPositionSetMultipleFlag()
- rvMdmDigitPositionSetTimerMode()

rvMdmDigitPositionAddEvents()

DESCRIPTION

Adds a range of DTMF events to a digit position object.

SYNTAX

```
void rvMdmDigitPositionAddEvents(  
    IN RvMdmDigitPosition*    x,  
    IN RvChar                  c1,  
    IN RvChar                  c2);
```

PARAMETERS

x

The digit position object.

c1

The first event in the range.

c2

The last event in the range.

REMARKS

- To enable an event of a single DTMF digit, set c1=c2.
- c1 and c2 must both be either numbers or letters.

rvMdmDigitPositionSetMultipleFlag()

DESCRIPTION

Sets the multiple flag of the digit position object.

Set this flag to indicate that zero or more repetitions of the digit position are allowed at this point in a digit string.

SYNTAX

```
void rvMdmDigitPositionSetMultipleFlag(  
    IN RvMdmDigitPosition*    x,  
    IN RvBool                  enable);
```

PARAMETERS

x

The digit position object.

enable

The new value of the flag.

RETURN VALUES

None.

RvMdmDigitPosition Accessor Functions

rvMdmDigitPositionSetTimerMode()

rvMdmDigitPositionSetTimerMode()

DESCRIPTION

Sets the timer mode of the digit position object.

Set the timer to be used when matching the events in this position and those after it.

SYNTAX

```
void rvMdmDigitPositionSetTimerMode(  
    IN RvMdmDigitPosition*      x,  
    IN RvMdmDigitPositionTimerMode mode);
```

PARAMETERS

x

The digit position object.

mode

The timer mode.

RETURN VALUES

None.

SEE ALSO

[RvMdmDigitPositionTimerMode](#)

RvMdmDigitPosition TYPE DEFINITIONS

This section includes:

- [RvMdmDigitPositionTimerMode](#)

RvMdmDigitPositionTimerMode

DESCRIPTION

The mode of timer to use for the given digit position in dial string matching.

SYNTAX

```
typedef enum
{
    RV_MDMDIGITPOSITION_NOCHANGE,
    RV_MDMDIGITPOSITION_SHORTTIMER,
    RV_MDMDIGITPOSITION_LONGTIMER
} RvMdmDigitPositionTimerMode;
```

PARAMETERS

RV_MDMDIGITPOSITION_NOCHANGE

Leaves the time mode unchanged, leaving it to be processed in the default manner.

RV_MDMDIGITPOSITION_SHORTTIMER

Uses the short timer that is set for this digit map string. The short timer is set by [rvMdmDigitMapSetShortTimeout\(\)](#).

RV_MDMDIGITPOSITION_LONGTIMER

Uses the long timer that is set for this digit map string. The long timer is set by [rvMdmDigitMapSetLongTimeout\(\)](#).

SEE ALSO

[rvMdmDigitPositionSetTimerMode\(\)](#)

3

RvMdmDigitString MODULE

WHAT'S IN THIS CHAPTER

This chapter contains the functions used to operate on RvMdmDigitString objects.

The RvMdmDigitString Type holds one possible dialing pattern in a dialing plan.

This chapter includes:

- [RvMdmDigitString General Control Functions](#)
- [RvMdmDigitString Accessor Functions](#)

RvMDMDIGITSTRIN G GENERAL CONTROL FUNCTIONS

This section includes:

- `rvMdmDigitStringConstruct()`
- `rvMdmDigitStringDestruct()`

rvMdmDigitStringConstruct()

DESCRIPTION

Constructs a digit string object.

SYNTAX

```
RvMdmDigitString* rvMdmDigitStringConstruct(  
    IN RvMdmDigitString* x);
```

PARAMETERS

x

The digit string object.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

rvMdmDigitStringDestruct()

DESCRIPTION

Destroys a digit string object.

SYNTAX

```
void rvMdmDigitStringDestruct(  
    IN RvMdmDigitString* x);
```

PARAMETERS

x

The digit string object.

RETURN VALUES

None.

RvMdmDigitString G Accessor FUNCTIONS

This section includes:

- `rvMdmDigitStringAddElement()`

rvMdmDigitStringAddElement()

DESCRIPTION

Adds a digit position object to a digit string object.

SYNTAX

```
void rvMdmDigitStringAddElement(  
    IN RvMdmDigitString*      x,  
    IN const RvMdmDigitPosition* pos);
```

PARAMETERS

x

The digit string object.

pos

The digit position object.

RETURN VALUES

None.

SEE ALSO

[RvMdmDigitPosition Module](#)

4

RvMdmEvent MODULE

WHAT'S IN THIS CHAPTER

This chapter contains functions used to operate on RvMdmEvent objects.

This chapter includes:

- [RvMdmEvent Functions](#)

RvMDMEvent FUNCTIONS

This section includes:

- [RvMdmEvent](#)
- [rvMdmEventGetParameterList\(\)](#)
- [rvMdmEventGetName\(\)](#)

RvMdmEvent

DESCRIPTION

An MDM event. The MDM operates through events that are used to handle the various terminations and connections. Each such event has its own name and parameters. Events given by the Multimedia Terminal Framework should be handled as read only objects. The application should not access the fields of this struct directly, but through the relevant access functions.

SYNTAX

```
typedef struct
{
    ...
} RvMdmEvent;
```

SEE ALSO

[rvMdmEventGetParameterList\(\)](#)

[rvMdmEventGetName\(\)](#)

rvMdmEventGetParameterList()

DESCRIPTION

Gets the parameter list of the event object.

SYNTAX

```
const RvMdmParameterList* rvMdmEventGetParameterList(  
    IN const RvMdmEvent *x);
```

PARAMETERS

x

The event object.

RETURN VALUE

The parameter list.

SEE ALSO

[rvMdmEventGetName\(\)](#)

[RvMdmEvent](#)

[RvMdmParameterList Module](#)

rvMdmEventGetName()

DESCRIPTION

Gets the name of the event object.

SYNTAX

```
const RvMdmPackageItem* rvMdmEventGetName (  
    IN const vMdmEvent* x);
```

PARAMETERS

[x](#)

The event object.

RETURN VALUE

The name.

SEE ALSO

[rvMdmEventGetParameterList\(\)](#)

[RvMdmEvent](#)

[RvMdmPackageItem Module](#)

RvMdmEvent Functions

`rvMdmEventGetName()`

5

RvMdmMediaStreamDescr Module

WHAT'S IN THIS CHAPTER

This chapter contains the functions and callback function templates used to operate on `RvMdmMediaStreamDescr` objects.

`RvMdmMediaStreamDescr` functions convey media information.

This chapter includes:

- [RvMdmMediaStreamDescr Accessor Functions](#)

RvMdmMediaStreamDescr Accessor Functions

This section includes:

- `rvMdmMediaStreamDescrGetControlParameters()`
- `rvMdmMediaStreamDescrGetLocalDescr()`
- `rvMdmMediaStreamDescrGetMode()`
- `rvMdmMediaStreamDescrGetRemoteDescr()`
- `rvMdmMediaStreamDescrGetReserveGroupMode()`
- `rvMdmMediaStreamDescrGetReserveValueMode()`
- `rvMdmMediaStreamDescrReportControlParameters()`
- `rvMdmMediaStreamDescrReportLocalDescr()`
- `rvMdmMediaStreamDescrReportRemoteDescr()`

rvMdmMediaStreamDescrGetControlParameters()

DESCRIPTION

Gets the local control parameters for a media stream.

SYNTAX

```
RvMdmParameterList  
rvMdmMediaStreamDescrGetControlParameters (  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

The local control parameters (NULL if absent).

RvMdmMediaStreamDescr Accessor Functions

rvMdmMediaStreamDescrGetLocalDescr()

rvMdmMediaStreamDescrGetLocalDescr()

DESCRIPTION

Gets the local Media Descriptor for a media stream.

SYNTAX

```
RvSdpMsgList rvMdmMediaStreamDescrGetLocalDescr(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

The local Media Descriptor (NULL if absent).

rvMdmMediaStreamDescrGetMode()

DESCRIPTION

Gets the mode of the media stream toward the outside of the context.

SYNTAX

```
RvMdmStreamMode rvMdmMediaStreamDescrGetMode(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

The media stream mode.

rvMdmMediaStreamDescrGetRemoteDescr()

DESCRIPTION

Gets the remote Media Descriptor for a media stream.

SYNTAX

```
RvSdpMsgList rvMdmMediaStreamDescrGetRemoteDescr(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

The remote Media Descriptor (NULL if absent).

rvMdmMediaStreamDescrGetReserveGroupMode()

DESCRIPTION

Gets the reserve group mode of a media stream.

SYNTAX

```
RvBool rvMdmMediaStreamDescrGetReserveGroupMode(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

Returns rvTrue if set. Otherwise, the function returns rvFalse.

rvMdmMediaStreamDescrGetReserveValueMode()

DESCRIPTION

Gets the reserve value mode of a media stream.

SYNTAX

```
RvBool rvMdmMediaStreamDescrGetReserveValueMode(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

Returns rvTrue if set. Otherwise, the function returns rvFalse.

rvMdmMediaStreamDescrReportControlParameters()

DESCRIPTION

Call this function to indicate that the local control parameters obtained by a call to rvMdmMediaStreamDescrGetControlParameters() have been modified by the application selecting from overspecified parameters or setting the value of an underspecified parameter. Calling this function is required to let the MDM know that the updated values of this field must be returned to the MGC.

SYNTAX

```
void rvMdmMediaStreamDescrReportControlParameters(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

None.

rvMdmMediaStreamDescrReportLocalDescr()

DESCRIPTION

Call this function to indicate that the local Media Descriptor obtained by a call to rvMdmMediaStreamDescrGetLocalDescr() has been modified by the application selecting from overspecified parameters or setting the value of an underspecified parameter. Calling this function is required to let the MDM know that the updated values of this field must be returned to the MGC.

SYNTAX

```
void rvMdmMediaStreamDescrReportLocalDescr(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

None.

rvMdmMediaStreamDescrReportRemoteDescr()

DESCRIPTION

Called to indicate that the remote Media Descriptor obtained by a call to rvMdmMediaStreamDescrGetRemoteDescr() has been modified by the application selecting from overspecified parameters or setting the value of an underspecified parameter. Calling this function is required to let the MDM know that the updated values of this field must be returned to the MGC.

SYNTAX

```
void rvMdmMediaStreamDescrReportRemoteDescr(  
    IN RvMdmMediaStreamDescr x);
```

PARAMETERS

x

The Media Descriptor object.

RETURN VALUES

None.

RvMdmMediaStreamDescr Accessor Functions

`rvMdmMediaStreamDescrReportRemoteDescr()`

6

RvMdmPackageItem Module

WHAT'S IN THIS CHAPTER

This chapter contains the functions used to operate on RvMdmPackageItem objects.

The RvMdmPackageItem Type stores a package specific identifier and is used to store properties, events, signals, and statistics.

This chapter includes:

- [RvMdmPackageItem General Control Functions](#)
- [RvMdmPackageItem Accessor Functions](#)

RvMDMPACKAGEITEM GENERAL CONTROL FUNCTIONS

This section includes:

- `rvMdmPackageItemConstruct()`
- `rvMdmPackageItemConstructA()`
- `rvMdmPackageItemConstructCopy()`
- `rvMdmPackageItemCopy()`
- `rvMdmPackageItemDestruct()`
- `rvMdmPackageItemEqual()`

rvMdmPackageItemConstruct()

DESCRIPTION

Constructs a package item object.

SYNTAX

```
RvMdmPackageItem* rvMdmPackageItemConstruct (  
    IN RvMdmPackageItem*    x,  
    IN const RvChar*        pkg,  
    IN const RvChar*        item);
```

PARAMETERS

x

The package item object.

pkg

The name of the package the item belongs to.

item

The name of the item.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

rvMdmPackageItemConstructA()

DESCRIPTION

Constructs a package item object, using a user-specified allocator.

SYNTAX

```
RvMdmPackageItem* rvMdmPackageItemConstructA(  
    IN RvMdmPackageItem*    x,  
    IN const RvChar*        pkg,  
    IN const RvChar*        item,  
    IN RvAlloc*              alloc);
```

PARAMETERS

x

The package item object.

pkg

The name of the package the item belongs to.

item

The name of the item.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

RvAlloc

rvMdmPackageItemConstructCopy()

DESCRIPTION

Constructs a copy of a package item object.

SYNTAX

```
RvMdmPackageItem* rvMdmPackageItemConstructCopy(  
    IN RvMdmPackageItem*      d,  
    IN const RvMdmPackageItem* s,  
    IN RvAlloc*                alloc);
```

PARAMETERS

d

The destination package item object.

s

The package item object to copy.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

RvAlloc

rvMdmPackageItemCopy()

DESCRIPTION

Copies the value of one package item object to another.

SYNTAX

```
RvMdmPackageItem* rvMdmPackageItemCopy(  
    IN RvMdmPackageItem*    d,  
    IN const RvMdmPackageItem* s);
```

PARAMETERS

d

The destination package item object.

s

The package item object to copy.

RETURN VALUES

A pointer to the destination object, or NULL if the copy failed.

rvMdmPackageItemDestruct()

DESCRIPTION

Destroys a package item object.

SYNTAX

```
void rvMdmPackageItemDestruct (  
    IN RvMdmPackageItem* x);
```

PARAMETERS

x

The package item object.

RETURN VALUES

None.

rvMdmPackageItemEqual()

DESCRIPTION

Compares two package item objects for equality.

SYNTAX

```
RvBool rvMdmPackageItemEqual (  
    IN const RvMdmPackageItem*    a,  
    IN const RvMdmPackageItem*    b) ;
```

PARAMETERS

a

The first package item object.

b

The second package item object.

RETURN VALUES

RV_TRUE if the objects are equal; RV_FALSE if not.

RvMDMPACKAGEITEM ACCESSOR FUNCTIONS

This section includes:

- `rvMdmPackageItemGetItem()`
- `rvMdmPackageItemGetPackage()`

RvMdmPackageItem Accessor Functions

rvMdmPackageItemGetItem()

rvMdmPackageItemGetItem()

DESCRIPTION

Gets the name of the package item object.

SYNTAX

```
const RvChar* rvMdmPackageItemGetItem(  
    IN const RvMdmPackageItem* x);
```

PARAMETERS

x

The package item object.

RETURN VALUES

The name of the item.

rvMdmPackageltemGetPackage()

DESCRIPTION

Gets the package name of the package item object.

SYNTAX

```
const RvChar* rvMdmPackageItemGetPackage(  
    IN const RvMdmPackageItem* x);
```

PARAMETERS

x

The package item object.

RETURN VALUES

The package name.

RvMdmPackageItem Accessor Functions

`rvMdmPackageItemGetPackage()`

7

RvMdmParameterList Module

WHAT'S IN THIS CHAPTER

This chapter contains the functions used to operate on RvMdmParameterList objects.

The RvMdmParameterList Type represents a list of parameters.

This chapter includes:

- [RvMdmParameterList General Control Functions](#)
- [RvMdmParameterList Accessor Functions](#)

RvMdmParameterList General Control Functions

This section includes:

- `rvMdmParameterListConstruct()`
- `rvMdmMdmParameterListConstructA()`
- `rvMdmParameterListConstructCopy()`
- `rvMdmParameterListCopy()`
- `rvMdmParameterListDestruct()`

rvMdmParameterListConstruct()

DESCRIPTION

Constructs a parameter list object.

SYNTAX

```
RvMdmParameterList* rvMdmParameterListConstruct (  
    IN RvMdmParameterList* x);
```

PARAMETERS

x

The parameter list object.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

RvMdmParameterList General Control Functions

rvMdmMdmParameterListConstructA()

rvMdmMdmParameterListConstructA()

DESCRIPTION

Constructs a parameter list object, using a user-specific allocator.

SYNTAX

```
RvMdmParameterList* rvMdmParameterListConstructA(  
    IN RvMdmParameterList*    x,  
    IN RvAlloc*                alloc);
```

PARAMETERS

x

The parameter list object.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

RvAlloc

rvMdmParameterListConstructCopy()

DESCRIPTION

Constructs a copy of a parameter list object.

SYNTAX

```
RvMdmParameterList* rvMdmParameterListConstructCopy(  
    IN RvMdmParameterList*    d,  
    IN const RvMdmParameterList* s,  
    IN RvAlloc*                alloc);
```

PARAMETERS

d

The destination parameter list object.

s

The parameter list object to copy.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

RvAlloc

rvMdmParameterListCopy()

DESCRIPTION

Copies the value of one parameter list object to another.

SYNTAX

```
RvMdmParameterList* rvMdmParameterListCopy(  
    IN RvMdmParameterList*    d,  
    IN const RvMdmParameterList* s);
```

PARAMETERS

d

The destination parameter list object.

s

The parameter list object to copy.

RETURN VALUES

A pointer to the destination object, or NULL if the copy failed.

rvMdmParameterListDestruct()

DESCRIPTION

Destroys a parameter list object.

SYNTAX

```
void rvMdmParameterListDestruct(  
    IN RvMdmParameterList* x);
```

PARAMETERS

x

The parameter list object.

RETURN VALUES

None.

RvMDMPARAMETERLIST ACCESSOR FUNCTIONS

This section includes:

- `rvMdmParameterListForEach()`
- `rvMdmParameterListGet()`
- `rvMdmParameterListGet2()`

rvMdmParameterListForEach()

DESCRIPTION

Calls a function for each parameter in a parameter list object.

SYNTAX

```
void rvMdmParameterListForEach(  
    IN const RvMdmParameterList*    x,  
    IN RvMdmParameterFunc           f,  
    IN void*                         data);
```

PARAMETERS

x

The parameter list object.

f

The function.

data

Any user data, to be passed to the function f.

REMARKS

The function passed to rvMdmParameterListForEach() must be of the following type:

```
typedef void (RvMdmParameterFunc) (  
    IN const RvMdmPackageItem*    name,  
    IN const RvMdmParameterValue* value,  
    IN void*                      data);
```

SEE ALSO

[RvMdmPackageItem Module](#)

[RvMdmParameterValue Module](#)

RvMdmParameterList Accessor Functions

rvMdmParameterListGet()

rvMdmParameterListGet()

DESCRIPTION

Gets the value of a parameter in a parameter list object.

SYNTAX

```
const RvMdmParameterValue* rvMdmParameterListGet (
    IN const RvMdmParameterList*    x,
    IN const RvMdmPackageItem*      name);
```

PARAMETERS

x

The parameter list object.

name

The parameter name.

RETURN VALUES

The parameter value, in RvMdmPackageItem format.

SEE ALSO

[RvMdmPackageItem Module](#)

[RvMdmParameterValue Module](#)

[rvMdmParameterListGet2\(\)](#)

rvMdmParameterListGet2()

DESCRIPTION

Gets the value of a parameter in a parameter list object.

SYNTAX

```
const RvMdmParameterValue* rvMdmParameterListGet2(  
    IN const RvMdmParameterList*    x,  
    IN const RvChar*                name);
```

PARAMETERS

x

The parameter list object.

name

The parameter name, as a NULL-terminated string.

RETURN VALUES

The parameter value.

SEE ALSO

[RvMdmParameterValue Module](#)

[rvMdmParameterListGet\(\)](#)

RvMdmParameterList Accessor Functions

rvMdmParameterListIsEmpty()

rvMdmParameterListIsEmpty()

DESCRIPTION

Checks whether the parameter list object is empty.

SYNTAX

```
RvBool rvMdmParameterListIsEmpty(  
    IN const RvMdmParameterList* x);
```

PARAMETERS

x

The parameter list object.

RETURN VALUES

RV_TRUE if the object is empty; RV_FALSE if not.

rvMdmParameterListSet()

DESCRIPTION

Sets a parameter in a parameter list object.

SYNTAX

```
void rvMdmParameterListSet (  
    IN RvMdmParameterList*      x,  
    IN const RvMdmPackageItem*   name,  
    IN const RvMdmParameterValue* value);
```

PARAMETERS

x

The parameter list object.

name

The parameter name.

value

The parameter value.

RETURN VALUES

None.

SEE ALSO

[RvMdmPackageItem Module](#)

[RvMdmParameterValue Module](#)

RvMdmParameterList Accessor Functions

`rvMdmParameterListSet()`

8

RvMdmParameterValue Module

WHAT'S IN THIS CHAPTER

This chapter contains the functions used to operate on RvMdmParameterValue objects.

The RvMdmParameterValue Type stores the value of a parameter and supports fully specified, over-specified, and under-specified parameters

This chapter includes:

- [RvMdmParameterValue General Control Functions](#)
- [RvMdmParameterValue Accessor Functions](#)

RvMdmParameter Value General Control Functions

This section includes:

- `rvMdmParameterValueConstruct()`
- `rvMdmParameterValueConstructA()`
- `rvMdmParameterValueConstructCopy()`
- `rvMdmParameterValueConstructList()`
- `rvMdmParameterValueConstructListA()`
- `rvMdmParameterValueDestruct()`

rvMdmParameterValueConstruct()

DESCRIPTION

Constructs a parameter value object.

SYNTAX

```
rvMdmParameterValue* rvMdmParameterValueConstruct(  
    IN rvMdmParameterValue*    x,  
    IN const RvChar*            val);
```

PARAMETERS

x

The parameter value object.

val

The parameter value.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

REMARKS

For under-specified and over-specified parameters use one of the special RvMdmParameterValue constructors instead.

rvMdmParameterValueConstructA()

DESCRIPTION

Constructs a parameter value object, using a user-specific allocator.

SYNTAX

```
rvMdmParameterValue* rvMdmParameterValueConstructA(  
    IN rvMdmParameterValue*    x,  
    IN const RvChar*           val,  
    IN RvAlloc*                 alloc);
```

PARAMETERS

x

The parameter value object.

val

The parameter value.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

REMARKS

For under-specified and over-specified parameters use one of the special RvMdmParameterValue constructors instead.

SEE ALSO

RvAlloc

rvMdmParameterValueConstructCopy()

DESCRIPTION

Constructs a copy of a parameter value object.

SYNTAX

```
RvMdmParameterValue* rvMdmParameterValueConstructCopy(  
    IN RvMdmParameterValue*    d,  
    IN const RvMdmParameterValue* s,  
    IN RvAlloc*                  alloc);
```

PARAMETERS

d

The destination parameter value object.

s

The parameter value object to copy.

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

RvAlloc

rvMdmParameterValueConstructList()

DESCRIPTION

Constructs a parameter value object that represents a list of possible values.

SYNTAX

```
RvMdmParameterValue* rvMdmParameterValueConstructList (  
    IN RvMdmParameterValue*    x,  
    IN RvMdmRelation            type) ;
```

PARAMETERS

[x](#)

The parameter value object.

[type](#)

The type of list, either "AND" or "OR".

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

[RvMdmRelation](#)

rvMdmParameterValueConstructListA()

DESCRIPTION

Constructs a parameter value object, using a user-specific allocator.

SYNTAX

```
RvMdmParameterValue* rvMdmParameterValueConstructListA(  
    IN RvMdmParameterValue*    x,  
    IN RvMdmRelation            type,  
    IN RvAlloc*                 alloc);
```

PARAMETERS

x

The parameter value object.

type

The type of list, either "AND" or "OR".

alloc

The allocator to use.

RETURN VALUES

A pointer to the constructed object, or NULL if construction failed.

SEE ALSO

[RvMdmRelation](#)

[RvAlloc](#)

rvMdmParameterValueDestruct()

DESCRIPTION

Destroys a parameter value object.

SYNTAX

```
void rvMdmParameterValueDestruct(  
    IN RvMdmParameterValue* x);
```

PARAMETERS

x

The parameter value object.

RETURN VALUES

None.

RvMdmParameterValue Accessor Functions

This section includes:

- `rvMdmParameterValueAddToList()`
- `rvMdmParameterValueGetListSize()`
- `rvMdmParameterValueGetListValue()`
- `rvMdmParameterValueGetType()`
- `rvMdmParameterValueGetValue()`

RvMdmParameterValue Accessor Functions

rvMdmParameterValueAddToList()

rvMdmParameterValueAddToList()

DESCRIPTION

Adds a value to a list of possible values.

SYNTAX

```
void rvMdmParameterValueAddToList(  
    IN RvMdmParameterValue*    x,  
    IN const RvChar*           val);
```

PARAMETERS

x

The parameter value object.

val

The value to be added.

RETURN VALUES

None.

rvMdmParameterValueGetListSize()

DESCRIPTION

Gets the number of possible values for a parameter value object that is a list.

SYNTAX

```
RvSize_t rvMdmParameterValueGetListSize(  
    IN const RvMdmParameterValue* x);
```

PARAMETERS

x

The parameter value object.

RETURN VALUES

The list size.

RvMdmParameterValue Accessor Functions

rvMdmParameterValueGetListValue()

rvMdmParameterValueGetListValue()

DESCRIPTION

Gets one item from a parameter value object that is a list.

SYNTAX

```
const RvChar* rvMdmParameterValueGetListValue(  
    IN const RvMdmParameterValue*    x,  
    IN RvSize_t                        index);
```

PARAMETERS

x

The parameter value object.

index

The index of the item.

RETURN VALUES

The value.

rvMdmParameterValueGetType()

DESCRIPTION

Gets the type of the parameter value object.

SYNTAX

```
RvMdmRelation rvMdmParameterValueGetType(  
    IN const RvMdmParameterValue* x);
```

PARAMETERS

[x](#)

The parameter value object.

RETURN VALUES

The type of the parameter value.

SEE ALSO

[RvMdmRelation](#)

RvMdmParameterValue Accessor Functions

rvMdmParameterValueGetValue()

rvMdmParameterValueGetValue()

DESCRIPTION

Gets the value of the parameter value object.

SYNTAX

```
const RvChar* rvMdmParameterValueGetValue(  
    IN const RvMdmParameterValue* x);
```

PARAMETERS

x

The parameter value object.

RETURN VALUES

The value.

9

RvMdmSignal Module

WHAT'S IN THIS CHAPTER

This chapter contains the functions used to operate on RvMdmSignal objects. A signal is an event that is applied on a termination. A signal contains an Id, with associated parameters. In addition each signal is related to a specific package. The callbacks that indicate this type of object are:

- [RvMdmTermPlaySignalCB\(\)](#)
- [RvMdmTermStartSignalCB\(\)](#)
- [RvMdmTermStopSignalCB\(\)](#)

This chapter includes:

- [RvMdmSignal Accessor Functions](#)
- [RvMdmSignal List of Packages and Events](#)

RvMDMSIGNAL ACCESSOR FUNCTIONS

This section includes:

- `rvMdmSignalGetId()`
- `rvMdmSignalGetPkg()`
- `rvMdmSignalGetArguments()`

rvMdmSignalGetId()

DESCRIPTION

Gets the signal Id.

SYNTAX

```
const RvChar* rvMdmSignalGetId(  
    IN const RvMdmSignal* signal);
```

PARAMETERS

[signal](#)

The signal object.

RETURN VALUES

The signal Id.

REMARKS

Each package has its own set of signals. For more information, see [RvMdmSignal List of Packages and Events](#).

rvMdmSignalGetPkg()

DESCRIPTION

Gets the package to which the signal belongs.

SYNTAX

```
const RvChar* rvMdmSignalGetPkg(  
    IN const RvMdmSignal* signal);
```

PARAMETERS

signal

The signal object.

RETURN VALUES

The signal package.

rvMdmSignalGetArguments()

DESCRIPTION

Gets the signal arguments.

SYNTAX

```
const RvMdmParameterList rvMdmSignalGetArguments(  
    IN const RvMdmSignal signal);
```

PARAMETERS

[signal](#)

The signal object.

RETURN VALUES

The arguments for the signal. Returns an empty list if there are no arguments.

SEE ALSO

[RvMdmParameterList Module](#)

RvMdmSignal List of Packages and Events

rvMdmSignalGetArguments()

RVMDMSIGNAL LIST OF PACKAGES AND EVENTS

This section includes:

- DISPLAY PACKAGE (dis)
- KEY PACKAGE (kp)
- FUNCTION KEY PACKAGE (kf)
- INDICATOR PACKAGE (ind)
- BASIC DTMF GENERATOR PACKAGE (dg)
- DTMF DETECTION PACKAGE (dd)
- CALL PROGRESS TONES GENERATOR PACKAGE (cg)
- ANALOG LINE SUPERVISION PACKAGE (al)
- RADVISION CALL CONTROL PACKAGE (rvcc)

DISPLAY PACKAGE (dis)

This package defines properties and signals associated with text display in the user interface elements. This package is used for the UI termination.

Signal ID	Description	Parameter	Parameter Values
di	Display Text	Row	Integer value
		Column	Integer value
		String	Text
cld	Clear display and reset cursor to 0,0	None	None

KEY PACKAGE (kp)

This package defines the basic key User Interface elements. Specific key IDs are selected by name (keyid) from the list of keys. The keypad package is used to represent a standard 10-digit keypad and the '*', '#', A, B, C, and D keys. This package is used to control the UI termination.

Event ID	Description	Parameter	Parameter Values
kd	Key Down: A key was pressed	Keyid	k0, k1, k2,...(see below)
ku	Key Up	Keyid	k0, k1, k2,...(see below)
		duration (Key press duration)	Duration in milliseconds
ce	DigitMap Completion Event (used to indicate that digit connection completed, see RvMdmDigitMap Module)	Ds (digit string)	The collected digits (as a null terminated string)
		Meth (method)	"UM"—Unambiguous match. "PM"—Partial match, completion by timer expiry or unmatched event. "FM"—Full match, completion by timer expiry or unmatched event.

Keyid	Description
k0	Keypad digit 0
k1	Keypad digit 1
k2	Keypad digit 2
k3	Keypad digit 3
k4	Keypad digit 4
k5	Keypad digit 5
k6	Keypad digit 6
k7	Keypad digit 7
k8	Keypad digit 8
k9	Keypad digit 9
ks	Keypad digit *
ko	Keypad digit #
kA	Keypad digit A
kB	Keypad digit B
kC	Keypad digit C
kD	Keypad digit D

FUNCTION KEY PACKAGE (kf)

This package adds additional key IDs that are used for common telephone function keys. This package is used for the UI termination.

Event ID	Description	Parameter	Parameter Values
kd	Key Down: A key was pressed	keyid	kh (Hookswitch) kl (Hold) kc (Conference) kt (Transfer) kbt (Blind Transfer) cfwu (Call Forward unconditional) cfwb (Call Forward busy) cfur (Call Forward no reply) mu (Mute) hf (Handsfree) ht (Headset) 1001 - 1999 (Set of line keys)
ku	Key Up	keyid	Same as above.
		duration (Key press duration)	Duration in milliseconds

Note The key sense of hookswitch (keyid=kh) is a special case. Key up indicates the hookswitch is depressed (for example, Handset is on-hook). Key down indicates the hookswitch is lifted (for example, Handset is off-hook).

INDICATOR PACKAGE (ind)

This package defines the basic behavior of indicator User Interface elements. Specific indicators are addressed by name (indId) from the list of indicators. The indicator will usually be mapped to a LED in the phone. The ir indicator addresses both the LED and the audible ring. This package is used for the UI termination.

Signal ID	Description	Parameter	Parameter Values
is	Set indicator state	indid	il (Hold indicator) ic (Conference indicator) 1001-1999 (line indicators) ir (Ringer/Alerter indicator) im (Message waiting)
		state	on, off, blink, fast_blink, slow_blink
			distRing (distinctive ringing)— A character string specifying an alternative ringing content.

BASIC PACKAGES

BASIC DTMF GENERATOR PACKAGE (dg)

This package defines the basic DTMF tones as signals. This package is used to apply DTMF tones to a Handset or Analog termination.

Signal ID	Description	Parameter	Parameter Values
d0	DTMF tone 0	None	None
d1	Dtmf tone 1	None	None
d1	DTMF tone 1	None	None
d2	DTMF tone 2	None	None
d3	DTMF tone 3	None	None
d4	DTMF tone 4	None	None
d5	DTMF tone 5	None	None
d6	DTMF tone 6	None	None
d7	DTMF tone 7	None	None
d8	DTMF tone 8	None	None
d9	DTMF tone 9	None	None
ds	DTMF tone *	None	None
do	DTMF tone #	None	None
da	DTMF tone A	None	None
db	DTMF tone B	None	None
dc	DTMF tone C	None	None
dd	DTMF tone D	None	None

DTMF DETECTION PACKAGE (dd)

This package defines the basic DTMF events, which are transmitted when the user presses a digit key. This package is used to report DTMF events on an Analog termination.

Event ID	Description	Parameter	Parameter Values
d0	DTMF tone 0	None	None
d1	Dtmf tone 1	None	None
d1	DTMF tone 1	None	None
d2	DTMF tone 2	None	None
d3	DTMF tone 3	None	None
d4	DTMF tone 4	None	None
d5	DTMF tone 5	None	None
d6	DTMF tone 6	None	None
d7	DTMF tone 7	None	None
d8	DTMF tone 8	None	None
d9	DTMF tone 9	None	None
ds	DTMF tone *	None	None
do	DTMF tone #	None	None
da	DTMF tone A	None	None

RvMdmSignal List of Packages and Events
DTMF DETECTION PACKAGE (dd)

Event ID	Description	Parameter	Parameter Values
db	DTMF tone B	None	None
dc	DTMF tone C	None	None
dd	DTMF tone D	None	None
ce	DigitMap Completion Event (used to indicate that digit collection is completed, see RvMdmDigitMap Module)	ds (digit string)	The collected digits (as a null terminated string)
		Meth (method)	"UM" Unambiguous match. "PM" Partial match, completion by timer expiry or unmatched event. "FM" Full match, completion by timer expiry or unmatched event.

CALL PROGRESS TONES GENERATOR PACKAGE (cg)

This package defines the basic call progress tones as signals. This package is used to apply tones to the Handset or Analog Line terminations.

Signal ID	Description	Parameter	Parameter Values
dt	Dial tone	None	None
rt	Ringing tone (the other side is ringing)	distRing (distinctive ringing)	A character string specifying an alternative ringback content.
bt	Busy tone	None	None
ct	Congestion tone	None	None
wt	Warning tone	None	None
cw	Call Waiting tone	None	None
cr	Caller Waiting tone	None	None

ANALOG LINE SUPERVISION PACKAGE (al)

This package defines events and signals for an analog line. This package is used to control an Analog termination.

Event ID	Description	Parameter	Parameter Values
on	Detects Handset going on-hook	None	None
of	Detects Handset going off-hook	None	None
fl	Detects Handset flash	None	None

Signal ID	Description	Parameter	Parameter Values
ri	Applies ringing on the line	None	None

RADVISION CALL CONTROL PACKAGE (rvcc)

This package is used to add special signals and events, and to add signals and events that are needed but are not present in existing packages.

Event ID	Description	Parameter	Parameter Values
ga	Gateway active. Sent when all terminations are registered with the MDM. The Call Control will initialize the Terminal Framework display.	None	None
reject	Reject an incoming call on a given line without answering.	keyid	1001-1999 (the line where the call is rejected).

Signal ID	Description	Parameter	Parameter Values
la	Line active. Indicate that this line has a call on it.	LineId	1001-1999
li	Line inactive. This line has become inactive.	LineId	1001-1999
callerId	Caller ID parameters. This signal is used when a call is received in the analog line, so that the user can format the information.	name	text (the caller name)
		number	text (the caller number)
		address	text (the caller IP address)

Note All events and signals in this package are used for the UI termination, except for callerId, which is used for the Analog termination.

RvMdmSignal List of Packages and Events
RADVISION CALL CONTROL PACKAGE (rvcc)

10

RvMdmTerm MODULE

WHAT'S IN THIS CHAPTER

This chapter contains the functions and callback function templates used to operate on RvMdmTermMgr objects.

RvMdmTerm functions provide access to data related to a Termination registered with the RvMdmTermMgr. They also provide a way to report events associated with the Termination.

This chapter includes:

- [RvMdmTerm Functions](#)

RVMMDMTERM FUNCTIONS

This section includes:

- `rvMdmTermGetId()`
- `rvMdmTermGetType()`
- `rvMdmTermGetUserData()`
- `rvMdmTermProcessEvent()`
- `rvMdmTermSetUserData()`
- `rvMdmTermPropertiesSetPresentationInfo()`
- `rvMdmTermPropertiesSetPhoneNumbers()`
- `rvMtfTerminalMakeCall()`
- `rvMdmTermModifyMedia()`

rvMdmTermGetId()

DESCRIPTION

Gets the Termination Id.

SYNTAX

```
const RvChar* rvMdmTermGetId(  
    IN RvMdmTerm* term);
```

PARAMETERS

term

A pointer to the Termination object.

RETURN VALUES

The Termination Id.

REMARKS

For a temporary Termination inside the RvMdmTermMgrSelectTerminationCB() callback (type RV_MDMTERMTYPE_UNKNOWN), this function will return a partial Id or an empty string.

rvMdmTermGetType()

DESCRIPTION

Gets the Termination type.

SYNTAX

```
RvMdmTermType rvMdmTermGetType(  
    IN RvMdmTerm* term);
```

PARAMETERS

[term](#)

A pointer to the Termination object.

RETURN VALUES

The Termination type (ephemeral, physical, undefined).

SEE ALSO

[RvMdmTermType](#)

rvMdmTermGetUserData()

DESCRIPTION

Returns the user data associated with the Termination.

SYNTAX

```
void* rvMdmTermGetUserData(  
    IN RvMdmTerm* term);
```

PARAMETERS

[term](#)

A pointer to the Termination object.

RETURN VALUES

The Termination user data, or NULL if not set.

SEE ALSO

[rvMdmTermSetUserData\(\)](#)

rvMdmTermProcessEvent()

DESCRIPTION

Reports an event detected in the Termination.

SYNTAX

```
void rvMdmTermProcessEvent (
    IN RvMdmTerm*          term,
    IN const RvChar*       pkg,
    IN const RvChar*       id,
    IN RvMdmMediaStream*   media,
    IN RvMdmParameterList* args);
```

PARAMETERS

term

A pointer to the Termination object.

pkg

The package Id.

id

The event Id.

media

The user Id of the media stream where the event was detected, or NULL if the event is not related to any media stream.

args

The event parameters or NULL.

RETURN VALUES

None.

SEE ALSO

[RvMdmSignal List of Packages and Events](#)

[RvMdmParameterList Module](#)

rvMdmTermSetUserData()

DESCRIPTION

Sets the user data associated with the Termination.

SYNTAX

```
void rvMdmTermSetUserData (
    IN RvMdmTerm*    term,
    IN void*          data);
```

PARAMETERS

[term](#)

A pointer to the Termination object.

[data](#)

The user data.

RETURN VALUES

None.

REMARKS

Use this function to associate application data and memory with the Termination. The application may then avoid the need to have its own database of Terminations.

SEE ALSO

[rvMdmTermGetUserData\(\)](#)

rvMdmTermPropertiesSetPresentationInfo()

DESCRIPTION

Sets the configuration of the presentation name and permission to present it in any outgoing messages.

SYNTAX

```
RvBool rvMdmTermPropertiesSetPresentationInfo(  
    IN RvMdmTermDefaultProperties*    x,  
    IN RvMdmTermPresentationInfo*    presentationInfo);
```

PARAMETERS

x

The default properties object.

presentationInfo

The presentation configuration to use.

RETURN VALUES

RV_TRUE on success, RV_FALSE on failure.

SEE ALSO

RvMdmTermPresentationInfo

RvMdmTerm Functions

rvMdmTermPropertiesSetPhoneNumbers()

rvMdmTermPropertiesSetPhoneNumbers()

DESCRIPTION

Sets the list of phone numbers in the properties of a Termination.

SYNTAX

```
void rvMdmTermPropertiesSetPhoneNumbers (  
    IN RvMdmTermDefaultProperties*    x,  
    IN RvMdmTermPhoneNumbers*        phoneNumbers) ;
```

PARAMETERS

x

The default properties object.

phoneNumbers

Phone numbers to set.

RETURN VALUES

None.

SEE ALSO

RvMdmTermPhoneNumbers Module

rvMtfTerminalMakeCall()

DESCRIPTION

Initiates an outgoing call to a destination address on the specified line and Termination. This function skips the phase of collecting the digits and then translating them into an address.

SYNTAX

```
RvStatus rvMtfTerminalMakeCall(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN const RvChar*          address);
```

PARAMETERS

terminalHndl

Terminal handle to dial on.

address

Destination address to dial to.

If this address is given as a string of DTMF digits only, then the Multimedia Terminal Framework will initiate the digit-to-address translation. Otherwise, the Multimedia Terminal Framework will treat this strong as the exact address to dial to in the specific protocol used.

RETURN VALUES

RV_OK on success, other on failure.

rvMdmTermModifyMedia()

DESCRIPTION

This function enables the user application to modify media parameters of an existing media stream during a connected call.

The function should be called during a connected call. It affects the active call (for example, if one call is On Hold and second is connected, it will affect the connected call). If this function is called when the terminal is in Idle state (no connected calls), the function will be ignored.

Calling this function causes the RV_CCTERMEVENT_MODIFYMEDIA event to be sent to the Multimedia Terminal Framework; hence, the process is asynchronous.

After calling this function two callback functions will be called:

- [RvMdmTermModifyMediaCB\(\)](#)—this callback is one of the media callbacks that are implemented for basic calls (see the Multimedia Terminal Framework *Programmer Guide*, Building Your Application chapter, Integrating Media section). This callback is called to modify media parameters in the device.
- [RvMdmTermModifyMediaCompletedCB\(\)](#)—this callback is called to indicate the end of the process.

For more information about this function, see the Multimedia Terminal Framework *Programmer Guide*, Appendix E, Dynamic Modify Media section.

SYNTAX

```
RvBool rvMdmTermModifyMedia(  
    IN RvMdmTerm*    term,  
    IN RvSdpMsg*     sdpMsg);
```

PARAMETERS

[term](#)

A pointer to terminal.

[sdpMsg](#)

A pointer to the SDP message that contains the new media parameters.

RETURN VALUES

This function currently always returns True.

RvMdmTerm Functions
rvMdmTermModifyMedia()

11

RvMdmTermClass Module

WHAT'S IN THIS CHAPTER

This chapter contains the functions and callback function templates used to operate on `RvMdmTermClass` objects.

`RvMdmTermClass` functions are used to define Termination classes. All the Terminations from the same class will share callback functions and capabilities. More than one class can be defined to support different types of Terminations with different capabilities and different callback function implementations.

This chapter includes:

- [RvMdmTermClass Functions](#)

RVMDMTERMC CLASS FUNCTIONS

This section includes:

- `rvMdmTermClassAddMediaCapabilities()`
- `rvMdmTermClassAddSupportedPkg()`
- `rvMdmTermClassRegisterCreateMediaCB()`
- `rvMdmTermClassRegisterDestroyMediaCB()`
- `rvMdmTermClassRegisterMapDialStringToAddressCB()`
- `rvMdmTermClassRegisterMatchDialStringCB()`
- `rvMdmTermClassRegisterModifyMediaCB()`
- `rvMdmTermClassRegisterPlaySignalCB()`
- `rvMdmTermClassRegisterStartSignalCB()`
- `rvMdmTermClassRegisterStopSignalCB()`
- `rvMdmTermClassRegisterMapDialStringToAddressCB()`
- `rvMdmTermClassRegisterMatchDialStringCB()`
- `rvMdmTermClassClearMediaCapabilities()`
- `rvMdmTermClassRegisterRegisterPhysTermCompletedCB()`
- `rvMdmTermClassRegisterUnregisterTermCompletedCB()`
- `rvMdmTermClassRegisterModifyMediaCompletedCB()`

rvMdmTermClassAddMediaCapabilities()

DESCRIPTION

Adds media capabilities to a Termination class.

Call this function once for every type of stream with different capabilities that the class supports. For example, call this function once for Terminations supporting one stream, such as audio, and a second time for Terminations supporting two types of streams, such as audio and video.

SYNTAX

```
void rvMdmTermClassAddMediaCapabilities (
    IN RvMdmTermClass*          c,
    IN const RvSdpMsgList*      localDescr,
    IN const RvSdpMsgList*      remoteDescr,
    IN const RvMdmParameterList* localProperties);
```

PARAMETERS

c

A pointer to the Termination class.

localDescr

A list of supported media values for the local media.

remoteDescr

A list of supported media values for the remote media.

localProperties

A list of supported local control properties for this stream.

RETURN VALUES

None.

SEE ALSO

[RvMdmParameterList Module](#)

RvMdmTermClass Functions

`rvMdmTermClassAddMediaCapabilities()`

`rvMdmTermClassClearMediaCapabilities()`

rvMdmTermClassAddSupportedPkg()

DESCRIPTION

Use this function to set the list of packages supported by a Termination. This function must be called for every package that a Termination of this class supports.

SYNTAX

```
void rvMdmTermClassAddSupportedPkg(  
    IN RvMdmTermClass*    c,  
    IN const RvChar*      name,  
    IN RvUInt              version);
```

PARAMETERS

c

A pointer to the Termination class.

name

The name of a supported package that is previously registered with the TermMgr.

version

The package id.

RETURN VALUES

None.

RvMdmTermClass Functions

`rvMdmTermClassRegisterCreateMediaCB()`

rvMdmTermClassRegisterCreateMediaCB()

DESCRIPTION

Sets the callback function to call when creating a media stream in a Termination of this class.

SYNTAX

```
void rvMdmTermClassRegisterCreateMediaCB(  
    IN RvMdmTermClass*      c,  
    IN RvMdmTermCreateMediaCB createMediaF);
```

PARAMETERS

c

A pointer to the Termination class object.

createMediaF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermCreateMediaCB\(\)](#)

rvMdmTermClassRegisterDestroyMediaCB()

DESCRIPTION

Sets the callback function to call when deleting a media stream in a Termination of this class.

SYNTAX

```
void rvMdmTermClassRegisterDestroyMediaCB(  
    IN RvMdmTermClass*      c,  
    IN RvMdmTermDestroyMediaCB destroyMediaF);
```

PARAMETERS

c

A pointer to the Termination class object.

destroyMediaF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermDestroyMediaCB\(\)](#)

RvMdmTermClass Functions

rvMdmTermClassRegisterModifyMediaCB()

rvMdmTermClassRegisterModifyMediaCB()

DESCRIPTION

Sets the callback to call when modifying an existing media stream in a Termination of this class.

SYNTAX

```
void rvMdmTermClassRegisterModifyMediaCB(  
    IN RvMdmTermClass*      c,  
    IN RvMdmTermModifyMediaCB modifyMediaF);
```

PARAMETERS

c

A pointer to the Termination class object.

modifyMediaF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermModifyMediaCB\(\)](#)

rvMdmTermClassRegisterPlaySignalCB()

DESCRIPTION

Sets the callback function that is used to play a signal in a Termination of this class.

This signal is played to completion by the application and not stopped by the Termination Manager.

SYNTAX

```
void rvMdmTermClassRegisterPlaySignalCB(  
    IN RvMdmTermClass*      c,  
    IN RvMdmTermPlaySignalCB playSignalF);
```

PARAMETERS

c

A pointer to the Termination class object.

playSignalF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermPlaySignalCB\(\)](#)

rvMdmTermClassRegisterStartSignalCB()

DESCRIPTION

Registers the callback function [RvMdmTermStartSignalCB\(\)](#) that is used to start a signal in a Termination of this class. This signal is played by the application until explicitly stopped by the Termination Manager, which calls the callback function [RvMdmTermStopSignalCB\(\)](#).

SYNTAX

```
void rvMdmTermClassRegisterStartSignalCB(  
    IN RvMdmTermClass      c,  
    IN RvMdmTermStartSignalCB startSignalF);
```

PARAMETERS

c

A pointer to the Termination class object.

startSignalF

The callback function.

RETURN VALUES

None.

REMARKS

The callback function [RvMdmTermStartSignalCB\(\)](#) applies to signals of the following types: On/Off (OO), and TimeOut (TO).

SEE ALSO

[RvMdmTermStartSignalCB\(\)](#)

rvMdmTermClassRegisterStopSignalCB()

DESCRIPTION

Registers the callback function [RvMdmTermStopSignalCB\(\)](#) that is used to stop a signal in a Termination of this class.

SYNTAX

```
void rvMdmTermClassRegisterStopSignalCB(  
    IN RvMdmTermClass*      c,  
    IN RvMdmTermStopSignalCB stopSignalF);
```

PARAMETERS

c

A pointer to the Termination class object.

stopSignalF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermStopSignalCB\(\)](#)

[RvMdmTermStartSignalCB\(\)](#)

RvMdmTermClass Functions

`rvMdmTermClassRegisterMapDialStringToAddressCB()`

rvMdmTermClassRegisterMapDialStringToAddressCB()

DESCRIPTION

Registers the user callback function

[RvMdmTermMapDialStringToAddressCB\(\)](#).

SYNTAX

```
void rvMdmTermClassRegisterMapDialStringToAddressCB (
    IN RvMdmTermClass*          c,
    IN
    RvMdmTermMapDialStringToAddressCB mapDialStringToAddressF);
```

PARAMETERS

c

A pointer to the Termination class object.

mapDialStringToAddressF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermMapDialStringToAddressCB\(\)](#)

rvMdmTermClassRegisterMatchDialStringCB()

DESCRIPTION

Installs the callback [RvMdmTermMatchDialStringCB\(\)](#).

SYNTAX

```
void rvMdmTermClassRegisterMatchDialStringCB(  
    IN RvMdmTermClass*          c,  
    IN RvMdmTermMatchDialStringCB matchDialStringF)
```

PARAMETERS

c

The Termination class for which this callback is called. The same callback will be called for all Terminations of this class.

matchDialStringF

The callback used to indicate when dialing is completed.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermMatchDialStringCB\(\)](#)

RvMdmTermClass Functions

`rvMdmTermClassClearMediaCapabilities()`

`rvMdmTermClassClearMediaCapabilities()`

DESCRIPTION

Clears all current media capabilities from a Termination class.

SYNTAX

```
void rvMdmTermClassClearMediaCapabilities(  
    IN RvMdmTermClass* c);
```

PARAMETERS

c

A pointer to the Termination class.

RETURN VALUES

None.

SEE ALSO

[`rvMdmTermClassAddMediaCapabilities\(\)`](#)

rvMdmTermClassRegisterRegisterPhysTermCompletedCB()

DESCRIPTION

Sets the callback to be called when a physical termination is registered, if called asynchronously.

SYNTAX

```
void rvMdmTermClassRegisterRegisterPhysTermCompletedCB(  
    IN RvMdmTermClass* C,  
    IN RvMdmTermRegisterPhysTermCompletedCB registerPhysTermC  
ompletedF);
```

PARAMETERS

C

A pointer to the termination class object.

registerPhysTermCompletedF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermRegisterPhysTermCompletedCB\(\)](#)

RvMdmTermClass Functions

rvMdmTermClassRegisterUnregisterTermCompletedCB()

rvMdmTermClassRegisterUnregisterTermCompletedCB()

DESCRIPTION

Sets the callback to be called when a physical termination is unregistered, if called asynchronously.

SYNTAX

```
void rvMdmTermClassRegisterUnregisterTermCompletedCB(  
    IN RvMdmTermClass*          c,  
    IN RvMdmTermUnregisterTermCompletedCB  unregisterTermCom  
pletedF) ;
```

PARAMETERS

c

A pointer to the termination class object.

unregisterPhysTermCompletedF

The callback function.

RETURN VALUES

None.

SEE ALSO

RvMdmTermUnregisterTermCompletedCB()

rvMdmTermClassRegisterModifyMediaCompletedCB()

DESCRIPTION

This function is called during initialization to register implementation for the callback [RvMdmTermModifyMediaCompletedCB\(\)](#).

SYNTAX

```
void rvMdmTermClassRegisterModifyMediaCompletedCB(  
    IN RvMdmTermClass*                c,  
    IN RvMdmTermModifyMediaCompletedCB registerMediaModifiedF);
```

PARAMETERS

c

Pointer to the RTP class (allocated during the initialization process).

registerMediaModifiedF

Pointer to the callback implementation.

RETURN VALUES

None.

RvMdmTermClass Functions

`rvMdmTermClassRegisterModifyMediaCompletedCB()`

12

RvMdmTermDefaultProperties MODULE

WHAT'S IN THIS CHAPTER

This chapter contains the functions and used to operate on RvMdmTermDefaultProperties objects.

When registering a Termination, the RvMdmTermDefaultProperties object is required if the Termination has media-independent properties that can be set or audited. An example of these properties is *nrows* (number of rows) for terminations supporting the dis (Display) package.

This chapter includes:

- [RvMdmTermDefaultProperties General Control Functions](#)
- [RvMdmTermDefaultProperties Accessor Functions](#)

RvMdmTermDefaultPropertiesGeneralControlFunctions

This section includes:

- `rvMdmTermDefaultPropertiesConstruct()`
- `rvMdmTermDefaultPropertiesDestruct()`

rvMdmTermDefaultPropertiesConstruct()

DESCRIPTION

Constructs the object.

SYNTAX

```
void rvMdmTermDefaultPropertiesConstruct(  
    IN RvMdmTermDefaultProperties* x);
```

PARAMETERS

x

The default properties object.

RETURN VALUES

None.

rvMdmTermDefaultPropertiesDestruct()

DESCRIPTION

Constructs the object.

SYNTAX

```
void rvMdmTermDefaultPropertiesDestruct (  
    IN RvMdmTermDefaultProperties* x);
```

PARAMETERS

x

The default properties object.

RETURN VALUES

None.

RvMdmTermDefaultProperties Accessor Functions

This section includes:

- `rvMdmTermDefaultPropertiesSetDigitMap()`
- `RvMdmTermDefaultPropertiesSetPassword()`
- `RvMdmTermDefaultPropertiesSetUsername()`

RvMdmTermDefaultProperties Accessor Functions

rvMdmTermDefaultPropertiesSetDigitMap()

rvMdmTermDefaultPropertiesSetDigitMap()

DESCRIPTION

Sets the default list of digit map patterns.

SYNTAX

```
void rvMdmTermDefaultPropertiesSetDigitMap(  
    IN RvMdmTermDefaultProperties*    x,  
    IN RvMdmDigitMap*                 digitMap,  
    IN const RvChar*                   name);
```

PARAMETERS

x

The default properties object.

digitMap

The digitmap patterns.

name

The digitmap's name.

RETURN VALUES

None.

SEE ALSO

[RvMdmDigitMap Module](#)

RvMdmTermDefaultPropertiesSetPassword()

DESCRIPTION

Sets the password parameter. This parameter will be used for Authentication.

SYNTAX

```
void rvMdmTermDefaultPropertiesSetPassword(  
    IN RvMdmTermDefaultProperties*    x,  
    IN const RvChar*                  password)
```

PARAMETERS

[x](#)

The default properties object.

[password](#)

The password.

RETURN VALUE

None.

SEE ALSO

[RvMdmTermDefaultPropertiesSetUsername\(\)](#)

RvMdmTermDefaultPropertiesSetUsername()

DESCRIPTION

Sets the username parameter. This parameter will be used for Authentication.

SYNTAX

```
void rvMdmTermDefaultPropertiesSetUsername(  
    IN RvMdmTermDefaultProperties*    x,  
    IN const RvChar*                  username)
```

PARAMETERS

x

The default properties object.

username

The user name.

RETURN VALUE

None.

SEE ALSO

[RvMdmTermDefaultPropertiesSetPassword\(\)](#)

13

RvMdmTermMgr MODULE

WHAT'S IN THIS CHAPTER

This chapter contains the functions and used to operate on RvMdmTermMgr objects.

The RvMdmTermMgr functions construct and destruct the MDM Termination Manager, which manages a set of terminations and the signals and events that can be applied to them for a given stack instance.

This chapter includes:

- [RvMdmTermMgr General Control Functions](#)
- [RvMdmTermMgr Accessor Functions](#)
- [RvMdmTermMgr Callback Functions](#)

RvMDMTERMGR GENERAL CONTROL FUNCTIONS

This section includes:

- `rvMdmTermMgrDestruct()`
- `rvMdmTermMgrRegisterAllTermsToNetwork()`
- `rvMdmTermMgrRegisterTermToNetwork()`
- `rvMdmTermMgrUnregisterTermFromNetwork()`
- `rvMdmTermMgrRegisterEphemeralTermination()`
- `rvMdmTermMgrRegisterPhysicalTermination()`
- `rvMdmTermMgrStart()`
- `rvMdmTermMgrStop()`
- `rvMdmTermMgrUnregisterTermination()`
- `rvMdmTermMgrRegisterPhysicalTerminationAsync()`
- `rvMdmTermMgrUnregisterTerminationAsync()`

rvMdmTermMgrDestruct()

DESCRIPTION

Destructs the Termination Manager and releases all the memory.

SYNTAX

```
void rvMdmTermMgrDestruct (  
    IN RvMdmTermMgr* mgr);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

RETURN VALUES

None.

rvMdmTermMgrRegisterAllTermsToNetwork()

DESCRIPTION

Registers all UI and Analog terminations to the Network Server (Proxy or Gatekeeper).

SYNTAX

```
RvBool rvMdmTermMgrRegisterAllTermsToNetwork(  
    IN RvMdmTermMgr* mgr)
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

RETURN VALUE

Returns RV_FALSE if it fails to register at least one of the terminations.

rvMdmTermMgrRegisterTermToNetwork()

DESCRIPTION

Registers one termination to the Network Server (Proxy or Gatekeeper), provided it is of the UI or Analog type.

SYNTAX

```
RvBool rvMdmTermMgrRegisterTermToNetwork(  
    IN RvMdmTermMgr*   mgr,  
    IN RvMdmTerm*      term)
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

term

A pointer to the termination.

RETURN VALUE

Returns RV_FALSE if it fails to register the termination.

rvMdmTermMgrUnregisterTermFromNetwork()

DESCRIPTION

The user application should call this function whenever it wants to unregister a termination from the Server. This API sends an Unregister message to the SIP Server that is configured in the parameter "registrarAddress".

SYNTAX

```
RvBool   rvMdmTermMgrUnregisterTermFromNetwork(  
    IN RvMdmTermMgr*   mgr,  
    IN RvMdmTerm*      term);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

term

A pointer to the termination that needs to be unregistered.

RETURN VALUE

Returns RV_FALSE in case sending the Unregister request to the Server has failed, or RV_TRUE if it was successful.

rvMdmTermMgrRegisterEphemeralTermination()

DESCRIPTION

Registers an ephemeral Termination of an existing class with the Termination Manager.

SYNTAX

```
RvMdmTerm* rvMdmTermMgrRegisterEphemeralTermination(  
    IN RvMdmTermMgr*      mgr,  
    IN RvMdmTermClass*    c,  
    IN const RvChar*      id,  
    IN RvMdmTermDefaultProperties* termProperties);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

c

A pointer to a previously created and initialized Termination class.

id

The Termination id.

termProperties

Default properties of the Termination, or NULL.

RETURN VALUES

A pointer to the new Termination, or NULL if it fails.

REMARKS

This function will not signal that the new Termination is coming up.

RvMdmTermMgr General Control Functions

`rvMdmTermMgrRegisterEphemeralTermination()`

SEE ALSO

[RvMdmTermClass Module](#)

[RvMdmTermDefaultProperties Module](#)

rvMdmTermMgrRegisterPhysicalTermination()

DESCRIPTION

Registers a physical Termination of an existing class with the Termination Manager.

SYNTAX

```
RvMdmTerm* rvMdmTermMgrRegisterPhysicalTermination(  
    IN RvMdmTermMgr*      mgr,  
    IN RvMdmTermClass*    c,  
    IN const RvChar*      id,  
    IN RvMdmTermDefaultProperties* termProperties,  
    IN RvMdmServiceChange* sc);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

c

A pointer to a previously created and initialized Termination class.

id

The Termination id.

termProperties

Default properties of the Termination or NULL.

sc

This parameter is unused and should be set to NULL.

RETURN VALUES

Returns a pointer to the new Termination, or NULL if it fails.

RvMdmTermMgr General Control Functions

`rvMdmTermMgrRegisterPhysicalTermination()`

REMARKS

If the function is called after calling [rvMdmTermMgrStart\(\)](#), it will signal that the new Termination is coming up.

SEE ALSO

[RvMdmTerm Module](#)

[RvMdmTermDefaultProperties Module](#)

rvMdmTermMgrStart()

DESCRIPTION

After calling this function the Termination Manager will start processing commands and events, and the Media Gateway will signal that it is going up.

SYNTAX

```
void rvMdmTermMgrStart(  
    IN RvMdmTermMgr*      mgr,  
    IN RvMdmServiceChange* sc,  
    IN RvInt32             delay);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

sc

This parameter is unused and should be set to NULL.

delay

The delay (in milliseconds) to wait before starting the MDM.

RETURN VALUES

None.

REMARKS

This function is usually called after registering the physical endpoints.

rvMdmTermMgrStop()

DESCRIPTION

Calling this function will cause the Media Gateway to signal that it is going down.

SYNTAX

```
void rvMdmTermMgrStop(  
    IN RvMdmTermMgr*      mgr,  
    IN RvMdmServiceChange* sc);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

sc

This parameter is unused and should be set to NULL.

RETURN VALUES

None.

rvMdmTermMgrUnregisterTermination()

DESCRIPTION

Unregisters a Termination. An unregistered Termination will not receive or send messages.

SYNTAX

```
RvBool rvMdmTermMgrUnregisterTermination(  
    IN RvMdmTermMgr*      mgr,  
    IN RvMdmTerm*         term,  
    IN RvMdmServiceChange* sc);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

term

A pointer to a Termination.

sc

This parameter is unused and should be set to NULL.

RETURN VALUES

Returns RV_FALSE if it fails.

REMARKS

If the function is called after calling [rvMdmTermMgrStart\(\)](#), and the Termination is of type Physical, it will signal that the Termination is going down.

SEE ALSO

[RvMdmTerm Module](#)
[rvMdmTermMgrStart\(\)](#)

rvMdmTermMgrRegisterPhysicalTerminationAsync()

DESCRIPTION

Registers a physical termination in an asynchronous manner.

SYNTAX

```
RvMdmTerm* rvMdmTermMgrRegisterPhysicalTerminationAsync(  
    IN RvMdmTermMgr*          mgr,  
    IN RvMdmTermClass*        c,  
    IN const RvChar*          id,  
    IN RvMdmTermDefaultProperties* termProperties,  
    IN void*                   userData)
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

c

A pointer to a previously created and initialized Termination class.

id

The Termination Id.

termProperties

The default properties of the Termination or NULL.

userData

The user data associated with the newly registered termination. This user data can be accessed through [rvMdmTermGetUserData\(\)](#) and [rvMdmTermSetUserData\(\)](#).

REMARKS

If the function is called after calling [rvMdmTermMgrStart\(\)](#), it will signal that the new termination is coming up.

RETURN VALUES

Returns a pointer to the new Termination, or NULL if it fails.

SEE ALSO

[RvMdmTermClass Module](#)

[RvMdmTermDefaultProperties Module](#)

[rvMdmTermMgrStart\(\)](#)

rvMdmTermMgrUnregisterTerminationAsync()

DESCRIPTION

Registers a physical Termination in an asynchronous manner.

SYNTAX

```
RvBool rvMdmTermMgrUnregisterTerminationAsync (
    IN RvMdmTermMgr*      mgr,
    IN RvMdmTerm*         mdmTerm,
    IN RvMdmServiceChange* sc) ;
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

mdmTerm

A pointer to the Termination.

sc

This parameter is unused and should be set to NULL.

RETURN VALUES

Returns a pointer to the new Termination, or NULL if it fails.

REMARKS

If the function is called after calling [rvMdmTermMgrStart\(\)](#), it will signal that the Termination is going down.

SEE ALSO

[RvMdmTerm Module](#)
[rvMdmTermMgrStart\(\)](#)

RvMDMTERMGR ACCESSOR FUNCTIONS

This section includes:

- rvMdmTermMgrFindTermination()
- rvMdmTermMgrForEachPhysicalTerm()
- rvMdmTermMgrGetIdleTermination()
- rvMdmTermMgrGetPackage()
- rvMdmTermMgrGetUserData()
- rvMdmTermMgrRegisterConnectCB()
- rvMdmTermMgrRegisterDeleteEphTermCB()
- rvMdmTermMgrRegisterDisconnectCB()
- rvMdmTermMgrRegisterSelectTermCB()
- rvMdmTermMgrSetUserData()
- rvMdmTermMgrCreateTermClass()

rvMdmTermMgrFindTermination()

DESCRIPTION

Finds a Termination by Id.

SYNTAX

```
RvMdmTerm* rvMdmTermMgrFindTermination(  
    IN RvMdmTermMgr*    mgr,  
    IN const RvChar*    id);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

id

The Termination id.

RETURN VALUES

A pointer to the Termination, or NULL if not found.

rvMdmTermMgrForEachPhysicalTerm()

DESCRIPTION

Calls the func parameter for each registered physical Termination until func returns RV_TRUE or there are no more terminations.

SYNTAX

```
RvBool rvMdmTermMgrForEachPhysicalTerm(  
    IN RvMdmTermMgr*      mgr,  
    IN RvMdmProcessEachTermCB func,  
    IN void*               data);
```

PARAMETERS

mgr

The Termination Manager.

func

The function to call.

data

User data to pass to the function.

RETURN VALUES

Returns the value returned by the last call to the func parameter.

REMARKS

The function type func must be:

```
typedef RvBool (RvMdmProcessEachTermCB) (  
    IN RvMdmTerm*      term,  
    IN void*           data);
```

rvMdmTermMgrGetIdleTermination()

DESCRIPTION

Returns an idle Termination when matching a partially specified id.

SYNTAX

```
RvMdmTerm* rvMdmTermMgrGetIdleTermination(  
    IN RvMdmTermMgr*    mgr,  
    IN const RvChar*    id);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

id

The partial id. Use an empty string to get any idle Termination; use '\$' to match a specific pattern.

RETURN VALUES

A pointer to the Termination, or NULL if not found.

rvMdmTermMgrGetPackage()

DESCRIPTION

Gets a handler to an existing package. This function is used to overwrite or add values.

SYNTAX

```
RvMdmPackage* rvMdmTermMgrGetPackage (  
    IN RvMdmTermMgr*    mgr,  
    IN const RvChar*    name);
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

name

The package name.

RETURN VALUES

The package handler, or NULL if not previously registered.

rvMdmTermMgrGetUserData()

DESCRIPTION

Returns the user data associated with the Termination Manager.

SYNTAX

```
void* rvMdmTermMgrGetUserData(  
    IN RvMdmTermMgr* mgr);
```

PARAMETERS

mgr

A pointer to the Termination Manager.

RETURN VALUES

The Termination Manager user data or NULL if not set.

SEE ALSO

[rvMdmTermMgrSetUserData\(\)](#)

rvMdmTermMgrRegisterConnectCB()

DESCRIPTION

Sets the callback function to call when connecting a media stream in one Termination to a media stream in another Termination.

SYNTAX

```
void rvMdmTermMgrRegisterConnectCB (  
    IN RvMdmTermMgr*          m,  
    IN RvMdmTermMgrConnectCB  connectF );
```

PARAMETERS

[mgr](#)

A pointer to the Termination Manager class object.

[connectF](#)

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermMgrConnectCB\(\)](#)

RvMdmTermMgr Accessor Functions

rvMdmTermMgrRegisterDeleteEphTermCB()

rvMdmTermMgrRegisterDeleteEphTermCB()

DESCRIPTION

Sets the callback function to be called to release an ephemeral Termination.

SYNTAX

```
void rvMdmTermMgrRegisterDeleteEphTermCB (  
    IN RvMdmTermMgr*          mgr,  
    IN RvMdmTermMgrDeleteEphTermCB deleteEphF) ;
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

deleteEphF

The callback function.

RETURN VALUES

None.

SEE ALSO

RvMdmTermMdmDeleteEphTermCB()

rvMdmTermMgrRegisterDisconnectCB()

DESCRIPTION

Sets the callback function to call when disconnecting a media stream in a Termination of this class from a media stream in another Termination.

SYNTAX

```
void rvMdmTermMgrRegisterDisconnectCB(  
    IN RvMdmTermMgr*          c,  
    IN RvMdmTermMgrDisconnectCB disconnectF);
```

PARAMETERS

c

A pointer to the Termination class object.

disconnectF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermMgrDisconnectCB\(\)](#)

rvMdmTermMgrRegisterSelectTermCB()

DESCRIPTION

Sets the callback function to be called to select a Termination.

SYNTAX

```
void rvMdmTermMgrRegisterSelectTermCB (  
    IN RvMdmTermMgr*                mgr,  
    IN RvMdmTermMgrSelectTerminationCB selectF) ;
```

PARAMETERS

mgr

A pointer to the Termination Manager object.

selectF

The callback function.

RETURN VALUES

None.

SEE ALSO

[RvMdmTermMgrSelectTerminationCB\(\)](#)

rvMdmTermMgrSetUserData()

DESCRIPTION

Sets user data associated with the Termination.

SYNTAX

```
void rvMdmTermMgrSetUserData(  
    IN RvMdmTermMgr*    mgr,  
    IN void*             data);
```

PARAMETERS

mgr

A pointer to the Termination Manager.

data

The user data.

RETURN VALUES

None.

REMARKS

Use this function to associate application data and memory with the Termination Manager.

SEE ALSO

[rvMdmTermMgrGetUserData\(\)](#)

rvMdmTermMgrCreateTermClass()

DESCRIPTION

Allocates and constructs a new Termination class in the Termination Manager.

SYNTAX

```
RvMdmTermClass* rvMdmTermMgrCreateTermClass(  
    IN RvMdmTermMgr* mgr);
```

PARAMETERS

[mgr](#)

A pointer to the Termination Manager object.

RETURN VALUES

Returns the new Termination class or NULL.

REMARKS

Use the returned RvMdmTermClass to set the properties for the new class.

SEE ALSO

[RvMdmTermClass Module](#)

RvMDMTERMGR CALLBACK FUNCTIONS

This section includes:

- RvMdmTermRegisterPhysTermCompletedCB()
- RvMdmTermUnregisterTermCompletedCB()
- RvMdmTermUnregisterTermFromNetworkCompletedCB()

RvMdmTermRegisterPhysTermCompletedCB()

DESCRIPTION

This callback is called after [rvMdmTermMgrRegisterPhysicalTermination\(\)](#) or [rvMdmTermMgrRegisterPhysicalTerminationAsync\(\)](#) was called, to indicate that the registration process is complete.

SYNTAX

```
void RvMdmTermRegisterPhysTermCompletedCB (  
    IN RvMdmTerm*      term,  
    IN RvMdmError*     mdmError);
```

PARAMETERS

[term](#)

A pointer to a termination that has completed registration.

[mdmError](#)

This parameter is not in use.

RETURN VALUES

None.

SEE ALSO

[rvMdmTermClassRegisterRegisterPhysTermCompletedCB\(\)](#)

RvMdmTermUnregisterTermCompletedCB()

DESCRIPTION

This callback is called after [rvMdmTermMgrUnregisterTermination\(\)](#) or [rvMdmTermMgrUnregisterTerminationAsync\(\)](#) was called, to indicate that the unregistration process is complete.

SYNTAX

```
void RvMdmTermUnregisterTermCompletedCB(  
    IN RvMdmTerm*      term,  
    IN RvMdmError*     mdmError);
```

PARAMETERS

[term](#)

A pointer to a termination that has completed unregistration.

[mdmError](#)

This parameter is not in use.

RETURN VALUES

None.

SEE ALSO

[rvMdmTermClassRegisterUnregisterTermCompletedCB\(\)](#)

RvMdmTermMgr Callback Functions

RvMdmTermUnregisterTermFromNetworkCompletedCB()

RvMdmTermUnregisterTermFromNetworkCompletedCB())

DESCRIPTION

The user application should implement this callback, which notifies the application that the unregistration process from the server is complete (i.e., a reply was received, or the timer expired). The user application cannot call a Register/Unregister API or shutdown the Multimedia Terminal Framework before this callback has been called.

SYNTAX

```
void RvMdmTermUnregisterTermFromNetworkCompletedCB(  
    IN RvMdmTerm*      term,  
    IN RvmdmError*     mdmError)
```

PARAMETERS

term

A pointer to the termination that needs to be unregistered

mdmError

This parameter is currently not in use.

RETURN VALUES

None.

SEE ALSO

rvMdmTermClassRegisterUnregisterTermFromNetworkCompletedCB()

14

CALL FORWARD MODULE

WHAT'S IN THIS CHAPTER

This chapter includes generic call forward functions and structures for the Multimedia Terminal Framework.

This chapter includes:

- Call Forward Functions
- Call Forward Callbacks
- Call Forward Structures
- Call Forward Type Definitions

CALL FORWARD FUNCTIONS

This section includes:

- `rvCCCfwGetTypeNumber()`

rvCCCfwGetTypeNumber()

DESCRIPTION

Converts from CFW type string to CFW type number.

SYNTAX

```
RvIppCfwType rvCCCfwGetTypeNumber(  
    IN const RvChar *typeValue);
```

PARAMETERS

[typeValue](#)

Pointer to CFW type string.

RETURN VALUE

Returns the valid CFW type Number if the string is valid. Returns RV_IPP_CFW_TYPE_NUM if the string is unknown.

SEE ALSO

[RvIppCfwType](#)

REMARKS

The translation is done using the table below:

typeValue string	Returned RvIppCfwType value
"cfwu"	RV_IPP_CFW_TYPE_UNCONDITIONAL
"cfwb"	RV_IPP_CFW_TYPE_BUSY
"cfnr"	RV_IPP_CFW_TYPE_NO_REPLY
Other	RV_IPP_CFW_TYPE_NONE

CALL FORWARD CALLBACKS

This section includes:

- `rvIppCfwActivateCompletedCB()`
- `rvIppCfwDeactivateCompletedCB()`

rvIppCfwActivateCompletedCB()

DESCRIPTION

This callback notifies the user when the activation process has ended, and indicates whether it was completed successfully or not. If process has failed, the reason of the failure will be indicated by this callback.

SYNTAX

```
void rvIppCfwActivateCompletedCB(  
    IN RvIppTerminalHandle    term,  
    IN RvIppCfwType           cfwType,  
    IN RvChar*                 cfwDestination,  
    IN RvIppCfwReturnReasons  returnCode);
```

PARAMETERS

[term](#)

Handle to termination to which the activation process applied.

[cfwType](#)

Type of CFW to which the activation process applied.

[cfwDestination](#)

Destination address to which calls will be forwarded.

[returnCode](#)

Reason code, indicating whether the process was completed successfully or not.

RETURN VALUE

None.

SEE ALSO

[RvIppTerminalHandle](#)

[RvIppCfwType](#)

Call Forward Callbacks

`rvIppCfwActivateCompletedCB()`

`RvIppCfwReturnReasons`

`rvIppCfwDeactivateCompletedCB()`

rvIppCfwDeactivateCompletedCB()

DESCRIPTION

This callback notifies the user when the deactivation process has ended, and indicates whether it was completed successfully or not. If the process failed, the reason of the failure will be indicated by this callback.

SYNTAX

```
void rvIppCfwDeactivateCompletedCB(  
    IN RvIppTerminalHandle    term,  
    IN RvIppCfwType           cfwType,  
    IN RvIppCfwReturnReasons  returnCode) ;
```

PARAMETERS

term

Handle to the termination to which the deactivation process applied.

cfwType

Type of CFW to which the deactivation process applied.

returnCode

Reason code, indicating whether the process was completed successfully or not.

RETURN VALUE

None.

SEE ALSO

[RvIppTerminalHandle](#)
[RvIppCfwType](#)
[RvIppCfwReturnReasons](#)
[rvIppCfwActivateCompletedCB\(\)](#)

CALL FORWARD STRUCTURES

This section includes:

- [RvIppCfwCBs](#)
- [RvIppCfwCfg](#)

RvIppCfwCBs

DESCRIPTION

This structure includes user callback definitions.

SYNTAX

```
typedef struct
{
    rvIppCfwActivateCompletedCB    activateCompleted;
    rvIppCfwDeactivateCompletedCB  deactivateCompleted;
} RvIppCfwCBs;
```

SEE ALSO

[RvIppCfwType](#)

[rvIppCfwActivateCompletedCB\(\)](#)

[rvIppCfwDeactivateCompletedCB\(\)](#)

RvIppCfwCfg

DESCRIPTION

This structure includes configuration parameters that should be set during initialization, to support call forwarding.

SYNTAX

```
typedef struct
{
    IN cfwCallbacks;
} RvIppCfwCfg;
```

PARAMETERS

[cfwCallbacks](#)

The callbacks implemented by the application that deal with call forwarding.

SEE ALSO

[RvIppCfwCBs](#)

[RvIppSipPhoneCfg](#)

CALL FORWARD TYPE DEFINITIONS

This section includes:

- RvIppCfwType
- RvIppCfwReturnReasons

RvIppCfwType

DESCRIPTION

The type of call forwarding being invoked.

SYNTAX

```
typedef enum
{
    RV_IPP_CFW_TYPE_UNCONDITIONAL,
    RV_IPP_CFW_TYPE_BUSY,
    RV_IPP_CFW_TYPE_NO_REPLY,
    RV_IPP_CFW_TYPE_NONE
} RvIppCfwType;
```

PARAMETERS

[RV_IPP_CFW_TYPE_UNCONDITIONAL](#)

Call forwarding is unconditional. The call should be forwarded no matter what the terminal's status is.

[RV_IPP_CFW_TYPE_BUSY](#)

Call forwarding should be invoked if the terminal is busy.

[RV_IPP_CFW_TYPE_NO_REPLY](#)

Call forwarding should be invoked when the terminal does not reply on incoming calls.

[RV_IPP_CFW_TYPE_NONE](#)

No call forwarding should be used.

SEE ALSO

[rvCCCfwGetTypeNumber\(\)](#)

RvIppCfwReturnReasons

DESCRIPTION

Return reasons for the activation and deactivation callbacks of call forwarding.

SYNTAX

```
typedef enum
{
    RV_IPP_CFW_SUCCESS,
    RV_IPP_CFW_INVALID_DEACTIVATION,
    RV_IPP_CFW_INVALID_PARAM,
    RV_IPP_CFW_ADDRESS_NOT_FOUND,
    RV_IPP_CFW_NOT_ALLOWED,
    RV_IPP_CFW_CANCELLED_BY_USER
} RvIppCfwReturnReasons;
```

PARAMETERS

RV_IPP_CFW_SUCCESS

Activation or deactivation procedure has successfully finished.

RV_IPP_CFW_INVALID_DEACTIVATION

Forwarding type was not activated.

RV_IPP_CFW_INVALID_PARAM

One of parameters is not valid.

RV_IPP_CFW_ADDRESS_NOT_FOUND

Local mapping of phone number failed.

RV_IPP_CFW_NOT_ALLOWED

Not allowed in the "blocked" cases.

RV_IPP_CFW_CANCELLED_BY_USER

Forwarding request for activation or deactivation was canceled by the user.

Call Forward Type Definitions

RvIppCfwReturnReasons

SEE ALSO

[rvIppCfwActivateCompletedCB\(\)](#)

[rvIppCfwDeactivateCompletedCB\(\)](#)

15

LOGGER MODULE

WHAT'S IN THIS CHAPTER

This chapter includes functions and type definitions related to logging for the Multimedia Terminal Framework.

This chapter includes:

- [Logger Functions](#)
- [Logger Type Definitions](#)

LOGGER FUNCTIONS

This section includes:

- IppLogInit()
- IppLogReload()
- IppLogEnd()

IppLogInit()

DESCRIPTION

Initializes Multimedia Terminal Framework logging. This function should be called once, during the construction of an Multimedia Terminal Framework instance.

SYNTAX

```
RvStatus IppLogInit(  
    IN IppLogSourceFilterElm*    ippFilters,  
    IN const RvChar*             szLogFileName);
```

PARAMETERS

[ippFilters](#)

An array of log filters to include in the log file. The array must end with an element that has an empty name for its log source.

RETURN VALUE

RV_OK on success, other on failure.

IppLogReload()

DESCRIPTION

Resets the log sources that are used for logging. This function cannot be called before [IppLogInit\(\)](#) is called.

SYNTAX

```
RvStatus IppLogReload(  
    IN IppLogSourceFilterElm*    ippFilters);
```

PARAMETERS

[ippFilters](#)

An array of log filters to include in the log file. The array must end with an element that has an empty name for its log source.

RETURN VALUE

RV_OK on success, other on failure.

IppLogEnd()

DESCRIPTION

Ends the use of the log in the Multimedia Terminal Framework. This function must be called last, after the Multimedia Terminal Framework has been terminated.

SYNTAX

```
RvStatus IppLogEnd(void);
```

PARAMETERS

None.

RETURN VALUE

RV_OK on success, other on failure.

IppLogMessage()

DESCRIPTION

Prints a message into the Multimedia Terminal Framework log. The message will be printed under the IPP_USERAPP source.

SYNTAX

```
void IppLogMessage(  
    IN RvBool          isError,  
    IN const RvChar*   message, ...);
```

PARAMETERS

isError

RV_TRUE for an error message, RV_FALSE for an information message.

message

The message itself. This string is handled in the same manner of the ANSI C printf() function.

RETURN VALUE

None.

LOGGER TYPE DEFINITIONS

This section includes:

- [IppLogSourceFilterElm](#)

IppLogSourceFilterElm

DESCRIPTION

A log source filter to be show in the log file of the Multimedia Terminal Framework. This struct is used as an array of elements passed to [IppLogInit\(\)](#). When done in this manner, to indicate the size of the array, the last element's logSrcName field should be set to an empty string.

SYNTAX

```
typedef struct
{
    RvChar                logSrcName[20];
    RvLogMessageType      messageMask;
} IppLogSourceFilterElm;
```

PARAMETERS

[logSrcName\[20\]](#)

This field indicates the module name. For a list of possible names, see the Logging chapter in the Programmer Guide.

[messageMask](#)

This field indicates the logging mask of the module. For a list of possible log masks, see the Logging chapter in the Programmer Guide.

16

ENUMERATED TYPES

WHAT'S IN THIS CHAPTER

This chapter includes the enumerated types of the Media Device Manager API.

This chapter includes:

- [Enumerated Types](#)

ENUMERATED TYPES

This section includes:

- RvMdmSignalType
- RvMdmStreamDirection
- RvMdmStreamMode
- RvMdmTermType
- RvMdmRelation
- RvCCTerminalState
- RvCCConnState
- RvCCTermConnState
- RvCCMediaState
- RvCCConnType
- RvCCCallState
- RvCCTerminalType
- RvCCTerminalEvent
- RvCCEventCause

RvMdmSignalType

DESCRIPTION

The signal type.

SYNTAX

```
typedef enum
{
    RV_MDMSIGNAL_DEFAULTTYPE,
    RV_MDMSIGNAL_ONOFF,
    RV_MDMSIGNAL_TIMEOUT,
    RV_MDMSIGNAL_BRIEF
} RvMdmSignalType;
```

TYPE VALUES

RV_MDMSIGNAL_DEFAULTTYPE

Use the normal type for the given signal.

RV_MDMSIGNAL_ONOFF

The signal lasts until it is turned off.

RV_MDMSIGNAL_TIMEOUT

The signal lasts until it is turned off or a specific period of time elapses.

RV_MDMSIGNAL_BRIEF

The signal duration is so short that it will stop on its own unless a new signal is applied that causes it to stop.

RvMdmStreamDirection

DESCRIPTION

Describes the direction of connected media flows.

TYPE VALUES

RV_MDMSTREAMDIRECTION_ISOLATE

No media flow.

RV_MDMSTREAMDIRECTION_BOTHWAYS

Bidirectional media flow.

RV_MDMSTREAMDIRECTION_SOURCE2TARGET

Unidirectional flow, from source to target.

RV_MDMSTREAMDIRECTION_TARGET2SOURCE

Unidirectional flow, from target to source.

RvMdmStreamMode

DESCRIPTION

Media streams can be either incoming or outgoing in terms of the media being sent on them. They can also be bidirectional. This enumeration enables the application to know what mode a specific stream uses, allowing it to handle the connection of the different devices with the RTP accordingly.

SYNTAX

```
typedef enum
{
    RV_MDMSTREAMMODE_NOTSET,
    RV_MDMSTREAMMODE_INACTIVE,
    RV_MDMSTREAMMODE_SENDOONLY,
    RV_MDMSTREAMMODE_RECVONLY,
    RV_MDMSTREAMMODE_SENDRECV,
    RV_MDMSTREAMMODE_LOOPBACK
} RvMdmStreamMode;
```

TYPE VALUES

RV_MDMSTREAMMODE_NOTSET

The stream mode is yet unknown.

RV_MDMSTREAMMODE_INACTIVE

The stream is inactive. It can probably be closed.

RV_MDMSTREAMMODE_SENDOONLY

The stream supports send only.

RV_MDMSTREAMMODE_RECVONLY

The stream supports receive only.

RV_MDMSTREAMMODE_SENDRECV

The stream supports both sending and receiving.

Enumerated Types

RvMdmStreamMode

RV_MDMSTREAMMODE_LOOPBACK

This enumeration value is reserved for future use.

RvMdmTermType

DESCRIPTION

Describes the Termination type.

SYNTAX

```
typedef enum
{
    RV_MDMTERMTYPE_UNKNOWN,
    RV_MDMTERMTYPE_PHYSICAL,
    RV_MDMTERMTYPE_EPHEMERAL
} RvMdmTermType;
```

TYPE VALUES

[RV_MDMTERMTYPE_UNKNOWN](#)

Undefined or unresolved type (for temporary Termination).

[RV_MDMTERMTYPE_PHYSICAL](#)

Physical Termination.

[RV_MDMTERMTYPE_EPHEMERAL](#)

Ephemeral Termination.

SEE ALSO

[rvMdmTermGetType\(\)](#)

RvMdmRelation

DESCRIPTION

An interpretation of parameter value.

SYNTAX

```
typedef enum
{
    RV_MDM_RELATION_EQUAL,
    RV_MDM_RELATION_AND,
    RV_MDM_RELATION_OR
} RvMdmRelation;
```

SEE ALSO

[RvMdmParameterValue Module](#)

RvCCTerminalState

DESCRIPTION

The terminal's state.

SYNTAX

```
typedef enum
{
    RV_CCTERMINAL_IDLE_STATE,
    RV_CCTERMINAL_CFW_ACTIVATING_STATE,
    RV_CCTERMINAL_ERROR_STATE
} RvCCTerminalState;
```

TYPE VALUES

[RV_CCTERMINAL_IDLE_STATE](#)

All events are accepted at any time.

[RV_CCTERMINAL_CFW_ACTIVATING_STATE](#)

CFW activating process started. This is used to block non-relevant events to be handled. For example, ONHOOK, TRANSFER events, etc.

[RV_CCTERMINAL_ERROR_STATE](#)

An error has occurred on this terminal.

SEE ALSO

[rvIppMdmTerminalGetState\(\)](#)

[rvIppMdmTerminalSetState\(\)](#)

RvCCConnState

DESCRIPTION

The connection's state.

SYNTAX

```
typedef enum
{
    RV_CCCONNSTATE_IDLE,
    RV_CCCONNSTATE_INITIATED,
    RV_CCCONNSTATE_DIALING,
    RV_CCCONNSTATE_ADDRESS_ANALYZE,
    RV_CCCONNSTATE_INPROCESS,
    RV_CCCONNSTATE_CALL_DELIVERED,
    RV_CCCONNSTATE_OFFERED,
    RV_CCCONNSTATE_ALERTING,
    RV_CCCONNSTATE_DISCONNECTED,
    RV_CCCONNSTATE_CONNECTED,
    RV_CCCONNSTATE_FAILED,
    RV_CCCONNSTATE_TRANSFER_INIT,
    RV_CCCONNSTATE_TRANSFER_INPROCESS,
    RV_CCCONNSTATE_TRANSFER_DELIVERED,
    RV_CCCONNSTATE_TRANSFER_OFFERED,
    RV_CCCONNSTATE_TRANSFER_ALERTING,
    RV_CCCONNSTATE_REJECTED,
    RV_CCCONNSTATE_ALERTING_REJECTED,
    RV_CCCONNSTATE_UNKNOWN,
    RV_CCCONNSTATE_USER
} RvCCConnState;
```

TYPE VALUES

RV_CCCONNSTATE_IDLE

This state is the initial state for all new connections. Connections in the IDLE state are not actively part of a call. Connections typically do not stay in the IDLE state for long, but instead transition to other states.

RV_CCCONNSTATE_INITIATED

This state indicates that the originating end of a call has begun the process of placing a call, but has not yet begun dialing the destination address. Typically, a terminal (phone) has gone off-hook.

RV_CCCONNSTATE_DIALING

This state indicates that the originating end of a call has begun dialing a destination telephone address, but has not yet completed dialing. At this stage the user callback [RvMdmTermMatchDialStringCB\(\)](#) is called for every digit received.

RV_CCCONNSTATE_ADDRESS_ANALYZE

This state is entered when the complete initial information package or dialing string from the originating party are available. At this stage the user callback [RvMdmTermMapDialStringToAddressCB\(\)](#) is called. This state is exited when the routing address becomes available.

RV_CCCONNSTATE_INPROCESS

This state implies that the connection object is contacting the destination side. The contact is established as a result of the underlying protocol messages.

RV_CCCONNSTATE_CALL_DELIVERED

This state indicates that an outgoing call is being offered to the destination side (which is in an ALERTING state). For incoming calls, the connection transitions to this state after the call is answered but before transitioning to the CONNECTED state.

RV_CCCONNSTATE_OFFERED

This state indicates that an incoming call is being offered to the connection.

RV_CCCONNSTATE_ALERTING

This state implies that the Terminal is being notified of an incoming call.

RV_CCCONNSTATE_DISCONNECTED

This state implies that the connection is no longer part of the call. A connection in this state is interpreted as one that previously belonged to a call.

RV_CCCONNSTATE_CONNECTED

This state implies that a connection is actively part of a call. In common terms, two people talking to one another are represented by two connections in the CONNECTED state.

RV_CCCONNSTATE_FAILED

This state indicates that a connection to this end of the call has failed.

RV_CCCONNSTATE_TRANSFER_INIT

This state is relevant for transferring endpoint (A). The state indicates that the Transfer process has been started.

RV_CCCONNSTATE_TRANSFER_INPROCESS

This state is relevant for transferring endpoint (A). The state is relevant to RV_CCCONNSTATE_INPROCESS, and indicates that the Transfer destination has been contacted.

RV_CCCONNSTATE_TRANSFER_DELIVERED

This state is relevant for transferee endpoint (B). For incoming transfer calls (User C in the example above), the connection transitions to this state after the call is answered and before transitioning to the CONNECTED state.

RV_CCCONNSTATE_TRANSFER_OFFERED

This state is relevant for transfer destination endpoint (C). This state indicates that an incoming transferred call is being offered to the connection. This connection represents the transfer destination (User C in the example above) and the call replaces the existing call with the transferring party. The call moves to the TRANSFER_ALERTING state.

RV_CCCONNSTATE_TRANSFER_ALERTING

This state is relevant for transfer destination endpoint (C). This state indicates that an incoming transferred call is alerting in the connection. This connection represents the transfer destination, and the call replaces the existing call with the transferring party. The call is automatically accepted and will move to the TRANSFER_DELIVERED state.

RV_CCCONNSTATE_REJECTED

This state indicates that an incoming call has been rejected before transitioning to the OFFERED state. For example, this may have happened because no lines were available.

RV_CCCONNSTATE_ALERTING_REJECTED

This state indicates that an incoming call has been rejected by the user while it was already in the ALERTING state. For example, because the user pressed a REJECT key.

RV_CCCONNSTATE_UNKNOWN

The state is unknown.

RV_CCCONNSTATE_USER

This must always be the final one.

SEE ALSO

[rvIppMdmConnGetState\(\)](#)

RvCCTermConnState

DESCRIPTION

The Termination connection's state.

SYNTAX

```
typedef enum
{
    RV_CCTERMCONSTATE_IDLE,
    RV_CCTERMCONSTATE_RINGING,
    RV_CCTERMCONSTATE_TALKING,
    RV_CCTERMCONSTATE_HELD,
    RV_CCTERMCONSTATE_REMOTE_HELD,
    RV_CCTERMCONSTATE_BRIDGED,
    RV_CCTERMCONSTATE_DROPPED,
    RV_CCTERMCONSTATE_MUTE,
    RV_CCTERMCONSTATE_REMOTE_HELD_LOCAL_HELD
} RvCCTermConnState;
```

TYPE VALUES

RV_CCTERMCONSTATE_IDLE

There are no calls on the terminal; all lines are idle.

RV_CCTERMCONSTATE_RINGING

The active line is ringing with the incoming call.

RV_CCTERMCONSTATE_TALKING

The active line is connected to the remote party.

RV_CCTERMCONSTATE_HELD

The active line has put the call on Hold.

RV_CCTERMCONSTATE_REMOTE_HELD

The active line was put on Hold by the remote party.

RV_CCTERMCONSTATE_BRIDGED

At least one call is connected, in addition to the active line.

RV_CCTERMCONSTATE_DROPPED

Either the local party or the remote party is disconnected.

RV_CCTERMCONSTATE_MUTE

The active line is Mute.

RV_CCTERMCONSTATE_REMOTE_HELD_LOCAL_HELD

Both parties have put the call on Hold.

SEE ALSO

[rvIppMdmConnGetTermState\(\)](#)

RvCCMediaState

DESCRIPTION

The media state, as associated with a given connection.

SYNTAX

```
typedef enum
{
    RV_CCEDIASTATE_NONE,
    RV_CCEDIASTATE_CREATING,
    RV_CCEDIASTATE_CREATED,
    RV_CCEDIASTATE_CONNECTED,
    RV_CCEDIASTATE_DISCONNECTED,
    RV_CCEDIASTATE_NOTSUPPORTED,
    RV_CCEDIASTATE_MODIFYING,
    RV_CCEDIASTATE_FAILED
} RvCCMediaState;
```

SEE ALSO

[rvIppMdmConnGetMediaState\(\)](#)

RvCCConnType

DESCRIPTION

The type of connections in the Multimedia Terminal Framework.

SYNTAX

```
typedef enum
{
    RV_CCCONNTYPE_MDM,
    RV_CCCONNTYPE_NETWORK
} RvCCConnType;
```

TYPE VALUES

[RV_CCCONNTYPE_MDM](#)

The connection type is an MDM one, associated with a local Termination.

[RV_CCCONNTYPE_NETWORK](#)

The connection is a network connection, associated with a remote client or server.

SEE ALSO

[rvIppMdmConnGetType\(\)](#)

Enumerated Types

RvCCallState

RvCCallState

DESCRIPTION

The call's state.

SYNTAX

```
typedef enum
{
    RV_CCCALLSTATE_NORMAL,
    RV_CCCALLSTATE_CONFERENCE_INIT,
    RV_CCCALLSTATE_CONFERENCE_COMPLETED,
    RV_CCCALLSTATE_TRANSFER_INIT
} RvCCallState;
```

SEE ALSO

[rvIppMdmConnGetCallState\(\)](#)

RvCCTerminalType

DESCRIPTION

The type associated with a given terminal.

SYNTAX

```
typedef enum
{
    RV_CCTERMINALTYPE_UNKNOWN,
    RV_CCTERMINALTYPE_EPHEMERAL,
    RV_CCTERMINALTYPE_ANALOG,
    RV_CCTERMINALTYPE_UI,
#ifdef RV_MTF_VIDEO
    RV_CCTERMINALTYPE_VT,
#endif
    RV_CCTERMINALTYPE_AT
} RvCCTerminalType;
```

TYPE VALUES

RV_CCTERMINALTYPE_UNKNOWN

The terminal type is unknown.

RV_CCTERMINALTYPE_EPHEMERAL

This is an ephemeral terminal, usually associated with an RTP session.

RV_CCTERMINALTYPE_ANALOG

This terminal is physical terminal, associated with an analog line.

RV_CCTERMINALTYPE_UI

A user interface termination, associated with keys, LCD displays, lights, etc.

RV_CCTERMINALTYPE_VT

Video transducer termination, representing a video device in the phone.

Enumerated Types

RvCCTerminalType

[RV_CCTERMINALTYPE_AT](#)

Audio transducer termination, representing an audio device in the phone.

SEE ALSO

[rvIppMdmTerminalGetType\(\)](#)

RvCCTerminalEvent

DESCRIPTION

Termination events. These events are indicated through [RvIppMdmPreProcessEventCB\(\)](#) and [RvIppMdmPostProcessEventCB\(\)](#), enabling the application to follow on the exact events occurring on a given termination. These events can affect the connections, the terminations, or the media itself.

SYNTAX

```
typedef enum
{
    RV_CCTERMEVENT_NONE,
    RV_CCTERMEVENT_UNKNOWN,
    RV_CCTERMEVENT_GW_ACTIVE,
    RV_CCTERMEVENT_OFFHOOK,
    RV_CCTERMEVENT_DIALTONE,
    RV_CCTERMEVENT_DIGITS,
    RV_CCTERMEVENT_DIGIT_END,
    RV_CCTERMEVENT_DIALCOMPLETED,
    RV_CCTERMEVENT_MAKECALL,
    RV_CCTERMEVENT_RINGBACK,
    RV_CCTERMEVENT_RINGING,
    RV_CCTERMEVENT_CALLANSWERED,
    RV_CCTERMEVENT_ONHOOK,
    RV_CCTERMEVENT_HOLD,
    RV_CCTERMEVENT_MUTE,
    RV_CCTERMEVENT_HOLDKEY,
    RV_CCTERMEVENT_UNHOLD,
    RV_CCTERMEVENT_CONFERENCE,
    RV_CCTERMEVENT_TRANSFER,
    RV_CCTERMEVENT_LINE,
    RV_CCTERMEVENT_LINEOTHER,
    RV_CCTERMEVENT_HEADSET,
    RV_CCTERMEVENT_HANDSFREE,
```

Enumerated Types

RvCCTerminalEvent

```
RV_CCTERMEVENT_AUDIOHANDSET,  
RV_CCTERMEVENT_AUDIOHANDSFREE,  
RV_CCTERMEVENT_FAILGENERAL,  
RV_CCTERMEVENT_MEDIAOK,  
RV_CCTERMEVENT_MEDIAFAIL,  
RV_CCTERMEVENT_DISCONNECTING,  
RV_CCTERMEVENT_DISCONNECTED,  
RV_CCTERMEVENT_INCOMINGCALL,  
RV_CCTERMEVENT_REJECTCALL,  
RV_CCTERMEVENT_TRANSFER_INIT,  
RV_CCTERMEVENT_TRANSFER_OFFERED,  
RV_CCTERMEVENT_REMOTE_DISCONNECTED,  
RV_CCTERMEVENT_ONHOOK_OTHER,  
RV_CCTERMEVENT_REJECT_KEY,  
RV_CCTERMEVENT_MEDIANOTACCEPTED,  
RV_CCTERMEVENT_MODIFYMEDIA,  
RV_CCTERMEVENT_MODIFYMEDIA_DONE,  
RV_CCTERMEVENT_BLIND_TRANSFER,  
RV_CCTERMEVENT_REDIAL,  
RV_CCTERMEVENT_CFW,  
RV_CCTERMEVENT_USER  
} RvCCTerminalEvent;
```

PARAMETERS

RV_CCTERMEVENT_NONE

No action will be taken.

RV_CCTERMEVENT_UNKNOWN

Unknown event, will be ignored.

RV_CCTERMEVENT_GW_ACTIVE

This event indicates that the gateway is active. It is used by the Call Control to initialize the display. Caused by an rvcc/ga event.

RV_CCTERMEVENT_OFFHOOK

The user has gone off-hook (has lifted the phone receiver) or has pressed a Line key. These events are logically equivalent, as a call can be answered or initiated by pressing the Line key. Caused by a kf/kd event with keyid of kh for a UI termination, or an al/of event for an Analog termination.

RV_CCTERMEVENT_DIALTONE

Applied by the Call Control to move from INITIATED to DIALING event.

RV_CCTERMEVENT_DIGITS

The user has pressed a DTMF key. Caused by a kp/kd event for UI termination, or a dd/d(n) event for Analog termination.

RV_CCTERMEVENT_DIGIT_END

The user has released a DTMF key. Caused by a kp/ku event for UI termination. Not used in Analog terminations.

RV_CCTERMEVENT_DIALCOMPLETED

Dialing is complete. This can either be an internal event (if the user returns from the RvMdmTermMatchDialStringCB() callback with a value RV_MDMDIGITMAP_UNAMBIGUOUSMATCH or RV_MDMDIGITMAP_NOMATCH) or sent directly by the user (if the user does the match asynchronously). For the UI termination, it is caused by the kp/ce event. For the Analog termination, it is caused by the dd/ce event.

RV_CCTERMEVENT_MAKECALL

Internal Call Control event. It initiates an incoming call in the connection.

RV_CCTERMEVENT_RINGBACK

Internal Call Control event. The outgoing call has reached the other end.

RV_CCTERMEVENT_RINGING

Internal Call Control event. An incoming call is transitioning to the ALERTING state.

RV_CCTERMEVENT_CALLANSWERED

Internal Call Control event. An outgoing call has been answered in the destination end and is moving to a CONNECTED state.

RV_CCTERMEVENT_ONHOOK

The user went on-hook, thereby disconnecting the call. For the UI termination this can be caused by a kf/ku event with keyid of kh, an al/on event, or a line event (kf/ku with keyid 100n) if the line was already in a call. For the Analog termination it is caused by the al/on event.

RV_CCTERMEVENT_HOLD

Internal hold event propagated to the Connection State Machine. This can originate, for example, from a user pressing the Hold key.

RV_CCTERMEVENT_MUTE

The user pressed the Mute key. Currently not implemented.

RV_CCTERMEVENT_HOLDKEY

The user pressed the Hold key. Caused by a kf/ku event with keyid of kl.

RV_CCTERMEVENT_UNHOLD

Internal Unhold event propagated to the Connection State Machine. This can originate, for example, from a user pressing the Line key on a line in a HELD state.

RV_CCTERMEVENT_CONFERENCE

The user pressed the Conference key. Caused by a kf/ku event with keyid of kc. The first time will activate an additional line to call the added party. The second time will connect all parties in the conference.

RV_CCTERMEVENT_TRANSFER

The user pressed the Transfer key. Caused by a kf/ku event with keyid of kt. The first time will activate an additional line to call the transfer destination. The second time will complete the transfer and drop the user from the transfer.

RV_CCTERMEVENT_LINE

The user pressed the Line key. Caused by a kf/ku event with keyid of l00n.

RV_CCTERMEVENT_LINEOTHER

This is an internal event. The user pressed the Line key, and there is a connected call (either on hold or not) in a different line.

RV_CCTERMEVENT_HEADSET

The user pressed the Headset key. Caused by a kf/ku event with keyid of ht.

RV_CCTERMEVENT_HANDSFREE

The user pressed the Handsfree key. Caused by a kf/ku event with keyid of hf.

RV_CCTERMEVENT_AUDIOHANDSET

This is an internal event. The event indicates the Call Control to make the Handset the active audio termination, meaning to move the media from the current active termination to the new one, etc.

RV_CCTERMEVENT_AUDIOHANDSFREE

This is an internal event. The event indicates to the state machine to make the Speaker the active audio termination, i.e., to move the media from the current active termination to the new one, etc.

RV_CCTERMEVENT_FAILGENERAL

This is an internal event. The event indicates a general failure, while the reason code specifies the reason for the failure.

RV_CCTERMEVENT_MEDIAOK

Internal Call Control event. Creating or modifying a media stream on a termination has succeeded.

RV_CCTERMEVENT_MEDIAFAIL

Internal Call Control event. Creating or modifying a media stream on a termination has failed.

RV_CCTERMEVENT_DISCONNECTING

Internal Call Control event. Causes the connection to disconnect from the call.

RV_CCTERMEVENT_DISCONNECTED

Internal Call Control event. The connection has been disconnected. Release resources.

RV_CCTERMEVENT_INCOMINGCALL

This is an internal event. The event is usually caused by an incoming signaling message indicating the establishment of a new call.

RV_CCTERMEVENT_REJECTCALL

Internal Call Control event. Rejects an incoming call before the OFFERED state. For example, because there are no available lines.

RV_CCTERMEVENT_TRANSFER_INIT

This event indicates to transferring endpoint (A) to start the Transfer process, i.e., to send the transferree endpoint (B) a signaling message indicating that it should establish a call with the Transfer destination endpoint.

RV_CCTERMEVENT_TRANSFER_OFFERED

Internal Call Control event. An incoming transferred call is being offered to the connection. This connection represents the transfer destination. The call will replace the existing call with the transferring party.

RV_CCTERMEVENT_REMOTE_DISCONNECTED

Internal Call Control event. The other party has disconnected the call.

RV_CCTERMEVENT_ONHOOK_OTHER

Internal Call Control event.

RV_CCTERMEVENT_REJECT_KEY

The user has rejected an incoming call on a given line. Caused by rvcc/reject event.

RV_CCTERMEVENT_MEDIANOTACCEPTED

This is an internal event. The event indicates that the media offered by the remote party is not supported by the local party. Processing the event will result in a local warning tone and an outgoing signaling message, indicating that the call is rejected with reason 415—Media Not Supported for SIP.

RV_CCTERMEVENT_MODIFYMEDIA

This event is relevant for SIP Phone only. The event indicates the beginning of a dynamic media change process and is caused by the user application calling [rvMdmTermModifyMedia\(\)](#). Processing the event will result in a Re-Invite message being sent that includes the new media.

RV_CCTERMEVENT_MODIFYMEDIA_DONE

This is an internal event. The event is processed when the state machine has finished processing a dynamic media change and will result in notifying the user application about whether or not the process has been completed successfully.

RV_CCTERMEVENT_BLIND_TRANSFER

The user pressed the Blind Transfer key. Caused by a kf/ku event with keyid of kbt.

RV_CCTERMEVENT_REDIAL

This event is sent to activate Redial functionality.

RV_CCTERMEVENT_CFW

This event is sent to activate Call Forward functionality.

RV_CCTERMEVENT_USER

All values higher than this value can be used by the user and will be ignored by the Call Control.

SEE ALSO

[RvCCEventCause](#)

RvCCEventCause

DESCRIPTION

The event reason field provides additional information to the Call Control about the events. These reasons are indicated by [RvIppMdmPreProcessEventCB\(\)](#), [RvIppMdmPostProcessEventCB\(\)](#) and [RvIppMdmDisplayCB\(\)](#).

SYNTAX

```
typedef enum
{
    RV_CCCAUSE_INCOMING_CALL,
    RV_CCCAUSE_OUTGOING_CALL,
    RV_CCCAUSE_CALL_WAITING,
    RV_CCCAUSE_BUSY,
    RV_CCCAUSE_NOT_FOUND,
    RV_CCCAUSE_REORDER_TONE,
    RV_CCCAUSE_TRANSFER,
    RV_CCCAUSE_UNHOLD,
    RV_CCCAUSE_CALL_CANCELLED,
    RV_CCCAUSE_LOCAL_HOLD,
    RV_CCCAUSE_REMOTE_HOLD,
    RV_CCCAUSE_NEW_CALL,
    RV_CCCAUSE_NORMAL,
    RV_CCCAUSE_RESOURCES_NOT_AVAILABLE,
    RV_CCCAUSE_MEDIA_NOT_SUPPORTED,
    RV_CCCAUSE_EVENT_BEGIN,
    RV_CCCAUSE_EVENT_END,
    RV_CCCAUSE_OPERATION_SUCCEEDED,
    RV_CCCAUSE_OPERATION_FAILED,
    RV_CCCAUSE_AUTH_FAIL,
    RV_CCCAUSE_UNKNOWN
} RvCCEventCause;
```


PARAMETERS

RV_CCCAUSE_INCOMING_CALL

The call is an incoming call.

RV_CCCAUSE_OUTGOING_CALL

The call is an outgoing call.

RV_CCCAUSE_CALL_WAITING

The incoming call is a call waiting (there is an active call in the terminal already).

RV_CCCAUSE_BUSY

The call is not completed because the party is busy (no lines available).

RV_CCCAUSE_NOT_FOUND

Reason for rejecting an incoming call destination was not found.

RV_CCCAUSE_REORDER_TONE

Indicates a general failure, will cause a warning tone to be heard.

RV_CCCAUSE_TRANSFER

The reason for the call is transfer.

RV_CCCAUSE_UNHOLD

Call was released from Hold.

RV_CCCAUSE_CALL_CANCELLED

Call was canceled by the local party.

RV_CCCAUSE_LOCAL_HOLD

The call was put on hold by the local user.

RV_CCCAUSE_REMOTE_HOLD

The call was put on Hold by the remote party. The event is caused by an incoming signaling message indicating that the call should be put on Hold.

RV_CCCAUSE_NEW_CALL

The call is a new call.

RV_CCCAUSE_NORMAL

This reason is used when no specific reason is required.

RV_CCCAUSE_RESOURCES_NOT_AVAILABLE

Operation failed due to lack of resources.

RV_CCCAUSE_MEDIA_NOT_SUPPORTED

The phone does not support the required media parameters.

RV_CCCAUSE_EVENT_BEGIN

This reason indicates that Key Down was pressed for a digit.

RV_CCCAUSE_EVENT_END

This reason indicates that Key Up was pressed for a digit.

RV_CCCAUSE_OPERATION_SUCCEEDED

The process ended successfully.

RV_CCCAUSE_OPERATION_FAILED

The process ended with failure.

RV_CCCAUSE_AUTH_FAIL

(SIP only) Authentication failed.

RV_CCCAUSE_UNKNOWN

Reason is unknown.

SEE ALSO

[RvCCTerminalEvent](#)

Enumerated Types

RvCCEventCause

PART 2: SIP CONTROL

17

SIP CONTROL GENERAL

WHAT'S IN THIS CHAPTER

This chapter includes SIP control functions.

This chapter includes:

- [SIP Control General Functions](#)
- [SIP Control General Type Definitions](#)

SIP CONTROL GENERAL FUNCTIONS

This section includes:

- `rvIppSipSystemInit()`
- `rvIppSipSystemEnd()`
- `rvIppSipRegisterExtCbks`
- `rvIppSipInitConfig()`
- `rvIppSipStackInitialize()`
- `rvIppSipPhoneConstruct()`
- `rvIppSipControlGetRegistrarAddress()`
- `rvIppSipControlGetUserDomain()`
- `rvIppSipControlGetOutboundProxyAddress()`
- `rvIppSipControlGetExtUserData()`
- `rvIppSipControlGetTransportType()`
- `rvIppSipControlGetStackHandle()`
- `rvIppSipControlGetStackCallbacks()`
- `rvIppSipControlMsgSetSdpBody()`
- `RvIppSipStackCallbacks`

rvIppSipSystemInit()

DESCRIPTION

Initializes the IP Phone elements that must be ready before the Multimedia Terminal Framework is constructed:

- Initialize core
- Reset extension APIs structure

This function is used to initialize the Multimedia Terminal Framework and should be the first function to be called.

SYNTAX

```
void rvIppSipSystemInit(void);
```

PARAMETERS

None.

RETURN VALUE

None.

SEE ALSO

[rvIppSipSystemEnd\(\)](#)

rvlppSipSystemEnd()

DESCRIPTION

Shutdown IP Phone system, end core.

This function is used to shutdown the Multimedia Terminal Framework and should be called as the final function.

SYNTAX

```
void rvlppSipSystemEnd(void);
```

PARAMETERS

None.

RETURN VALUE

None.

SEE ALSO

[rvlppSipSystemInit\(\)](#)

rvIppSipRegisterExtCbks

DESCRIPTION

Registers SIP user callbacks. These callbacks can be used by the application to extend the functionality of the Terminal Framework. This function should be called before [rvIppSipStackInitialize\(\)](#).

SYNTAX

```
void rvIppSipRegisterExtCbks(  
    IN RvIppSipExtCbks* clbks);
```

PARAMETERS

[clbks](#)

Structure that includes pointers to user implementations.

RETURN VALUE

None.

SEE ALSO

[RvIppSipExtCbks](#)

rvlppSipInitConfig()

DESCRIPTION

Initializes the configuration structure.

This function fills the configuration structure with the default values. It should be called prior to calling [rvIppSipStackInitialize\(\)](#). The application can modify the values within the struct after calling this function and before initializing the SIP Stack.

SYNTAX

```
RvBool rvlppSipInitConfig(  
    OUT RvIppSipPhoneCfg*    cfg) ;
```

OUTPUT PARAMETERS

[cfg](#)

Configuration structure to fill with default values.

RETURN VALUE

RV_TRUE if successful. RV_FALSE if failed.

SEE ALSO

[RvIppSipPhoneCfg](#)
[rvIppSipStackInitialize\(\)](#)
[rvIppSipPhoneConstruct\(\)](#)
[rvIppSipTlsInitConfig\(\)](#)

rvIppSipStackInitialize()

DESCRIPTION

Initializes and configures the SIP Stack.

SYNTAX

```
RvBool rvIppSipStackInitialize(  
    OUT RvSipStackHandle*    stackHandle,  
    IN  RvIppSipPhoneCfg*    cfg);
```

PARAMETERS

[cfg](#)

Structure of the Multimedia Terminal Framework configuration parameters.

OUTPUT PARAMETERS

[stackHandle](#)

Handle to the SIP Stack that was constructed.

RETURN VALUE

Return RV_TRUE if the Stack was successfully initialized. RV_FALSE if failed.

SEE ALSO

[RvIppSipPhoneCfg](#)

[rvIppSipInitConfig\(\)](#)

[rvIppSipPhoneConstruct\(\)](#)

rvlppSipPhoneConstruct()

DESCRIPTION

Constructs the SIP Multimedia Terminal Framework. After calling this function, the application should register the various callbacks.

SYNTAX

```
void rvlppSipPhoneConstruct(  
    OUT RvMdmTermMgr*      mgr,  
    IN  RvIppSipPhoneCfg*   cfg,  
    IN  RvSipStackHandle    sipStack);
```

PARAMETERS

[cfg](#)

The pointer to the configuration data.

[sipStack](#)

The SIP Stack handle.

OUTPUT PARAMETERS

[mgr](#)

The pointer to the MdmTermination Manager.

RETURN VALUE

None.

SEE ALSO

[RvMdmTermMgr Module](#)
[RvIppSipPhoneCfg](#)
[rvIppSipStackInitialize\(\)](#)
[rvMdmTermMgrSetUserData\(\)](#)

rvIppSipControlGetRegistrarAddress()

DESCRIPTION

Returns the IP address of the Registrar.

SYNTAX

```
void rvIppSipControlGetRegistrarAddress(  
    IN  RvIppSipControlHandle  sipMgrHndl,  
    OUT RvChar*                 registrarAddress,  
    IN  RvSize_t                registrarAddressLen);
```

PARAMETERS

[sipMgrHndl](#)

Handle to the SipControl object.

[registrarAddressLen](#)

The length of the registrarAddress string buffer provided by the application.

OUTPUT PARAMETERS

[registrarAddress](#)

The address of the Registrar.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

rvlppSipControlGetUserDomain()

DESCRIPTION

Returns the domain of the User.

SYNTAX

```
void rvlppSipControlGetUserDomain(  
    IN  RvIppSipControlHandle    sipMgrHndl,  
    OUT RvChar*                  userDomain,  
    IN  RvSize_t                  userDomainLen);
```

PARAMETERS

[sipMgrHndl](#)

Handle to the SipControl object.

[userDomainLen](#)

The maximum length of the userDomain string buffer provided by the application.

OUTPUT PARAMETERS

[userDomain](#)

The domain of the User.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

rvIppSipControlGetOutboundProxyAddress()

DESCRIPTION

Returns the IP address of the Outbound Proxy.

SYNTAX

```
void rvIppSipControlGetOutboundProxyAddress (  
    IN  RvIppSipControlHandle    sipMgrHndl,  
    OUT RvChar*                  outboundProxyAddress,  
    IN  RvSize_t                  outboundProxyAddressLen);
```

PARAMETERS

[sipMgrHndl](#)

Handle to the SipControl object.

[outboundProxyAddressLen](#)

The maximum length of the outboundProxyAddress string buffer provided by the application.

OUTPUT PARAMETERS

[outboundProxyAddress](#)

The address of the Outbound Proxy.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

rvIppSipControlGetExtUserData()

DESCRIPTION

Returns user data that was set by the user when extension callbacks were registered.

SYNTAX

```
void* rvIppSipControlGetExtUserData(  
    IN RvIppSipControlHandle    sipMgrHndl);
```

PARAMETERS

[sipMgrHndl](#)

Handle to the SipControl object.

RETURN VALUE

User data.

SEE ALSO

[RvIppSipControlHandle](#)

[rvIppSipRegisterExtCbks](#)

rvIppSipControlGetTransportType()

DESCRIPTION

Checks the transport type that is used.

SYNTAX

```
RvSipTransport rvIppSipControlGetTransportType(  
    IN RvIppSipControlHandle    sipMgrHndl);
```

PARAMETERS

[sipMgrHndl](#)

Handle to SipControl object

RETURN VALUE

Transport type.

SEE ALSO

[RvIppSipControlHandle](#)

rvIppSipControlGetStackHandle()

DESCRIPTION

Returns the SIP Stack handle stored inside the SipControl object. The Stack handle can be used to access SIP Stack APIs directly when the need arises.

SYNTAX

```
RvSipStackHandle rvIppSipControlGetStackHandle(  
    IN RvIppSipControlHandle    sipMgrHndl);
```

PARAMETERS

[sipMgrHndl](#)

Handle to the SipControl object.

RETURN VALUE

The SIP Stack handle.

SEE ALSO

[RvIppSipControlHandle](#)

rvIppSipControlGetStackCallbacks()

DESCRIPTION

Gets the list of Stack callbacks used by the Multimedia Terminal Framework.

SYNTAX

```
RvIppSipStackCallbacks* rvIppSipControlGetStackCallbacks(  
    IN RvIppSipControlHandle    sipMgrHndl);
```

PARAMETERS

[sipMgrHndl](#)

Handle to the SipControl object.

RETURN VALUE

Pointer to the SIP Stack callbacks that are used.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipStackCallbacks](#)

rvlppSipControlMsgSetSdpBody()

DESCRIPTION

Adds an SDP body to a SIP message.

SYNTAX

```
RvBool rvlppSipControlMsgSetSdpBody(  
    IN RvSipMsgHandle    hMsg,  
    IN const RvSdpMsg*    sdpMsg);
```

PARAMETERS

hMsg

Handle to the SIP message.

sdpMsg

Pointer to the SDP structure.

RETURN VALUE

RV_TRUE if successful. RV_FALSE if failed.

RvIppSipStackCallbacks

DESCRIPTION

This structure holds a set of callbacks that the Multimedia Terminal Framework uses on top of the SIP Stack. The application can override the callbacks used, or add callbacks that are never used by the Multimedia Terminal Framework to extend the functionality of the application.

SYNTAX

```
typedef struct
{
    RvSipSubsEvHandlers          sipSubsEvHandlers;
    RvSipCallLegEvHandlers       sipCallLegEvHandlers;
    RvSipTransactionEvHandlers   sipTransEvHandlers;
    RvSipRegClientEvHandlers     sipRegClientEvHandlers;
    RvSipAuthenticatorEvHandlers sipAuthEvHandlers;
    RvSipTransportMgrEvHandlers  sipTransportEvHandlers;
} RvIppSipStackCallbacks;
```

PARAMETERS

[sipSubsEvHandlers](#)

SIP callbacks related to subscription events.

[sipCallLegEvHandlers](#)

SIP callbacks related to call-leg events.

[sipTransEvHandlers](#)

SIP callbacks related to transaction events.

[sipRegClientEvHandlers](#)

SIP callbacks related to client registration events.

[sipAuthEvHandlers](#)

SIP callbacks related to authentication events.

SIP Control General Functions

RvIppSipStackCallbacks

[sipTransportEvHandlers](#)

SIP callbacks related to network transport events.

NOTES

- Setting the callbacks should be done with care in order not to break the existing implementation.
- Setting the callbacks the relevant SIP Stack event handler setting function for each field in this struct.

SEE ALSO

[rvIppSipControlGetStackCallbacks\(\)](#)

SIP CONTROL GENERAL TYPE DEFINITIONS

This section includes:

- `RvIppSipPhoneCfg`
- `RvIppSipControlHandle`

RvIppSipPhoneCfg

DESCRIPTION

This structure contains the parameters for the configuration of the SIP Stack before calling [rvIppSipStackInitialize\(\)](#).

SYNTAX

```
typedef struct
{
    RvUInt16          stackTcpPort;
    RvUInt16          stackUdpPort;
    RvChar*           userDomain;
    RvChar*           localAddress;
    RvChar*           registrarAddress;
    RvUInt16          registrarPort;
    RvChar*           outboundProxyAddress;
    RvUInt16          outboundProxyPort;
    RvSipTransport     transportType;
    int               maxCallLegs;
    int               maxRegClients;
    RvBool            tcpEnabled;
    int               priority;
    RvChar*           username;
    RvChar*           password;
    RvBool            autoRegister;
    RvInt32            registrationExpire;
    RvInt32            unregistrationExpire;
    RvInt32            referTimeout;
    int               debugLevel;
    unsigned int       dialToneDuration;
    RvUInt3            watchdogTimeout;
    RvCallWaitingReply callWaitingReply;
    RvBool            outOfBandDtmf;

#ifdef RV_CFLAG_TLS
```

```
        RvIppTransportTlsCfg    transportTlsCfg;  
#endif  
        RvIppCfwCfg             cfwCallCfg;  
} RvIppSipPhoneCfg;
```

TYPE VALUES

stackTcpPort

The TCP port on which the Stack listens.
Default: 5060.

stackUdpPort

The UDP port on which the Stack listens.
Default: 5060.

userDomain

This domain name will be sent in the From header of outgoing INVITE messages in either of the following cases: If the Registrar address parameter is configured as an IP address, or if the Registrar address is not configured at all.

localAddress

Local IP address. This parameter is mandatory and may not be left empty.

registrarAddress

The Registrar IP address or domain name. If this parameter is not set, Registration messages will not be sent.
Default: NULL.

registrarPort

The number of the Port on which the Registrar listens.
Default: 5060.

outboundProxyAddress

The IP address of the outbound Proxy. If this parameter is set, all outgoing messages (including Registration messages) will be sent to this Proxy according to the Stack behavior.

Default: NULL.

Note To configure a host name instead of an IP address, set this parameter to NULL and configure the outboundProxyHostName parameter in the SIP Stack.

outboundProxyPort

The number of the Port on which the outbound Proxy listens.

Default: 5060.

transportType

The Transport type of the outgoing messages. Valid values are:

- RVSIP_TRANSPORT_UDP
- RVSIP_TRANSPORT_TCP

Default: RVSIP_TRANSPORT_UDP.

maxCallLegs

The maximum number of call-legs the Stack can handle simultaneously.

Default: 10.

maxRegClients

The maximum number of RegClient objects the Stack can handle simultaneously.

Default: 2.

tcpEnabled

This parameter indicates support for TCP. When set to RV_TRUE, the Terminal Framework will support a TCP connection. When set to RV_FALSE, the Terminal Framework will not support a TCP connection (incoming TCP messages will be ignored).

Default: RV_TRUE.

priority

The priority of the Terminal Framework task. It is recommended to use one of the following values, which define the appropriate value according to the operating system:

- RV_THREAD_PRIORITY_MAX
- RV_THREAD_PRIORITY_DEFAULT
- RV_THREAD_PRIORITY_MIN

These values are defined in the file rvthread.h.

Default: RV_THREAD_PRIORITY_DEFAULT.

username

Used for Authentication. This parameter can also be configured for each termination separately. If this parameter is configured, it will be used for all terminations whose username has not been configured. If this parameter is left empty and is not configured in a termination, the Authentication header will not be sent with the Registration request.

Default: NULL.

password

Used for Authentication. This parameter can also be configured for each termination separately. If this parameter is configured, it will be used for all terminations whose username has not been configured. If this parameter is left empty and is not configured in a termination, the Authentication header will not be sent with the Registration request.

Default: NULL.

autoRegister

If set to RV_TRUE, the Terminal Framework will send the initial registration request to the Registrar for every termination that registers to the Multimedia Terminal Framework. If set to RV_FALSE, the initial registration request will not be sent. It can be sent manually at any time by calling [rvMdmTermMgrRegisterAllTermsToNetwork\(\)](#) or [rvMdmTermMgrRegisterTermToNetwork\(\)](#). Regardless of the value of this parameter, the Terminal Framework will send re-Registration requests.

Default: RV_FALSE.

registrationExpire

The timeout (in seconds) for sending Re-registration requests to the Registrar.
Default: 60,000 seconds.

unregistrationExpire

This parameter defines the timeout (in seconds) to wait for a reply from the Registrar, after an Unregistration request was sent. The Unregistration request is sent when the user application unregisters a termination from the Multimedia Terminal Framework by calling [rvMdmTermMgrUnregisterTermination\(\)](#). Once this function is called, an un-registration process begins. This process is asynchronous. If the termination is registered with a Registrar, the Multimedia Terminal Framework will send an Unregistration request to the Registrar and wait for a reply. The UnregistrationExpire parameter defines the timeout to wait for a reply. The process will continue only when the timeout expires or when a reply is received from the Registrar. When either of these scenarios occurs, the Multimedia Terminal Framework will call the user callback [RvMdmTermUnregisterTermCompletedCB\(\)](#) to notify the user application that the unregistration process has ended.

Default: 20 seconds.

referTimeout

The timeout (in milliseconds) for waiting for NOTIFY after sending REFER, before disconnecting the call-leg.

Default: 2,000 milliseconds.

debugLevel

This parameter is not used.

logOptions

This parameter defines log options for the Sip Stack. See the Logging chapter in the Programmer's Guide.

Default: NULL.

dialToneDuration

Duration of Dial Tone signal (in milliseconds) when going off-hook. When the user goes off-hook and timeout expires, Dial Tone will be stopped and the connection will disconnect. A value of 0 indicates an infinite Dial Tone.

Default: 30,000 milliseconds.

watchdogTimeout

This parameter indicates the time interval (in seconds) used for periodically printing the resources of the SIP Stack and the Multimedia Terminal Framework. Multimedia Terminal Framework resources will be printed only if the Multimedia Terminal Framework is **not** compiled with the flag `RV_MTF_PERFORMANCE_ON`. If this parameter is set to zero, the timer is disabled.

Watchdog printouts will be added to the log only if the module `IPP_SIPCTRL` was added to the log configuration. See the Logging chapter in the Programmer's Guide.

Default: 0.

callWaitingReply

When the incoming call is a Call Waiting call, this parameter indicates which SIP message will be sent as a reply to the INVITE. Possible values are:

- `RV_REPLY_RINGING=180`
- `RV_REPLY_QUEUED=182`

Default: `RV_REPLY_QUEUED`.

outOfBandDtmf

When this parameter is set to `RV_TRUE`, out-of-band DTMF is enabled.

transportTlsCfg

TLS Transport-related configuration.

For more information, see [RvIppTransportTlsCfg](#).

cfwCallCfg

Call Forwarding-related configuration.

For more information, see [RvIppCfwCfg](#).

transportType

The Transport type of the outgoing messages. Valid values are:

- `_RVSIP_TRANSPORT_UDP`
- `_RVSIP_TRANSPORT_TCP`

SIP Control General Type Definitions

RvIppSipPhoneCfg

- RVSIP_TRANSPORT_UNDEFINED

RvIppSipControlHandle

DESCRIPTION

SIP Control object handle. SIP Control handle enables the user application with better control of the SIP Stack in changing or adding to the default functionality of the Multimedia Terminal Framework.

SYNTAX

```
RV_DECLARE_HANDLE (RvIppSipControlHandle);
```

SIP Control General Type Definitions

RvIppSipControlHandle

18

STUN MODULE

WHAT'S IN THIS CHAPTER

This chapter describes STUN functions. STUN is an optional add-on module to the MTF, which can be used for NAT traversal.

This chapter includes:

- [STUN Functions](#)
- [STUN Callbacks](#)
- [STUN Type Definitions](#)

STUN FUNCTIONS

This section describes the following STUN functions.

- `RvIppStunMgrCreate()`
- `RvIppStunAddressResolveComplete()`
- `RvIppStunMgrDelete()`
- `RvIppStunMgrGetSendMethod()`

RvIppStunMgrCreate()

DESCRIPTION

Initializes STUN support in the Multimedia Terminal Framework (StunMgr object). This function should be called after a terminal instance is initialized using [rvIppSipPhoneConstruct\(\)](#).

SYNTAX

```
RvIppStunMgrHandle RvIppStunMgrCreate(  
    IN RvIppStunMgrParam* param);
```

PARAMETERS

[param](#)

Parameters with which the StunMgr object is initialized.

RETURN VALUE

Handle of the StunMgr object on success. NULL on failure.

SEE ALSO

[RvIppStunMgrParam](#)

[RvIppStunMgrDelete\(\)](#)

RvIppStunAddressResolveComplete()

DESCRIPTION

Function called by the STUN Client of the user application when the STUN resolution procedure for concrete {ip:port} has been completed or has failed.

SYNTAX

```
RvStatus RvIppStunAddressResolveComplete(  
    IN RvIppStunAddrData*    addrData,  
    IN RvBool                 bStatusOK);
```

PARAMETERS

[addrData](#)

Data describing the STUN response:

- ▣ [addrData.inAddr](#)—STUN resolution procedure requested
- ▣ [addrData.outAddr](#)—"mapped to" response of the STUN server

[bStatusOK](#)

RV_TRUE if the STUN resolution procedure has succeeded. Otherwise RV_FALSE.

RETURN VALUE

RV_OK on success.

SEE ALSO

[RvIppAddressResolveReplyCB\(\)](#)

RvIppStunMgrDelete()

DESCRIPTION

Stops STUN support in the Multimedia Terminal Framework (destroys the StunMgr object).

SYNTAX

```
void RvIppStunMgrDelete(  
    IN RvIppStunMgrHandle stunMgrHndl);
```

PARAMETERS

[stunMgrHndl](#)

Handle of the StunMgr object.

RETURN VALUE

None.

SEE ALSO

[RvIppStunMgrCreate\(\)](#)

STUN Functions

RvIppStunMgrGetSendMethod()

RvIppStunMgrGetSendMethod()

DESCRIPTION

API used when the STUN Client of the user application tries to send a STUN message over a SIP signaling socket to get the NAT mapping of its address.

SYNTAX

```
void RvIppStunMgrGetSendMethod(  
    IN  RvIppStunMgrHandle    stunMgr,  
    OUT SendMethod**          method);
```

PARAMETERS

stunMgr

StunMgr handle.

OUTPUT PARAMETERS

method

Pointer to a structure containing a function that is called to send a buffer over a socket.

RETURN VALUE

RV_OK on success.

STUN CALLBACKS

This section describes the following STUN functions.

- `RvIppStunIsAddressResolveNeededCB()`
- `RvIppStunAddressResolveStartCB()`
- `RvIppAddressResolveReplyCB()`

STUN Callbacks

RvIppStunIsAddressResolveNeededCB()

RvIppStunIsAddressResolveNeededCB()

DESCRIPTION

Callback prototype implemented by the STUN Client of the user application. Checks if a STUN resolution is required for SIP messages sent to a concrete destination endpoint. Implies that this destination resides outside the private network.

The STUN module in the Multimedia Terminal Framework invokes this callback for each outgoing message, letting the application decide if it wishes to search and replace the given address in the callback with a different one.

SYNTAX

```
RvBool RvIppStunIsAddressResolveNeededCB(  
    IN RvAddress* addrDest);
```

PARAMETERS

[addrDest](#)

Address of the destination endpoint.

RETURN VALUE

RV_TRUE if the destination endpoint lies outside of the private network. .

RV_FALSE should be returned by the application if no address resolution is required.

SEE ALSO

[RvIppStunAddressResolveStartCB\(\)](#)

RvIppStunAddressResolveStartCB()

DESCRIPTION

Callback prototype implemented by the STUN Client of the user application. Starts a STUN resolution procedure for a specific {ip:port} pair.

At this stage the application should use its STUN Client to get a public address to replace the one given to the callback as a parameter.

SYNTAX

```
RvStatus RvIppStunAddressResolveStartCB(  
    IN RvIppStunAddrData* addrData);
```

PARAMETERS

[addrData](#)

Data describing the STUN request. Only `addrData.inAddr` is relevant.

RETURN VALUE

RV_OK if the user has started a STUN resolution procedure successfully.

NOTES

- The implementation of this callback can be asynchronous, letting the application return from this callback immediately without blocking until the STUN Server replies.
- While waiting for a response, the application might need to use the [RvIppAddressResolveReplyCB\(\)](#) callback to parse possible incoming STUN Server replies on SIP-related sockets.
- Address resolution due to STUN Server reply is done using [RvIppStunAddressResolveComplete\(\)](#).

SEE ALSO

[RvIppStunAddrData](#)

RvIppAddressResolveReplyCB()

DESCRIPTION

Callback prototype implemented by the STUN Client of the user application. Must be implemented if the user wants to receive replies from the STUN server to the SIP signaling socket. This callback is called for each message received by the SIP signaling socket. The callback tests if a current message is received from the STUN server, handles the message, and discards it.

SYNTAX

```
RvStatus RvIppAddressResolveReplyCB(  
    IN  RvChar*      buf,  
    IN  RvSize_t     size,  
    OUT RvBool*      bDiscardMsg);
```

PARAMETERS

buf

Received data.

size

Size of the received data in bytes.

OUTPUT

bDiscardMsg

RV_TRUE if this buffer should be discarded. This will be the case if the application identifies the incoming data as a STUN response.

RETURN VALUE

RV_OK on success. Any other return value will be treated as if the data in the buffer should not be discarded.

SEE ALSO

[RvIppStunAddressResolveStartCB\(\)](#)

STUN TYPE DEFINITIONS

This section describes the following STUN type definitions.

- [RvIppStunMgrParam](#)
- [RvIppStunAddrData](#)

RvIppStunMgrParam

DESCRIPTION

This structure contains the parameters for the configuration of the STUN module in the Multimedia Terminal Framework.

SYNTAX

```
typedef struct
{
    RvIppStunIsAddressResolveNeededCB isAddressResolveNeededCB;
    RvIppStunAddressResolveStartCB    addressResolveStartCB;
    RvIppAddressResolveReplyCB        addressResolveReplyCB;
} RvIppStunMgrParam;
```

PARAMETERS

[isAddressResolveNeededCB](#)

Callback function to check whether an address resolution is required or not.

[addressResolveStartCB](#)

Callback function requests for address resolution of private addresses to public ones.

[addressResolveReplyCB](#)

Callback function enabling to test if the buffer received on the SIP signaling socket is sent by the STUN Server. If the user's STUN Client always requests replies to be sent to its own socket then this parameter should be set NULL.

SEE ALSO

[RvIppStunMgrCreate\(\)](#)
[RvIppStunIsAddressResolveNeededCB\(\)](#)
[RvIppStunAddressResolveStartCB\(\)](#)
[RvIppAddressResolveReplyCB\(\)](#)

RvIppStunAddrData

DESCRIPTION

Address translation data structure used by the Multimedia Terminal Framework's STUN module. This struct is used by the Multimedia Terminal Framework to request the application to translate a private address into a public one. The application should view all of the fields in this struct as read-only fields, besides the outAddr field, which should be modified by the application to the public address.

SYNTAX

```
typedef struct
{
    RvIppSipAddressFieldType    type;
    RvAddress                    inAddr;
    RvAddress                    outAddr;
    RvInt16                     index;
    RvIppStunAddrDataStatus     status;
} RvIppStunAddrData;
```

PARAMETERS

type

The type of address. This indicates from which part of the outgoing SIP message the address comes.

inAddr

The address that was found in the outgoing SIP message. This is the address that requires translation.

outAddr

The address to which the inAddr field translates. The value of this field is given by the application, along with a call to RvIppStunAddressResolveComplete().

STUN Type Definitions

RvIppStunAddrData

index

The index of the address within the given SIP message header. This can be used by the application to distinguish between different addresses located in the same SIP header. The index value is 0-based.

status

The parameter is not for the use of the application. It is used internally by the Multimedia Terminal Framework.

19

TLS MODULE

WHAT'S IN THIS CHAPTER

This chapter describes TLS functions. TLS is a security mechanism that operates on top of TCP, enabling SIP entities to send and receive data in a secure and authenticated manner.

This chapter includes:

- TLS Functions
- TLS Callbacks
- TLS Type Definitions

TLS FUNCTIONS

This section describes the following TLS functions:

- `rvIppSipTlsInitConfig()`
- `rvIppSipTlsRegisterExtClbks()`

rvlppSipTlsInitConfig()

DESCRIPTION

Sets default TLS values into the SIP Phone Configuration structure.

SYNTAX

```
void rvlppSipTlsInitConfig(  
    OUT RvIppSipPhoneCfg* cfg);
```

OUTPUT PARAMETERS

[cfg](#)

Pointer to SIP Phone configuration structure. The transportTlsCfg field within this structure is overridden by this function.

RETURN VALUE

None.

SEE ALSO

[RvIppSipPhoneCfg](#)

[rvlppSipInitConfig\(\)](#)

TLS Functions

rvIppSipTlsRegisterExtCbks()

rvIppSipTlsRegisterExtCbks()

DESCRIPTION

Registers SIP TLS user callbacks. This function should be called before [rvIppSipStackInitialize\(\)](#).

SYNTAX

```
void rvIppSipTlsRegisterExtCbks(  
    IN RvIppSipTlsExtCbks* clbks);
```

PARAMETERS

[clbks](#)

Structure includes pointers to user implementations.

RETURN VALUE

None.

SEE ALSO

[RvIppSipTlsExtCbks](#)

[rvIppSipStackInitialize\(\)](#)

TLS CALLBACKS

This section describes the following TLS functions:

- `RvIppTlsGetBufferCB()`
- `RvIppSipTlsPostConnectionAssertionCB()`

RvIppTlsGetBufferCB()

DESCRIPTION

Callback prototype that retrieves a certificate or key buffer data from the application.

SYNTAX

```
RvBool RvIppTlsGetBufferCB(
    IN  RvIppTlsBufferType  bufferType,
    OUT RvChar*              tlsBuffer,
    OUT RvUInt32*            tlsBufferLen);
```

PARAMETERS

[bufferType](#)

Type of buffer to retrieve.

OUTPUT PARAMETERS

[tlsBuffer](#)

Placeholder for the buffer.

[tlsBufferLen](#)

Length of the buffer.

RETURN VALUE

RV_TRUE if the desired data was returned successfully. RV_FALSE if failures occurred.

SEE ALSO

[RvIppTlsBufferType](#)

RvIppSipTlsPostConnectionAssertionCB()

DESCRIPTION

This callback is used to override the Stack's default post connection assertion. Once a connection has completed the TLS handshake, it is necessary to make sure that the certificate presented was indeed issued for the address to which the connection was made. This assertion is done automatically by the Stack. If the application wants to override the assertion, the application can implement this callback.

SYNTAX

```
void RvIppSipTlsPostConnectionAssertionCB(  
    IN  RvSipTransportConnectionHandle    hConnection,  
    IN  RvSipTransportConnectionAppHandle hAppConnection,  
    IN  RvChar*                           strHostName,  
    IN  RvSipMsgHandle                    hMsg,  
    OUT RvBool*                           pbAsserted);
```

PARAMETERS

hConnection

The handle of the connection that changed TLS state.

hAppConnection

The application handle for the connection.

strHostName

A NULL terminated string, indicating the host name (IP/FQDN) to which the connection was meant to connect.

hMsg

A message if the connection was asserted against a message.

TLS Callbacks

RvIppSipTlsPostConnectionAssertionCB()

OUTPUT PARAMETERS

pbAsserted

Fill the Boolean with the result of your assertion.

RV_TRUE: Indicates that the connection was asserted successfully.

RV_FALSE: Indicates that the assertion failed. The connection will be terminated automatically.

RETURN VALUE

None.

TLS TYPE DEFINITIONS

This section describes the following TLS type definitions:

- RvIppTransportTlsCfg
- RvIppSipTlsExtCbks
- RvIppTlsBufferType

RvIppTransportTlsCfg

DESCRIPTION

This structure contains the parameters for the configuration of the TLS transport when that is used with the SIP Stack. It is part of the general configuration of the Multimedia Terminal Framework.

SYNTAX

```
typedef struct
{
    RvUInt16                stackTlsPort;
    RvString                stackTlsAddress;
    RvInt16                 stackNumOfTlsAddresses;
    RvBool                  tlsPostConnectAssertFlag;
    RvSipTransportTlsMethod tlsMethod;
    RvSipTransportPrivateKeyType privateKeyType;
    RvInt32                 certDepth;
} RvIppTransportTlsCfg;
```

PARAMETERS

stackTlsPort

Secure port used for TLS. Default: 5064.

tlsPostConnectAssertFlag

This parameter is not used by the TLS module. It may be used by the application for its own purposes. Default: RV_TRUE.

tlsMethod

The SSL methods that will be used in the application's TLS engine.
Default: RVSIP_TRANSPORT_TLS_METHOD_TLS_V1.

privateKeyType

Informs the engine of the type of private key it should use.
Default: RVSIP_TRANSPORT_PRIVATE_KEY_TYPE_RSA_KEY.

certDepth

Defines the depth that an engine will consider legal in a certificate chain to which it is presented before it is considered invalid. Default: 5.

SEE ALSO

[RvIppSipPhoneCfg](#)

RvIppSipTlsExtCbks

DESCRIPTION

This structure contains the callbacks associated with TLS that can be used for extensibility purposes by the application.

SYNTAX

```
typedef struct
{
    RvIppTlsGetBufferCB          tlsGetBufferF;
    RvIppSipTlsPostConnectionAssertionCB tlsPostConnectionAssertF;
} RvIppSipTlsExtCbks;
```

PARAMETERS

[tlsGetBufferF](#)

Retrieves a certificate or a key buffer.

[tlsPostConnectionAssertF](#)

Overrides the Multimedia Terminal Framework's assertion regarding the connection's certificate.

SEE ALSO

[rvIppSipTlsRegisterExtCbks\(\)](#)

[RvIppTlsGetBufferCB\(\)](#)

[RvIppSipTlsPostConnectionAssertionCB\(\)](#)

RvIppTlsBufferType

DESCRIPTION

The type of data required by the TLS module to complete its negotiation.

SYNTAX

```
typedef enum
{
    IPP_TLS_UNKNOWN_BUFFER_TYPE = -1,
    IPP_TLS_CLIENT_KEY_BUFFER,
    IPP_TLS_CA_BUFFER,
    IPP_TLS_SERVER_KEY_BUFFER,
    IPP_TLS_SERVER_CA_BUFFER
} RvIppTlsBufferType;
```

PARAMETERS

[IPP_TLS_UNKNOWN_BUFFER_TYPE = -1,](#)

Unknown buffer type.

SEE ALSO

[RvIppTlsGetBufferCB\(\)](#)

TLS Type Definitions

RvIppTlsBufferType

PART 3: USER CALLBACKS

20

USER CALLBACKS

WHAT'S IN THIS CHAPTER

This chapter contains functions and callback functions used by the MDM to support Call Control. For more information about callbacks, see the Multimedia Terminal Framework *Programmer Guide*.

This chapter includes:

- [User Callbacks](#)

USER CALLBACKS

This section includes:

- RvMdmTermMapDialStringToAddressCB()
- RvMdmTermMatchDialStringCB()
- RvMdmTermCreateMediaCB()
- RvMdmTermDestroyMediaCB()
- RvMdmTermModifyMediaCB()
- RvMdmTermPlaySignalCB()
- RvMdmTermSetStateCB()
- RvMdmTermStartSignalCB()
- RvMdmTermStopSignalCB()
- RvMdmTermMgrConnectCB()
- RvMdmTermMgrDeleteEphTermCB()
- RvMdmTermMgrDisconnectCB()
- RvMdmTermMgrSelectTerminationCB()
- RvMdmTermModifyMediaCompletedCB()

RvMdmTermMapDialStringToAddressCB()

DESCRIPTION

Maps dialed digits to a destination IP address or DNS name.

This callback is invoked when all digits have been collected and require mapping to a destination address.

SYNTAX

```
RvBool RvMdmTermMapDialStringToAddressCB(  
    IN RvMdmTerm*      term,  
    IN const RvChar*    dialString,  
    OUT RvChar*         address);
```

PARAMETERS

term

A pointer to the Termination where the dialing was done.

dialString

The collected dial string.

OUTPUT PARAMETERS

address

Must contain the destination address on return.

The length of the returned string must be at most 64 bytes—otherwise, a buffer overrun will occur.

For SIP, the address format of this string is:

<scheme>:<username>@<ip address/domain name>:<port>

RETURN VALUES

RV_TRUE if the application successfully mapped the given dial string into a destination address.

User Callbacks

RvMdmTermMapDialStringToAddressCB()

RV_FALSE if the application failed to do the mapping. In such a case, Multimedia Terminal Framework will try to resolve the address on its own, through a Registrar for SIP.

SEE ALSO

[RvMdmTerm Module](#)

[rvMdmTermClassRegisterMapDialStringToAddressCB\(\)](#)

RvMdmTermMatchDialStringCB()

DESCRIPTION

This callback is called for each new digit dialed by the user. Is it used by the user to indicate when digit collection is completed or cannot be done any further.

SYNTAX

```
RvMdmDigitMapMatchType RvMdmTermMatchDialStringCB(  
    IN RvMdmTerm*      term,  
    IN const RvChar*    dialString,  
    OUT RvUnit*         timerDuration);
```

PARAMETERS

[term](#)

A pointer to the Termination where the dialing was done.

[dialString](#)

The dial string collected up to now (the digits dialed).

OUTPUT PARAMETERS

[timerDuration](#)

In this field, a value is returned to indicate how long to wait for an additional digit. If the timer expires, a completion event will be generated. Set to zero to keep the timer from starting. The timer duration should be given in seconds.

RETURN VALUES

The status of the current dial string collection mapping to complete string.

SEE ALSO

[RvMdmTerm Module](#)
[RvMdmDigitMapMatchType](#)
[RvMdmTermMatchDialStringCB\(\)](#)

RvMdmTermCreateMediaCB()

DESCRIPTION

This callback is invoked by the Multimedia Terminal Framework when there is a need to create a new media stream on a Termination.

SYNTAX

```
RvBool RvMdmTermCreateMediaCB (
    IN      RvMdmTerm*          term,
    IN      RvMdmMediaStream*   media,
    INOUT   RvMdmMediaStreamDescr* streamDescr,
    OUT     RvMdmError*         mdmError) ;
```

PARAMETERS

term

The RTP Termination.

media

Pointer to the new media object.

streamDescr

This structure includes all the media capabilities of the local party and may include capabilities of the remote party. The application can use this data to choose the preferred codecs and parameters with which to open the media stream.

After choosing the codecs and media parameters with which the media stream is opened, the application should clear this structure and add only the chosen parameters.

OUTPUT PARAMETERS

mdmError (Optional)

Used to set error information.

RETURN VALUES

Returns RV_FALSE if it fails. In this case, mdmError can be set. If mdmError is not set, a default error value will be assigned.

REMARKS

StreamDescr must contain only the selected values. The application must fill underspecified values and choose from overspecified values. The application must call the following:

- [rvMdmMediaStreamDescrReportLocalDescr\(\)](#)
- [rvMdmMediaStreamDescrReportRemoteDescr\(\)](#)
- and/or [rvMdmMediaStreamDescrReportControlParameters\(\)](#) in the streamDescr to indicate which fields were modified and have to be reported back to the Media Gateway Controller.

The function [rvMdmMediaStreamSetUserData\(\)](#) can be called to associate the media with user data.

SEE ALSO

[RvMdmTerm Module](#)

[RvMdmMediaStreamDescr Module](#)

[rvMdmTermClassRegisterCreateMediaCB\(\)](#)

RvMdmTermDestroyMediaCB()

DESCRIPTION

Functions that follow this template are called to close a media stream on a Termination and release resources.

SYNTAX

```
RvBool RvMdmTermDestroyMediaCB(  
    IN  RvMdmTerm          term,  
    IN  RvMdmMediaStream   media,  
    OUT RvMdmError          mdmError);
```

PARAMETERS

term

The Termination.

mediaStream

User identifier of the media.

mdmError (Optional)

Used to set error information.

RETURN VALUES

Returns rvFalse if it fails. In this case, mdmError can be set. If mdmError is not set, a default error value will be assigned.

REMARKS

Call rvMdmMediaStreamGetUserData() to get the user data associated with media.

RvMdmTermModifyMediaCB()

DESCRIPTION

This function is called to modify the characteristics of a media stream on a Termination.

SYNTAX

```
RvBool RvMdmTermModifyMediaCB (
    IN      RvMdmTerm          term,
    IN      RvMdmMediaStream   media,
    INOUT   RvMdmMediaStreamDescr streamDescr,
    IN      RvMdmError         mediaError);
```

PARAMETERS

term

The Termination.

mediaStream

User identifier of the media stream.

streamDescr (Input/Output)

Properties of the media.

mediaError (Optional)

Use this paramter to set error information.

RETURN VALUES

Returns rvFalse if it fails. In this case mediaError can be set. If mediaError is not set, a default error value will be assigned.

StreamDescr must contain only the selected values. The application must fill underspecified values and choose from overspecified values. The application must call the following:

- [rvMdmMediaStreamDescrReportLocalDescr\(\)](#)
- [rvMdmMediaStreamDescrReportRemoteDescr\(\)](#)

User Callbacks

RvMdmTermModifyMediaCB()

- and/or [rvMdmMediaStreamDescrReportControlParameters\(\)](#) in the streamDescr to indicate which fields were modified and have to be reported back to the Media Gateway Controller.

RvMdmTermPlaySignalCB()

DESCRIPTION

This callback function is called to play a signal on a Termination. This signal is played to completion by the application and not stopped by the Termination Manager.

SYNTAX

```
RvBool RvMdmTermPlaySignalCB(  
    IN  RvMdmTerm*    term,  
    IN  RvMdmSignal*  s,  
    IN  RvBool         reportCompletion,  
    OUT RvMdmError*    mdmError);
```

PARAMETERS

term

The Termination.

s

The signal.

reportCompletion

If RV_TRUE, the application must call rvMdmTermSignalCompleted() when the signal ends.

OUTPUT PARAMETERS

mdmError

Use to set error information.

RETURN VALUES

Return RV_FALSE if it fails. In this case mdmError can be set. If mdmError is not set, a default error value will be assigned.

User Callbacks

RvMdmTermPlaySignalCB()

REMARKS

Call [rvMdmTermGetUserData\(\)](#) to get the user data associated with term.

SEE ALSO

[RvMdmTerm Module](#)

[RvMdmSignal Module](#)

[rvMdmTermSignalCompleted\(\)](#)

[rvMdmTermClassRegisterPlaySignalCB\(\)](#)

RvMdmTermSetStateCB()

DESCRIPTION

This callback function is called to update the state of the Termination (stream-independent properties).

SYNTAX

```
RvBool RvMdmTermSetStateCB(  
    IN      RvMdmTerm          term,  
    INOUT   RvMdmParameterList stateParameters,  
    OUT     RvMdmError          mdmError);
```

PARAMETERS

term

The Termination.

stateParameters (Input/Output)

List of state properties.

mdmError (Optional)

Used to set error information.

RETURN VALUES

Returns "rvFalse" if it fails. In this case mdmError can be set. If mdmError is not set, a default error value will be assigned.

StateParameters must contain only the selected values. The application must fill underspecified values and choose from overspecified values.

RvMdmTermStartSignalCB()

DESCRIPTION

This callback function is used to start a signal in a Termination of this class. This signal is played by the application until explicitly stopped by the Termination Manager, which calls the callback function [RvMdmTermStopSignalCB\(\)](#).

SYNTAX

```
RvBool RvMdmTermStartSignalCB(  
    IN  RvMdmTerm*      term,  
    IN  RvMdmSignal*    s,  
    OUT RvMdmError*      mdmError);
```

PARAMETERS

[term](#)

The Termination.

[s](#)

The signal.

OUTPUT PARAMETERS

[mdmError](#)

Used to set error information.

RETURN VALUES

Returns `RV_FALSE` if it fails. In this case `mdmError` can be set. If `mdmError` is not set, a default error value will be assigned.

REMARKS

Call the function [rvMdmTermGetUserData\(\)](#) to get the user data associated with `term`.

SEE ALSO[RvMdmTerm Module](#)[RvMdmSignal Module](#)[RvMdmTermStopSignalCB\(\)](#)[rvMdmTermClassRegisterStartSignalCB\(\)](#)

RvMdmTermStopSignalCB()

DESCRIPTION

This callback function is called by the Termination Manager to explicitly stop a signal in a Termination.

SYNTAX

```
RvBool RvMdmTermStopSignalCB(  
    IN  RvMdmTerm*      term,  
    IN  RvMdmSignal*    s,  
    OUT RvMdmError*     mdmError);
```

PARAMETERS

term

The Termination.

s

The signal.

mdmError (Optional)

Used to set error information.

RETURN VALUES

Returns "rvFalse" if it fails. In this case mdmError can be set. If mdmError is not set, a default error value will be assigned.

REMARKS

Call the function [rvMdmTermGetUserData\(\)](#) to get the user data associated with term.

RvMdmTermMgrConnectCB()

DESCRIPTION

Called to connect a media stream in one Termination to a media stream in another Termination.

SYNTAX

```
RvBool (RvMdmTermMgrConnectCB) (  
    IN  RvMdmTermMgr      mgr,  
    IN  RvMdmTerm         source,  
    IN  RvMdmMediaStream  m1,  
    IN  RvMdmTerm         target,  
    IN  RvMdmMediaStream  m2,  
    IN  RvMdmStreamDirection direction,  
    OUT RvMdmError        mdmError);
```

PARAMETERS

mgr

The Termination Manager.

source

First Termination.

m1

Media stream in first Termination.

target

Second Termination.

m2

Media stream in second Termination.

User Callbacks

RvMdmTermMgrConnectCB()

direction

Direction of the media flow (RV_MDMSTREAMDIRECTION_BOTHWAYS or RV_MDMSTREAMDIRECTION_SOURCE2TARGET).

mdmError (Optional)

Used to set error information.

RETURN VALUES

Returns "rvFalse" if it fails. In this case mdmError can be set. If mdmError is not set, a default error value will be assigned.

RvMdmTermMgrDeleteEphTermCB()

DESCRIPTION

Called to notify the application that the resources used for an ephemeral Termination can be released.

Returning from this callback function the Termination is no longer registered with the Termination Manager.

SYNTAX

```
void RvMdmTermMgrDeleteEphTermCB(  
    IN RvMdmTermMgr*    mgr,  
    IN RvMdmTerm*       ephTerm);
```

PARAMETERS

mgr

The Termination Manager.

ephTerm

The ephemeral Termination.

RETURN VALUES

None.

SEE ALSO

[rvMdmTermMgrRegisterDeleteEphTermCB\(\)](#)

RvMdmTermMgrDisconnectCB()

DESCRIPTION

Called to disconnect a media stream in one Termination from a media stream in another Termination.

SYNTAX

```
RvBool RvMdmTermMgrDisconnectCB (
    IN  RvMdmTermMgr*      mgr,
    IN  RvMdmTerm*         source,
    IN  RvMdmMediaStream*  m1,
    IN  RvMdmTerm*         target,
    IN  RvMdmMediaStream*  m2,
    OUT RvMdmError*        mdmError);
```

PARAMETERS

mgr

The Termination Manager.

source

First Termination.

m1

Media stream in first Termination.

target

Second Termination.

m2

Media stream in second Termination.

mdmError (Optional)

Used to set error information.

RETURN VALUES

Returns RV_FALSE if it fails. In this case mdmError can be set. If mdmError is not set, a default error value will be assigned.

SEE ALSO

[rvMdmTermMgrRegisterDisconnectCB\(\)](#)

User Callbacks

RvMdmTermMgrSelectTerminationCB()

RvMdmTermMgrSelectTerminationCB()

DESCRIPTION

Called to select a Termination, either physical or ephemeral.

SYNTAX

```
RvMdmTerm* RvMdmTermMgrSelectTerminationCB (  
    IN RvMdmTermMgr*    mgr,  
    IN RvMdmTerm*       tempTerm) ;
```

PARAMETERS

mgr

The Termination Manager.

tempTerm

A temporary Termination that can be used by the application to get information about the Termination to be selected, such as media requirements, partial name, etc.

RETURN VALUES

None.

REMARKS

To select an "ephemeral" Termination, first register it by calling [rvMdmTermMgrRegisterEphemeralTermination\(\)](#).

SEE ALSO

[rvMdmTermMgrRegisterSelectTermCB\(\)](#)

RvMdmTermModifyMediaCompletedCB()

DESCRIPTION

This callback is called when the process of dynamic media change has come to an end. The process is initiated by the local user by calling the function [rvMdmTermModifyMedia\(\)](#).

SYNTAX

```
typedef void (*RvMdmTermModifyMediaCompletedCB) (
    IN   RvMdmTerm*           term,
    IN   RvBool               status,
    IN   RvMdmMediaDescriptor* media,
    OUT  RvMdmMediaStreamDescr* streamDescr,
    IN   RvMdmTermReasonModifyMedia reason);
```

PARAMETERS

[term](#)

A pointer to the terminal.

[status](#)

Indicates the result of the process: True if successful, False if not.

[media](#)

Includes the SDP message as it was received in the 200 OK reply from the remote party.

[reason](#)

May be set to one of the following values:

- **RV_MDMTERMREASON_UNKNOWN**—the reason for failure is unknown.
- **RV_MDMTERMREASON_SUCCESS**—the process has succeeded.
- **RV_MDMTERMREASON_IN_PROCESS**—a preceding modify media process that has not yet ended.

User Callbacks

RvMdmTermModifyMediaCompletedCB()

- RV_MDMTERMREASON_REMOTE_REJECTED—the remote party has rejected the Re-Invite sent by the local party.
- RV_MDMTERMREASON_LOCAL_FAILED—local failure (for example, sending the signaling message has failed).

OUTPUT PARAMETERS

streamDescr

The user application should return the SDP message as it was opened (in the event that it changed after the remote party reply was received). It will be stored in the Multimedia Terminal Framework for later media operations.

RETURN VALUES

None.

PART 4: EXTENSIBILITY

21

SIP CONTROL EXTENSION

WHAT'S IN THIS CHAPTER

This chapter includes SIP Stack extension functions.

This chapter includes:

- [SIP Control Extension Functions](#)
- [SIP Control Extension Type Definitions](#)

SIP CONTROL EXTENSION FUNCTIONS

This section includes:

- `rvIppSipControlGetMdmTerminal()`
- `RvIppSipExtCbks`

rvIppSipControlGetMdmTerminal()

DESCRIPTION

Returns a handle to Mdm Terminal.

SYNTAX

```
RvIppTerminalHandle rvIppSipControlGetMdmTerminal(  
    IN RvSipAppCallLegHandle    hAppCallLeg);
```

PARAMETERS

hAppCallLeg

A handle to the application call leg handle.

RETURN VALUE

Handle to Mdm Terminal.

SEE ALSO

[RvIppTerminalHandle](#)

RvIppSipExtCbks

DESCRIPTION

Structure containing the SIP extension callbacks to be implemented by the user application. The `userData` parameter of this struct is provided whenever one of the callbacks is invoked.

SYNTAX

```
typedef struct
{
    RvIppSipStackConfigCB stackConfigF;
    RvIppSipRegisterStackEventsCB registerStackEventsF;
    RvIppSipPreCallLegCreatedIncomingCB preCallLegCreatedIncomingF;
    RvIppSipPostCallLegCreatedIncomingCB postCallLegCreatedIncomingF;
    RvIppSipPreCallLegCreatedOutgoingCB preCallLegCreatedOutgoingF;
    RvIppSipPostCallLegCreatedOutgoingCB postCallLegCreatedOutgoingF;
    RvIppSipPreStateChangedCB preStateChangedF;
    RvIppSipPreMsgToSendCB preMsgToSendF;
    RvIppSipPostMsgToSendCB postMsgToSendF;
    RvIppSipPreMsgReceivedCB preMsgReceivedF;
    RvIppSipPostMsgReceivedCB postMsgReceivedF;
    RvIppSipPreRegClientStateChangedCB
        PreRegClientStateChangedF;
    void* userData;
} RvIppSipExtCbks;
```

PARAMETERS

[userData](#)

The user-related data that will be used when invoking any of the callbacks in this structure.

SEE ALSO

[rvIppSipRegisterExtCbks](#)
[RvIppSipStackConfigCB\(\)](#)
[RvIppSipRegisterStackEventsCB\(\)](#)
[RvIppSipPreCallLegCreatedIncomingCB\(\)](#)
[RvIppSipPostCallLegCreatedIncomingCB\(\)](#)

RvIppSipPreCallLegCreatedOutgoingCB()
RvIppSipPostCallLegCreatedOutgoingCB()
RvIppSipPreStateChangedCB()
RvIppSipPreMsgToSendCB()
RvIppSipPostMsgToSendCB()
RvIppSipPreMsgReceivedCB()
RvIppSipPostMsgReceivedCB()
RvIppSipPreRegClientStateChangedCB()

SIP CONTROL EXTENSION TYPE DEFINITIONS

This section includes:

- RvIppSipStackConfigCB()
- RvIppSipRegisterStackEventsCB()
- RvIppSipPreCallLegCreatedIncomingCB()
- RvIppSipPostCallLegCreatedIncomingCB()
- RvIppSipPreCallLegCreatedOutgoingCB()
- RvIppSipPostCallLegCreatedOutgoingCB()
- RvIppSipPreStateChangedCB()
- RvIppSipPostStateChangedCB()
- RvIppSipPreMsgToSendCB()
- RvIppSipPostMsgToSendCB()
- RvIppSipPreMsgReceivedCB()
- RvIppSipPostMsgReceivedCB()
- RvIppSipPreRegClientStateChangedCB()
- RvIppProviderHandle
- RvIppTerminalHandle
- RvIppConnectionHandle

RvIppSipStackConfigCB()

DESCRIPTION

Prototype for callback to configure the SIP Stack.

When this function is called, pStackCfg is filled with default values that can be changed by the user. After this function returns, validation checking is done on the values by both the Multimedia Terminal Framework and the SIP Stack. Invalid values will be changed to default values, and unsupported parameters will be ignored.

SYNTAX

```
void RvIppSipStackConfigCB(  
    IN RvSipStackCfg* pStackCfg);
```

PARAMETERS

[pStackCfg](#)

Pointer to the SIP Stack configuration structure.

RETURN VALUE

None.

REMARKS

This callback is invoked from within the call to [rvIppSipStackInitialize\(\)](#).

SEE ALSO

[rvIppSipStackInitialize\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipRegisterStackEventsCB()

DESCRIPTION

Prototype for registering callbacks to the SIP Stack.

This function enables the user to listen for Stack events where the Multimedia Terminal Framework does not.

SYNTAX

```
void RvIppSipRegisterStackEventsCB(  
    IN RvIppSipControlHandle    sipMgrHndl);
```

PARAMETERS

[sipMgrHndl](#)

Pointer to the SipControl instance.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipExtClbks](#)

RvIppSipPreCallLegCreatedIncomingCB()

DESCRIPTION

This callback is called before the Multimedia Terminal Framework processes an incoming call. The application can modify the call's information or halt the call's creation altogether in this callback.

SYNTAX

```
RvBool RvIppSipPreCallLegCreatedIncomingCB (
    IN  RvIppSipControlHandle    sipMgrHndl,
    IN  RvSipCallLegHandle       hCallLeg,
    OUT RvSipAppCallLegHandle*   phAppCallLeg,
    IN  void*                    userData) ;
```

PARAMETERS

sipMgrHndl

Pointer to the sipMgr handle.

hCallLeg

Call-leg handle.

phAppCallLeg

The application's call-leg handle. If RV_FALSE is returned from this callback, this parameter should be provided by the application. Otherwise, it is ignored.

userData

The user-specified data given in the [RvIppSipExtCbks](#) struct.

RETURN VALUE

RV_FALSE if you do not want the IP Phone to perform its default processing.
RV_TRUE to continue as usual.

If RV_FALSE is returned, the application is expected to take care of this call directly on top of the SIP Stack.

SIP Control Extension Type Definitions

`RvIppSipPreCallLegCreatedIncomingCB()`

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPostCallLegCreateIncomingCallCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPostCallLegCreatedIncomingCB()

DESCRIPTION

This callback is called just after an incoming call in the SIP Stack begins to be handled by the Multimedia Terminal Framework.

SYNTAX

```
void RvIppSipPostCallLegCreatedIncomingCB(  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle       hCallLeg,  
    IN RvSipAppCallLegHandle*   phAppCallLeg,  
    IN void*                    userData);
```

PARAMETERS

[sipMgrHndl](#)

Pointer to sipMgr handle.

[hCallLeg](#)

Call-leg handle.

[phAppCallLeg](#)

Application call-leg handle.

[userData](#)

The user-specified data given in the [RvIppSipExtCbks](#) struct.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPreCallLegCreatedIncomingCallCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPreCallLegCreatedOutgoingCB()

DESCRIPTION

This callback is called before Multimedia Terminal Framework build headers for the call-leg object, and will use "to" and "from" values returned by the user to build TO and FROM headers in outgoing SIP calls.

SYNTAX

```
void RvIppSipPreCallLegCreatedOutgoingCB(  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle      hCallLeg,  
    IN RvChar*                  to,  
    IN RvChar*                  from,  
    IN void*                    userData);
```

PARAMETERS

sipMgrHndl

Pointer to sipMgr handle.

hCallLeg

Call-leg handle.

to

Destination address.

from

Source address.

userData

The userData provided by the application in its [RvIppSipExtCbks](#) struct.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPostCallLegCreatedOutgoingCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPostCallLegCreatedOutgoingCB()

DESCRIPTION

This callback is called after the Multimedia Terminal Framework has built headers in the call-leg object, and before dialing out on a call. The user can change any of the headers in the call-leg object.

SYNTAX

```
void RvIppSipPostCallLegCreatedOutgoingCB (  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle       hCallLeg,  
    IN void*                    userData);
```

PARAMETERS

sipMgrHndl

Pointer to the sipMgr handle.

hCallLeg

Call-leg handle.

userData

The userData provided by the application in its [RvIppSipExtClbks](#) struct.

RETURN VALUE

None.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPreCallLegCreatedOutgoingCB\(\)](#)

[RvIppSipExtClbks](#)

RvIppSipPreStateChangedCB()

DESCRIPTION

This callback is called when the SIP Stack event RvSipCallLegStateChangedEv() is invoked, before the Multimedia Terminal Framework handles a SipCallLeg state machine change.

SYNTAX

```
RvBool RvIppSipPreStateChangedCB (
    IN RvIppSipControlHandle      sipMgrHndl,
    IN RvSipCallLegHandle         hCallLeg,
    IN RvSipAppCallLegHandle      hAppCallLeg,
    IN RvSipCallLegState          eState,
    IN RvSipCallLegStateChangeReason eReason,
    IN void*                      userData) ;
```

PARAMETERS

sipMgrHndl

Pointer to the sipMgr handle.

hCallLeg

Call-leg handle.

phAppCallLeg

Application call-leg handle.

eState

CallLeg state.

eReason

State change reason.

userData

The userData provided by the application in its [RvIppSipExtCbks](#) structure.

SIP Control Extension Type Definitions

RvIppSipPreStateChangedCB()

RETURN VALUE

RV_FALSE if the application does not want the Multimedia Terminal Framework to perform its default processing. RV_TRUE to continue as usual.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipExtCbks](#)

RvIppSipPostStateChangedCB()

DESCRIPTION

This callback is called when the SIP Stack event RvSipCallLegStateChangedEv() is invoked, after the Multimedia Terminal Framework handles a SipCallLeg state machine change.

SYNTAX

```
void RvIppSipPreStateChangedCB (
    IN RvIppSipControlHandle      sipMgrHndl,
    IN RvSipCallLegHandle         hCallLeg,
    IN RvSipAppCallLegHandle      hAppCallLeg,
    IN RvSipCallLegState          eState,
    IN RvSipCallLegStateChangeReason eReason,
    IN void*                      userData) ;
```

PARAMETERS

[sipMgrHndl](#)

Pointer to the sipMgr handle.

[sipMgrHndl](#)

Call-leg handle.

[sipMgrHndl](#)

Application call-leg handle.

[sipMgrHndl](#)

CallLeg state.

[sipMgrHndl](#)

State-change reason.

[sipMgrHndl](#)

The userData provided by the application in its [RvIppSipExtCbks](#) structure.

SIP Control Extension Type Definitions

RvIppSipPostStateChangedCB()

RETURN VALUE

None.

NOTE

When eState equals RVSIP_CALL_LEG_STATE_DISCONNECTED, hAppCallLeg is no longer valid and will be set to NULL.

RvIppSipPreMsgToSendCB()

DESCRIPTION

This callback is invoked before the Multimedia Terminal Framework modifies the message that is about to be sent. Enables the application to change the outgoing SIP message.

SYNTAX

```
RvBool RvIppSipPreMsgToSendCB(  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle       hCallLeg,  
    IN RvSipAppCallLegHandle    hAppCallLeg,  
    IN RvSipMsgHandle           hMsg,  
    IN void*                    userData);
```

PARAMETERS

sipMgrHndl

Pointer to the sipMgr handle.

hCallLeg

Call-leg handle.

phAppCallLeg

Application call-leg handle.

hMsg

SIP message handle.

userData

The userData provided by the application in its [RvIppSipExtClibks](#) struct.

RETURN VALUE

RV_FALSE if the application does not want the Multimedia Terminal Framework to perform its default processing. RV_TRUE to continue as usual.

SIP Control Extension Type Definitions

`RvIppSipPreMsgToSendCB()`

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPostMsgToSendCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPostMsgToSendCB()

DESCRIPTION

This callback is invoked before the Multimedia Terminal Framework modifies the message that is about to be sent. Enables the application to change the outgoing SIP message.

SYNTAX

```
void RvIppSipPostMsgToSendCB(  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle       hCallLeg,  
    IN RvSipAppCallLegHandle    hAppCallLeg,  
    IN RvSipMsgHandle           hMsg,  
    IN void*                    userData);
```

PARAMETERS

sipMgrHndl

Pointer to the sipMgr handle.

hCallLeg

Call-leg handle.

phAppCallLeg

Application call-leg handle.

hMsg

SIP message handle.

userData

The userData provided by the application in its [RvIppSipExtCbks](#) struct.

RETURN VALUE

None.

SIP Control Extension Type Definitions

`RvIppSipPostMsgToSendCB()`

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPreMsgToSendCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPreMsgReceivedCB()

DESCRIPTION

This callback is invoked before the Multimedia Terminal Framework modifies the message received. It enables the application to change the incoming SIP message. It enables the application to change incoming SIP message and to decide whether messages should be ignored or processed by the SIP Stack or the Multimedia Terminal Framework.

SYNTAX

```
typedef RvMtfMsgProcessType (*RvIppSipPreMsgReceivedCB) (  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle      hCallLeg,  
    IN RvSipAppCallLegHandle    hAppCallLeg,  
    IN RvSipMsgHandle          hMsg,  
    IN void*                    userData);
```

PARAMETERS

sipMgrHndl

Pointer to the sipMgr handle.

hCallLeg

Call-leg handle.

phAppCallLeg

Application call-leg handle.

hMsg

SIP message handle.

userData

The userData provided by the application in its [RvIppSipExtCbks](#) struct.

SIP Control Extension Type Definitions

RvIppSipPreMsgReceivedCB()

RETURN VALUE

RvMtfMsgProcessType can be set to one of the following values:

RV_MTF_IGNORE_BY_STACK

This value indicates to both the SIP Stack and the Multimedia Terminal Framework to ignore the message. When this value is returned, the callback RvIppSipPreCallLegCreatedIncomingCB() should return False as well.

Otherwise, a memory leak will occur (when RvIppSipPreCallLegCreatedIncomingCB() is called, resources—of the SIP connection—will be allocated but not released).

RV_MTF_IGNORE_BY_MTF

This value indicates to the Multimedia Terminal Framework to ignore the message, but the message will still be processed by the SIP Stack.

RV_MTF_DONT_IGNORE

This value indicates to both the SIP Stack and the Multimedia Terminal Framework to process the message.

SEE ALSO

[RvIppSipPostMsgReceivedCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPostMsgReceivedCB()

DESCRIPTION

This callback is invoked before the Multimedia Terminal Framework modifies the received message. It enables the application to change the incoming SIP message.

SYNTAX

```
void RvIppSipPostMsgReceivedCB(  
    IN RvIppSipControlHandle    sipMgrHndl,  
    IN RvSipCallLegHandle       hCallLeg,  
    IN RvSipAppCallLegHandle    hAppCallLeg,  
    IN RvSipMsgHandle           hMsg,  
    IN void*                    userData);
```

PARAMETERS

sipMgrHndl

Pointer to sipMgr handle.

hCallLeg

Call-leg handle.

phAppCallLeg

Application call-leg handle.

hMsg

SIP message handle.

userData

The userData provided by the application in its [RvIppSipExtCbks](#) struct.

RETURN VALUE

None.

SIP Control Extension Type Definitions

`RvIppSipPostMsgReceivedCB()`

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppSipPreMsgReceivedCB\(\)](#)

[RvIppSipExtCbks](#)

RvIppSipPreRegClientStateChangedCB()

DESCRIPTION

This callback is invoked before the Multimedia Terminal Framework handles registration state changes of the SIP User Agent.

SYNTAX

```
typedef RvIppSipPreRegClientStateChangedCB (  
    IN RvIppSipControlHandle          sipMgrHndl,  
    IN RvSipRegClientHandle           hRegClient,  
    IN RvIppTerminalHandle            mdmTerminalHndl,  
    IN RvSipRegClientState            eState,  
    IN RvSipRegClientStateChangeReason eReason,  
    IN void*                          userData) ;
```

PARAMETERS

sipMgrHndl

Pointer to sipMgr handle.

hRegClient

SIP registration client instance.

mdmTerminalHndl

The terminal handle used.

eState

SIP registration client state.

eReason

Reason for the state changed that occurred.

userData

The userData provided by the application in its [RvIppSipExtCbks](#) struct.

SIP Control Extension Type Definitions

RvIppSipPreRegClientStateChangedCB()

RETURN VALUE

RV_FALSE if the application does not want the Multimedia Terminal Framework to perform any default processing due to the state change.
RV_TRUE to continue as usual.

SEE ALSO

[RvIppSipControlHandle](#)

[RvIppTerminalHandle](#)

[RvIppSipExtCbks](#)

RvIppProviderHandle

DESCRIPTION

The provider handle in the Multimedia Terminal Framework. This handle associates a group of terminations ([RvIppTerminalHandle](#)) into a given Multimedia Terminal Framework instance.

SYNTAX

```
RV_DECLARE_HANDLE (RvIppProviderHandle);
```

RvIppTerminalHandle

DESCRIPTION

A termination handle in the Multimedia Terminal Framework. This handle indicates a specific terminal within the Multimedia Terminal Framework. When the Multimedia Terminal Framework is used as an IAD, there may be several terminals/terminations, each able to hold several different connections. When the Multimedia Terminal Framework is used as a client/terminal, there may be a single terminal/termination that may hold several different connections. Connections on such terminations are associated to [RvIppConnectionHandle](#).

SYNTAX

```
RV_DECLARE_HANDLE (RvIppTerminalHandle);
```

RvIppConnectionHandle

DESCRIPTION

A connection handle in the Multimedia Terminal Framework. This handle indicates a specific, temporary connection/call within the Multimedia Terminal Framework on a specific termination. Connections belong to specific terminations through [RvIppTerminalHandle](#).

SYNTAX

```
RV_DECLARE_HANDLE (RvIppConnectionHandle);
```

SIP Control Extension Type Definitions

RvIppConnectionHandle

22

MDM CONTROL EXTENSION

WHAT'S IN THIS CHAPTER

This chapter includes control internal extension functions for the Media Device Manager API.

This chapter includes:

- [MDM Control Extension Functions](#)
- [MDM Control Extension Provider API](#)
- [MDM Control Extension Terminal API](#)
- [MDM Control Extension Connection API](#)
- [MDM Control Extension Type Definitions](#)

MDM CONTROL EXTENSION FUNCTIONS

This section includes:

- `rvIppMdmRegisterExtCbks()`

rvIppMdmRegisterExtClbks()

DESCRIPTION

This function is used to register the MDM Control's extension callbacks. Most of the callbacks have an additional userData parameter, which will get the value indicated in the clbks struct.

SYNTAX

```
void rvIppMdmRegisterExtClbks(  
    IN RvIppMdmExtClbks*    clbks);
```

PARAMETERS

[clbks](#)

The callbacks and userData to register.

RETURN VALUE

None.

SEE ALSO

[rvIppMdmExtClbks](#)

MDM CONTROL EXTENSION PROVIDER API

This section includes:

- `rvIppMdmProviderGetDialToneDuration()`
- `rvIppMdmProviderGetDisplayData()`
- `rvIppMdmProviderFindTerminalByTermId()`
- `rvIppMdmProviderFindTerminalByAddress()`
- `rvIppMdmProviderFindTerminalByNumber()`
- `rvIppMdmProviderFindAnyTerminal()`

rvIppMdmProviderGetDialToneDuration()

DESCRIPTION

Retrieves Dial-Tone Duration configured value.

SYNTAX

```
RvUInt32 rvIppMdmProviderGetDialToneDuration(  
    IN RvIppProviderHandle providerHndl);
```

PARAMETERS

[providerHndl](#)

Provider handle.

RETURN VALUE

Number of seconds to play dial tone.

SEE ALSO

[RvIppProviderHandle](#)

rvIppMdmProviderGetDisplayData()

DESCRIPTION

Retrieves the display data.

SYNTAX

```
void* rvIppMdmProviderGetDisplayData(  
    IN RvIppProviderHandle providerHndl);
```

PARAMETERS

providerHndl

Provider handle.

RETURN VALUE

Pointer to display data structure.

SEE ALSO

[RvIppProviderHandle](#)

[RvIppTerminalHandle](#)

[rvIppMdmProviderFindTerminalByAddress\(\)](#)

[rvIppMdmProviderFindTerminalByNumber\(\)](#)

rvIppMdmProviderFindTerminalByTermId()

DESCRIPTION

Retrieves the terminal handle based on terminal ID.

SYNTAX

```
RvIppTerminalHandle rvIppMdmProviderFindTerminalByTermId(  
    IN RvIppProviderHandle    providerHndl,  
    IN const RvChar*          termId);
```

PARAMETERS

providerHndl

Provider handle.

termId

Terminal ID.

RETURN VALUE

Terminal handle.

rvIppMdmProviderFindTerminalByAddress()

DESCRIPTION

Retrieves the terminal handle based on the IP address.

SYNTAX

```
RvIppTerminalHandle rvIppMdmProviderFindTerminalByAddress(  
    IN RvIppProviderHandle    providerHndl,  
    IN const RvChar*          address);
```

PARAMETERS

providerHndl

Provider handle.

address

IP address.

RETURN VALUE

Terminal handle.

SEE ALSO

[RvIppProviderHandle](#)

[RvIppTerminalHandle](#)

[rvIppMdmProviderFindTerminalByTermId\(\)](#)

[rvIppMdmProviderFindTerminalByNumber\(\)](#)

rvIppMdmProviderFindAnyTerminal()

DESCRIPTION

Retrieves the first terminal handle that exists under this provider.

SYNTAX

```
RvIppTerminalHandle rvIppMdmProviderFindAnyTerminal(  
    IN RvIppProviderHandle providerHndl);
```

PARAMETERS

[providerHndl](#)

Provider handle.

RETURN VALUE

Terminal handle.

SEE ALSO

[RvIppProviderHandle](#)

[RvIppTerminalHandle](#)

rvIppMdmProviderFindTerminalByNumber()

DESCRIPTION

Retrieves the first terminal handle based on the E.164 number.

SYNTAX

```
RvIppTerminalHandle rvIppMdmProviderFindTerminalByNumber(  
    IN RvIppProviderHandle    providerHndl,  
    IN const RvChar*          phoneNumber);
```

PARAMETERS

providerHndl

Provider handle.

RETURN VALUE

Terminal handle.

SEE ALSO

[RvIppProviderHandle](#)

[RvIppTerminalHandle](#)

[rvIppMdmProviderFindTerminalByTermId\(\)](#)

[rvIppMdmProviderFindTerminalByNumber\(\)](#)

MDM CONTROL EXTENSION TERMINAL API

This section includes:

- `rvIppMdmTerminalGetProvider()`
- `rvIppMdmTerminalGetId()`
- `rvIppMdmTerminalGetType()`
- `rvIppMdmTerminalGetLastEvent()`
- `rvIppMdmTerminalGetMediaCaps()`
- `rvIppMdmTerminalGetMediaStream()`
- `rvIppMdmTerminalIsFirstDigit()`
- `rvIppMdmTerminalGetLastDigit()`
- `rvIppMdmTerminalGetPhoneNumber()`
- `rvIppMdmTerminalGetActiveAudioType()`
- `rvIppMdmTerminalGetActiveAudioTerm()`
- `rvIppMdmTerminalStopSignals()`
- `rvIppMdmTerminalStartUserSignalUI()`
- `rvIppMdmTerminalStartUserSignalAT()`
- `rvIppMdmTerminalStartDialToneSignal()`
- `rvIppMdmTerminalStartRingingSignal()`
- `rvIppMdmTerminalStopRingingSignal()`
- `rvIppMdmTerminalStartRingbackSignal()`
- `rvIppMdmTerminalStartCallWaitingSignal()`
- `rvIppMdmTerminalStartCallWaitingCallerSignal()`
- `rvIppMdmTerminalStartBusySignal()`
- `rvIppMdmTerminalStartWarningSignal()`
- `rvIppMdmTerminalStartDTMFTone()`
- `rvIppMdmTerminalStopDTMFTone()`
- `rvIppMdmTerminalSetHoldInd()`
- `rvIppMdmTerminalSetLineInd()`
- `rvIppMdmTerminalSendLineActive()`
- `rvIppMdmTerminalSetDisplay()`
- `rvIppMdmTerminalClearDisplay()`
- `rvIppMdmTerminalSendCallerId()`
- `rvIppMdmTerminalSetWaitForDigits()`

- `rvIppMdmTerminalGetDialString()`
- `rvIppMdmTerminalResetDialString()`
- `rvIppMdmTerminalIsOutOfBandDtmfEnabled()`
- `rvIppMdmTerminalIsDtmfPlayEnabled()`
- `rvIppMdmTerminalGetMaxConnections()`
- `rvIppMdmTerminalGetNumActiveConnections()`
- `rvIppMdmTerminalGetActiveConnection()`
- `rvIppMdmTerminalGetCurDisplayRow()`
- `rvIppMdmTerminalGetCurDisplayColumn()`
- `rvIppMdmTerminalOtherHeldConnExist()`
- `rvIppMdmTerminalGetHoldConn()`
- `rvIppMdmTerminalSetState()`
- `rvIppMdmTerminalGetState()`
- `rvIppMdmTerminalGetMdmTerm()`
- `rvIppMdmTerminalGetHandle`

rvIppMdmTerminalGetProvider()

DESCRIPTION

Retrieves the provider to which this terminal belongs.

SYNTAX

```
RvIppProviderHandle rvIppMdmTerminalGetProvider(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Provider handle.

SEE ALSO

[RvIppProviderHandle](#)

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetId()

DESCRIPTION

Retrieves the terminal ID.

SYNTAX

```
RvBool rvIppMdmTerminalGetId(  
    IN    RvIppTerminalHandle    terminalHndl,  
    INOUT RvChar*                 termId,  
    IN    RvSize_t               termIdLen);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[termId](#)

Place holder for the terminal ID.

[termIdLen](#)

Length of the termId parameter, in bytes.

OUTPUT PARAMETERS

[termId](#)

The terminal's ID string if the return value of the function is RV_TRUE.

RETURN VALUE

RV_TRUE if the value has been retrieved, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetType()

DESCRIPTION

Retrieves the terminal type based on terminalHandle.

SYNTAX

```
RvCCTerminalType rvIppMdmTerminalGetType(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Terminal type.

SEE ALSO

[RvIppTerminalHandle](#)

[RvCCTerminalType](#)

rvIppMdmTerminalGetLastEvent()

DESCRIPTION

Retrieves the last MDM event received for this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalGetLastEvent (  
    IN    RvIppTerminalHandle    terminalHndl,  
    INOUT RvMdmEvent*            event);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[event](#)

Pointer to the Mdm Event structure where the last MDM event received will be stored.

OUTPUT PARAMETERS

[event](#)

This struct will be filled with the last MDM event that was received if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if the retrieval was successful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

[RvMdmEvent](#)

rvIppMdmTerminalGetMediaCaps()

DESCRIPTION

Retrieves the terminal's media capabilities.

SYNTAX

```
RvBool rvIppMdmTerminalGetMediaCaps (  
    IN    RvIppTerminalHandle    terminalHndl,  
    INOUT RvSdpMsg*              mediaCaps);
```

PARAMETERS

terminalHndl

Terminal handle.

mediaCaps

Pointer to the mediaCapability structure where the media capabilities will be stored.

OUTPUT PARAMETERS

mediaCaps

This struct will be filled with the terminal's media capabilities if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if the retrieval was successful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetMediaStream()

DESCRIPTION

Retrieves a specific media stream.

SYNTAX

```
RvBool rvIppMdmTerminalGetMediaStream(  
    IN      RvIppTerminalHandle    terminalHndl,  
    IN      RvUInt32                mediaStreamId,  
    INOUT   RvMdmMediaStreamInfo*   mediaStream);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[mediaStreamId](#)

ID of the media stream.

[mediaStream](#)

Pointer to the RvMdmMediaStreamInfo structure where the media stream will be stored.

OUTPUT PARAMETERS

[mediaStream](#)

This struct will be filled with the specific media stream if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if the retrieval was successful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

[RvMdmMediaStreamInfo](#)

rvIppMdmTerminalIsFirstDigit()

DESCRIPTION

Finds one digit in the current dial string, if applicable.

SYNTAX

```
RvBool rvIppMdmTerminalIsFirstDigit(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if there is one digit, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetLastDigit()

DESCRIPTION

Retrieves the last digit of the current dial string.

SYNTAX

```
RvChar rvIppMdmTerminalGetLastDigit(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

The last digit in the current dial string.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetPhoneNumber()

DESCRIPTION

Retrieves the terminal's phone number.

SYNTAX

```
RvBool rvIppMdmTerminalGetPhoneNumber(  
    IN    RvIppTerminalHandle    terminalHndl,  
    INOUT RvChar*                 phoneNumber,  
    IN    RvSize_t                phoneNumberLen);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[phoneNumber](#)

Place holder for the phone number.

[phoneNumberLen](#)

Length of the [phoneNumber](#) parameter, in bytes.

OUTPUT PARAMETERS

[phoneNumber](#)

The terminal's phone number will be copied into this buffer if this function returns `RV_TRUE`.

RETURN VALUE

`RV_TRUE` if succesful, `RV_FALSE` otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetActiveAudioType()

DESCRIPTION

Retrieves the type of the active audio termination.

SYNTAX

```
RvCCTerminalAudioType rvIppMdmTerminalGetActiveAudioType(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Audio type:

RV_CCTERMAUDIO_NONE,
RV_CCTERMAUDIO_HANDSET,
RV_CCTERMAUDIO_HANDSFREE,
RV_CCTERMAUDIO_HEADSET

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetActiveAudioTerm()

DESCRIPTION

Retrieves the handle of the active audio termination.

SYNTAX

```
RvIppTerminalHandle rvIppMdmTerminalGetActiveAudioTerm(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Current terminal handle.

RETURN VALUE

Active audio termination handle.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStopSignals()

DESCRIPTION

Stops playing signals on this terminal.

SYNTAX

```
void rvIppMdmTerminalStopSignals(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

None.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartUserSignalUI()

DESCRIPTION

Starts playing a user signal on the UI terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartUserSignalUI(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN const RvChar*          pkg,  
    IN const RvChar*          id,  
    IN RvMdmParameterList*    params);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[pkg](#)

Package name.

[id](#)

Event ID.

[params](#)

Parameter list.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

[RvMdmParameterList](#) Module

rvlppMdmTerminalStartUserSignalAT()

DESCRIPTION

Starts playing a user signal on the audio terminal.

SYNTAX

```
RvBool rvlppMdmTerminalStartUserSignalAT(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN const RvChar*          pkg,  
    IN const RvChar*          id,  
    IN RvMdmParameterList*    params);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[pkg](#)

Package name.

[id](#)

Event ID.

[params](#)

Parameters list.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

[RvMdmParameterList](#) Module

rvIppMdmTerminalStartDialToneSignal()

DESCRIPTION

Starts playing dial tone on this terminal. Note that the signal will be stopped according to the dialTone timer value.

SYNTAX

```
RvBool rvIppMdmTerminalStartDialToneSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartRingingSignal()

DESCRIPTION

Starts ringing on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartRingingSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStopRingingSignal()

DESCRIPTION

Stops ringing on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStopRingingSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartRingbackSignal()

DESCRIPTION

Starts Ringback tone on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartRingbackSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartCallWaitingSignal()

DESCRIPTION

Starts CallWaiting tone on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartCallWaitingSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartCallWaitingCallerSignal()

DESCRIPTION

Starts caller CallWaiting tone on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartCallWaitingCallerSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartBusySignal()

DESCRIPTION

Starts Busy tone on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartBusySignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartWarningSignal()

DESCRIPTION

Starts Warning tone on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartWarningSignal(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalStartDTMFTone()

DESCRIPTION

Starts DTMF tone on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalStartDTMFTone(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvChar                  digit);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[digit](#)

The digit to send.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvlppMdmTerminalStopDTMFTone()

DESCRIPTION

Stops DTMF tone on this terminal.

SYNTAX

```
RvBool rvlppMdmTerminalStopDTMFTone(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvChar                  digit);
```

PARAMETERS

terminalHndl

Terminal handle.

digit

The digit to send.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalSetHoldInd()

DESCRIPTION

Sets or clears the Hold indicator.

SYNTAX

```
RvBool rvIppMdmTerminalSetHoldInd(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvBool                  on);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[on](#)

If RV_TRUE, the indicator is lir/.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvlppMdmTerminalSetLineInd()

DESCRIPTION

Sets or clears the Line indicator.

SYNTAX

```
RvBool rvlppMdmTerminalSetLineInd(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvInt32                lineId,  
    IN RvCCTerminalMdmIndState state);
```

PARAMETERS

terminalHndl

Terminal handle.

lineId

ID of the line.

state

Indicator state:

RV_INDSTATE_ON, RV_INDSTATE_OFF,
RV_INDSTATE_BLINK, RV_INDSTATE_FAST_BLINK,
RV_INDSTATE_SLOW_BLINK

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalSendLineActive()

DESCRIPTION

Sets or clears the Line indicator.

SYNTAX

```
RvBool rvIppMdmTerminalSendLineActive(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvInt32                lineId,  
    IN RvBool                  active);
```

PARAMETERS

terminalHndl

Terminal handle.

lineId

ID of the line.

active

If RV_TRUE, the line becomes active.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalSetDisplay()

DESCRIPTION

Places text on the terminal's display.

SYNTAX

```
RvBool rvIppMdmTerminalSetDisplay(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN const RvChar*          text,  
    IN RvInt32                 row,  
    IN RvInt32                 column);
```

PARAMETERS

terminalHndl

Terminal handle.

text

Text to display.

row

Location column.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalClearDisplay()

DESCRIPTION

Clears the terminal display.

SYNTAX

```
RvBool rvIppMdmTerminalClearDisplay(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvlppMdmTerminalSendCallerId()

DESCRIPTION

Updates the caller ID.

SYNTAX

```
RvBool rvlppMdmTerminalSendCallerId(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN const RvChar*          callerName,  
    IN const RvChar*          callerNumber,  
    IN const RvChar*          address,  
    IN const RvChar*          callerId);
```

PARAMETERS

terminalHndl

Terminal handle.

callerName

String representation of caller name.

callerNumber

E.164 number.

address

IP address.

callerId

Display name.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalSetWaitForDigits()

DESCRIPTION

Instructs the terminal to start collecting digits.

SYNTAX

```
void rvIppMdmTerminalSetWaitForDigits(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

None.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetDialString()

DESCRIPTION

Retrieves the current dial string.

SYNTAX

```
RvBool rvIppMdmTerminalGetDialString(  
    IN    RvIppTerminalHandle    terminalHndl,  
    INOUT RvChar*                dialString,  
    IN    RvSize_t               dialStringLength);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

[dialString](#)

Place holder for the dial string.

[dialStringLength](#)

Length of the dialString parameter, in bytes.

OUTPUT PARAMETERS

[dialString](#)

The dial string will be copied to this buffer if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalResetDialString()

DESCRIPTION

Clears the current dial string.

SYNTAX

```
void rvIppMdmTerminalResetDialString(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

None.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalIsOutOfBandDtmfEnabled()

DESCRIPTION

Finds whether Out-of-Band DTMF is enabled.

SYNTAX

```
RvBool rvIppMdmTerminalIsOutOfBandDtmfEnabled(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

RV_TRUE—Out-of-Band DTMF enabled. RV_FALSE—Disabled.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalIsDtmfPlayEnabled()

DESCRIPTION

Finds whether DTMF Play is supported on this terminal.

SYNTAX

```
RvBool rvIppMdmTerminalIsDtmfPlayEnabled(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

terminalHndl

Terminal handle.

RETURN VALUE

RV_TRUE—DTMF Play enabled. RV_FALSE—Disabled.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetMaxConnections()

DESCRIPTION

Gets the maximum number of connections on the terminal.

SYNTAX

```
RvInt32 rvIppMdmTerminalGetMaxConnections(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Maximum number of connections.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetNumActiveConnections()

DESCRIPTION

Gets the number of active connections on the terminal.

SYNTAX

```
RvInt32 rvIppMdmTerminalGetNumActiveConnections(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Number of active onnections.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetActiveConnection()

DESCRIPTION

Gets the active connection.

SYNTAX

```
RvIppConnectionHandle rvIppMdmTerminalGetActiveConnection(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Connection handle.

SEE ALSO

[RvIppTerminalHandle](#)

[RvIppConnectionHandle](#)

rvIppMdmTerminalGetCurDisplayRow()

DESCRIPTION

Gets the current location of the cursor on the display.

SYNTAX

```
RvInt32 rvIppMdmTerminalGetCurDisplayRow(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Location in rows.

SEE ALSO

[RvIppTerminalHandle](#)

rvIppMdmTerminalGetCurDisplayColumn()

DESCRIPTION

Gets the current location of the cursor on the display.

SYNTAX

```
RvInt32 rvIppMdmTerminalGetCurDisplayColumn(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Location in columns.

SEE ALSO

[RvIppTerminalHandle](#)

rvlppMdmTerminalOtherHeldConnExist()

DESCRIPTION

Finds whether there is another held connection on this terminal.

SYNTAX

```
RvBool rvlppMdmTerminalOtherHeldConnExist (  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvIppConnectionHandle currConnHndl);
```

PARAMETERS

terminalHndl

Terminal handle.

currConnHndl

Current connection handle.

RETURN VALUE

RV_TRUE if succesful, RV_FALSE otherwise.

SEE ALSO

[RvIppTerminalHandle](#)

[RvIppConnectionHandle](#)

rvIppMdmTerminalGetHoldConn()

DESCRIPTION

Gets held connection on this terminal.

SYNTAX

```
RvIppConnectionHandle rvIppMdmGetHoldConn(  
    IN RvIppTerminalHandle    terminalHndl,  
    OUT RvInt32*               lineId);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

OUTPUT PARAMETERS

[LineId](#)

Line ID of the held connection.

RETURN VALUE

Held connection.

SEE ALSO

[RvIppTerminalHandle](#)

[RvIppConnectionHandle](#)

rvlppMdmTerminalSetState()

DESCRIPTION

Sets terminal state.

SYNTAX

```
void rvlppMdmTerminalSetState(  
    IN RvIppTerminalHandle    terminalHndl,  
    IN RvCCTerminalState      state);
```

PARAMETERS

terminalHndl

Terminal handle.

state

Terminal state.

RETURN VALUE

None.

SEE ALSO

[RvCCTerminalState](#)

rvIppMdmTerminalGetState()

DESCRIPTION

Gets held connection on this terminal.

SYNTAX

```
RvCCTerminalState rvIppMdmTerminalGetState(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

Terminal state.

SEE ALSO

[RvIppTerminalHandle](#)

[RvCCTerminalState](#)

rvIppMdmTerminalGetMdmTerm()

DESCRIPTION

Returns a pointer to the MDM termination object.

SYNTAX

```
RvMdmTerm* rvIppMdmTerminalGetMdmTerm(  
    IN RvIppTerminalHandle terminalHndl);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

RETURN VALUE

A pointer to the MDM termination object.

SEE ALSO

[RvIppTerminalHandle](#)

[RvMdmTerm Module](#)

rvIppMdmTerminalGetHandle

DESCRIPTION

Gets the terminal's handle from the MDM termination.

SYNTAX

```
RvIppTerminalHandle rvIppMdmTerminalGetHandle(  
    IN RvMdmTerm*    mdmTerm);
```

PARAMETERS

[mdmTerm](#)

MDM termination handle.

RETURN VALUE

Returns a pointer to the terminal's handle, or NULL if it fails.

SEE ALSO

[RvIppTerminalHandle](#)

MDM CONTROL EXTENSION CONNECTION API

This section includes:

- `rvIppMdmConnGetTerminal()`
- `rvIppMdmConnGetLineId()`
- `rvIppMdmConnGetState()`
- `rvIppMdmConnGetTermState()`
- `rvIppMdmConnGetMediaState()`
- `rvIppMdmConnGetType()`
- `rvIppMdmConnGetConnectParty()`
- `rvIppMdmConnGetLocalMedia()`
- `rvIppMdmConnGetMediaCaps()`
- `rvIppMdmConnSetUserData()`
- `rvIppMdmConnGetUserData()`
- `rvIppMdmConnGetCallerName()`
- `rvIppMdmConnGetCallerNumber()`
- `rvIppMdmConnGetCallerNumberByIndex()`
- `rvIppMdmConnGetCallerAddress()`
- `rvIppMdmConnGetCallerId()`
- `rvIppMdmConnGetRemotePresentationInfo()`
- `rvIppMdmConnGetCallState()`

rvIppMdmConnGetTerminal()

DESCRIPTION

Gets a handle to the terminal object associated with the connection.

SYNTAX

```
RvIppTerminalHandle rvIppMdmConnGetTerminal(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

[connHndl](#)

A handle to the connection.

RETURN VALUE

Returns a handle to the terminal object associated with the connection.

SEE ALSO

[RvIppConnectionHandle](#)

[RvIppTerminalHandle](#)

rvIppMdmConnGetLineId()

DESCRIPTION

Gets the ID of the line on which the call was created.

SYNTAX

```
RvInt32 rvIppMdmConnGetLineId(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

connHndl

Connection handle.

RETURN VALUE

Returns the ID of the line on which the call was created.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetState()

DESCRIPTION

Returns the connection state.

SYNTAX

```
RvCCConnState rvIppMdmConnGetState(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

[connHndl](#)

A handle to the connection.

RETURN VALUE

Returns the state of the connection.

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCConnState](#)

rvIppMdmConnGetTermState()

DESCRIPTION

Gets the termination state of the connection.

SYNTAX

```
RvCCTermConnState rvIppMdmConnGetTermState(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

connHndl

A handle to the connection.

RETURN VALUE

Returns the termination state of the connection.

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCTermConnState](#)

rvIppMdmConnGetMediaState()

DESCRIPTION

Gets the media state of the connection.

SYNTAX

```
RvCCMediaState rvIppMdmConnGetMediaState(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

[connHndl](#)

A handle to the connection.

RETURN VALUE

Returns the media state of the connection.

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCMediaState](#)

rvIppMdmConnGetType()

DESCRIPTION

Gets the connection type.

SYNTAX

```
RvCCConnType rvIppMdmConnGetType(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

connHndl

A handle to the connection.

RETURN VALUE

Returns the connection type (MDM or network).

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCConnType](#)

rvIppMdmConnGetConnectParty()

DESCRIPTION

Gets the other party connected to this connection.

SYNTAX

```
RvIppConnectionHandle rvIppMdmConnGetConnectParty(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

[connHndl](#)

A handle to the connection.

RETURN VALUE

Returns the connection handle connected to this connection.

SEE ALSO

[RvIppConnectionHandle](#)

rvlppMdmConnGetLocalMedia()

DESCRIPTION

Gets the media descriptor, which includes the parameters of the local media.

SYNTAX

```
RvBool rvlppMdmConnGetLocalMedia(  
    IN    RvIppConnectionHandle    connHndl,  
    INOUT RvSdpMsg*                localMedia);
```

PARAMETERS

connHndl

A handle to the connection.

localMedia

A pointer to the media descriptor filled with the parameters of the local media.

OUTPUT PARAMETERS

localMedia

This struct will be filled with the local media parameters if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if local media is not empty. RV_FALSE if it is empty.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetMediaCaps()

DESCRIPTION

Gets the capabilities loaded by the user application during initialization.

SYNTAX

```
RvBool rvIppMdmConnGetMediaCaps (
    IN      RvIppConnectionHandle  connHndl ,
    INOUT   RvSdpMsg*              mediaCaps );
```

PARAMETERS

connHndl

A handle to the connection.

mediaCaps

A pointer to the media descriptor that includes all media capabilities.

OUTPUT PARAMETERS

mediaCaps

This struct will be filled with the media capabilities if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if media capabilities is not empty. RV_FALSE if it is empty.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnSetUserData()

DESCRIPTION

Sets the user data in the connection object.

SYNTAX

```
void rvIppMdmConnSetUserData(  
    IN RvIppConnectionHandle    connHndl,  
    IN void*                    userData);
```

PARAMETERS

connHndl

A handle to the connection.

userData

A pointer to the user data to be stored.

RETURN VALUE

Returns a pointer to the user data in the connection object.

SEE ALSO

[RvIppConnectionHandle](#)

[rvIppMdmConnGetUserData\(\)](#)

rvIppMdmConnGetUserData()

DESCRIPTION

Gets the user data in the connection object.

SYNTAX

```
void* rvIppMdmConnGetUserData(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

[connHndl](#)

A handle to the connection.

RETURN VALUE

Returns a pointer to the user data in the connection object.

SEE ALSO

[RvIppConnectionHandle](#)

[rvIppMdmConnSetUserData\(\)](#)

rvIppMdmConnGetCallerName()

DESCRIPTION

Gets the caller name as received from the remote party in an incoming call. Empty in an outgoing call.

SYNTAX

```
RvBool rvIppMdmConnGetCallerName(  
    IN    RvIppConnectionHandle    connHndl,  
    INOUT RvChar*                   callerName,  
    IN    RvSize_t                  callerNameLen);
```

PARAMETERS

connHndl

A handle to the connection.

callerName

A pointer to the string that will hold the caller's name.

callerNameLen

Length of the callerName parameter, in bytes.

OUTPUT PARAMETERS

callerName

The caller's name will be copied to this buffer if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if callerName is not empty. RV_FALSE if it is empty.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetCallerNumber()

DESCRIPTION

Gets the caller number as received from the remote party in an incoming call. Empty in outgoing calls.

SYNTAX

```
RvBool rvIppMdmConnGetCallerNumber (  
    IN      RvIppConnectionHandle  connHndl,  
    INOUT   RvChar*                callerNumber,  
    IN      RvSize_t                callerNumberLen);
```

PARAMETERS

connHndl

A handle to the connection.

callerNumber

A pointer to the string that will hold the caller's number.

callerNumberLen

Length of the callerNumber parameter, in bytes.

OUTPUT PARAMETERS

callerNumber

The caller's address will be copied to this buffer if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if callerNumber is not empty. RV_FALSE if it is empty.

REMARKS

If more than one number exists, the function retrieves the first E.164 number.

MDM Control Extension Connection API

`rvIppMdmConnGetCallerNumber()`

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetCallerNumberByIndex()

DESCRIPTION

Gets the caller number as received from the remote party in an incoming call. Empty in outgoing calls.

SYNTAX

```
RvBool rvIppMdmConnGetCallerNumberByIndex(  
    IN    RvIppConnectionHandle    connHndl,  
    INOUT RvChar*                   callerNumber,  
    IN    RvSize_t                   callerNumberLen,  
    IN    RvSize_t                   index);
```

PARAMETERS

connHndl

A handle to the connection.

callerNumber

A pointer to the string that will hold the caller's number.

callerNumberLen

Length of the callerNumber parameter, in bytes.

index

Index of the number to retrieve.

OUTPUT PARAMETERS

callerNumber

The caller's address will be copied to this buffer if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if callerNumber is not empty. RV_FALSE if it is empty.

MDM Control Extension Connection API

`rvIppMdmConnGetCallerNumberByIndex()`

REMARKS

If more than one number exists, the function retrieves the first E.164 number.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetCallerAddress()

DESCRIPTION

Gets the caller address as received from the remote party in an incoming call. Will be empty in outgoing calls.

SYNTAX

```
RvBool rvIppMdmConnGetCallerAddress (
    IN      RvIppConnectionHandle  connHndl,
    INOUT   RvChar*                callerAddress,
    IN      RvSize_t               callerAddressLen) ;
```

PARAMETERS

connHndl

A handle to the connection.

callerAddress

A pointer to the string that will hold the caller's address.

callerAddressLen

Length of the callerAddress parameter, in bytes.

OUTPUT PARAMETERS

callerAddress

The caller's address will be copied to this buffer if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if callerAddress is not empty. RV_FALSE if it is empty.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetCallerId()

DESCRIPTION

Gets the caller ID as received from the remote party in an incoming call. This will be empty for outgoing calls and will return RV_FALSE.

SYNTAX

```
RvBool rvIppMdmConnGetCallerId(  
    IN    RvIppConnectionHandle    connHndl,  
    INOUT RvChar*                   callerId,  
    IN    RvSize_t                  callerIdLen);
```

PARAMETERS

connHndl

A handle to the connection.

callerId

A pointer to the string that will hold the caller's ID.

callerIdLen

Length of the callerId parameter in bytes.

OUTPUT PARAMETERS

callerId

The caller's ID will be copied to this buffer if this function returns RV_TRUE.

RETURN VALUE

RV_TRUE if callerId is not empty, RV_FALSE if it is empty.

SEE ALSO

[RvIppConnectionHandle](#)

rvIppMdmConnGetRemotePresentationInfo()

DESCRIPTION

Gets presentation information (name and permission) of the remote party. This information was retrieved from incoming messages.

SYNTAX

```
RvBool rvIppMdmConnGetRemotePresentationInfo(  
    IN  RvIppConnectionHandle    connHndl,  
    OUT RvMdmTermPresentationInfo* presentationInfo);
```

PARAMETERS

[terminalHndl](#)

Terminal handle.

OUTPUT PARAMETERS

[presentationInfo](#)

A pointer to the presentation information object.

RETURN VALUE

RV_TRUE if presentation information was returned successfully. RV_FALSE if not.

REMARKS

Implemented in H.323 only.

SEE ALSO

[RvIppConnectionHandle](#)

RvMdmTermPresentationInfo Module

rvIppMdmConnGetCallState()

DESCRIPTION

Gets the call state of this connection.

SYNTAX

```
RvCCCallState rvIppMdmConnGetCallState(  
    IN RvIppConnectionHandle    connHndl);
```

PARAMETERS

connHndl

A handle to the connection.

RETURN VALUE

The call's state.

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCCallState](#)

MDM CONTROL EXTENSION TYPE DEFINITIONS

This section includes:

- RvIppMdmPreProcessEventCB()
- RvIppMdmPostProcessEventCB()
- RvIppMdmConnectionCreatedCB()
- RvIppMdmConnectionDestructedCB()
- RvIppMdmMapUserEventCB()
- RvIppMdmDisplayCB()
- rvIppMdmExtCbks

RvIppMdmPreProcessEventCB()

DESCRIPTION

Prototype for PreProcessEvent callback.

SYNTAX

```
RvCCTerminalEvent RvIppMdmPreProcessEventCB (
    IN      RvIppConnectionHandle    connHndl,
    IN      RvCCTerminalEvent        eventId,
    INOUT   RvCCEventCause*          reason) ;
```

PARAMETERS

connHndl

Connection handle.

eventId

Terminal event ID.

reason

Pointer to event reason for the event before processing it.

OUTPUT PARAMETERS

reason

Pointer to the event reason, which can be modified by the application prior to processing the event.

RETURN VALUE

None.

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCTerminalEvent](#)

RvCCEventCause

rvIppMdmRegisterExtClbks()

rvIppMdmExtClbks

RvIppMdmPostProcessEventCB()

DESCRIPTION

Prototype for PostProcessEvent callback.

SYNTAX

```
void RvIppMdmPostProcessEventCB(  
    IN RvIppConnectionHandle    connHndl,  
    IN RvCCTerminalEvent        eventId,  
    IN RvCCEventCause           reason);
```

PARAMETERS

connHndl

Connection handle.

eventId

Terminal event ID.

reason

Event reason.

RETURN VALUE

None.

SEE ALSO

[RvIppConnectionHandle](#)

[RvCCTerminalEvent](#)

[RvCCEventCause](#)

[rvIppMdmRegisterExtCbks\(\)](#)

[rvIppMdmExtCbks](#)

RvIppMdmConnectionCreatedCB()

DESCRIPTION

Prototype for Mdm Connection Created callback.

SYNTAX

```
void RvIppMdmConnectionCreatedCB(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

[connHndl](#)

Handle of new connection.

RETURN VALUE

None.

SEE ALSO

[RvIppConnectionHandle](#)
[rvIppMdmRegisterExtClbks\(\)](#)
[rvIppMdmExtClbks](#)

RvIppMdmConnectionDestructedCB()

DESCRIPTION

Prototype for Mdm Connection Destructed callback.

SYNTAX

```
void RvIppMdmConnectionDestructedCB(  
    IN RvIppConnectionHandle connHndl);
```

PARAMETERS

connHndl

Handle of the new connection.

RETURN VALUE

None.

SEE ALSO

[RvIppConnectionHandle](#)

[rvIppMdmRegisterExtClbks\(\)](#)

[rvIppMdmExtClbks](#)

RvIppMdmMapUserEventCB()

DESCRIPTION

Prototype for MapUserEvent callback. Enables the user application to use its own events.

SYNTAX

```
RvCCTerminalEvent RvIppMdmMapUserEventCB(  
    IN const RvChar*      pkg,  
    IN const RvChar*      id,  
    IN RvMdmParameterList* args,  
    IN RvChar*            key);
```

PARAMETERS

pkg

Package of the event.

id

ID of the event.

args

Arguments.

key

Key (Megaco event specification).

RETURN VALUE

RvCCTerminalEvent—Event mapped by user application.

SEE ALSO

[RvCCTerminalEvent](#)

[RvMdmParameterList Module](#)

[rvIppMdmRegisterExtCbks\(\)](#)

MDM Control Extension Type Definitions

RvIppMdmMapUserEventCB()

[rvIppMdmExtCbks](#)

RvIppMdmDisplayCB()

DESCRIPTION

Prototype for Display callback. Enables user application to control the display.

SYNTAX

```
void RvIppMdmDisplayCB(  
    IN RvIppConnectionHandle    connHndl,  
    IN RvIppTerminalHandle      terminalHndl,  
    IN RvCCTerminalEvent        event,  
    IN RvCCEventCause           cause,  
    IN void*                    displayData);
```

PARAMETERS

connHndl

Connection handle.

terminalHndl

Terminal handle.

event

Terminal event.

cause

Event cause.

OUTPUT PARAMETERS

displayData

Pointer to user-defined structure used to display.

RETURN VALUE

None.

MDM Control Extension Type Definitions

RvIppMdmDisplayCB()

SEE ALSO

[RvIppConnectionHandle](#)

[RvIppTerminalHandle](#)

[RvCCTerminalEvent](#)

[RvCCEventCause](#)

[rvIppMdmRegisterExtCbks\(\)](#)

[rvIppMdmExtCbks](#)

rvlppMdmExtCbks

DESCRIPTION

MDM extension callbacks struct. These callbacks can be used by the application to support proprietary behavior or to add additional functionality.

SYNTAX

```
typedef struct
{
    RvIppMdmDisplayCB            display;
    RvIppMdmMapUserEventCB      mapUserEvent;
    RvIppMdmPreProcessEventCB   preProcessEvent;
    RvIppMdmPostProcessEventCB  postProcessEvent;
    RvIppMdmConnectionCreatedCB connectionCreated;
    RvIppMdmConnectionDestructedCB connectionDestructed;
} RvIppMdmExtCbks;
```

PARAMETERS

display

Enables the application to control the display.

mapUserEvent

Enables the application to handle its own events.

preProcessEvent

Enables the application to intervene prior to the processing of events by the Multimedia Terminal Framework.

postProcessEvent

Enables the application to act after an event has been processed by the Multimedia Terminal Framework.

connectionCreated

Notifies the application on newly-created MDM connections.

MDM Control Extension Type Definitions

rvIppMdmExtCbks

[connectionDestroyed](#)

Notifies the application on destructed MDM connections.

RETURN VALUE

None.

SEE ALSO

[rvIppMdmRegisterExtCbks\(\)](#)

[RvIppMdmDisplayCB\(\)](#)

[RvIppMdmMapUserEventCB\(\)](#)

[RvIppMdmPreProcessEventCB\(\)](#)

[RvIppMdmPostProcessEventCB\(\)](#)

[RvIppMdmConnectionCreatedCB\(\)](#)

[RvIppMdmConnectionDestroyedCB\(\)](#)

INDEX

C

call forward functions 181, 195

E

enumerated types 203

I

lppLogEnd() 199
lppLogInit() 197
lppLogMessage() 200
lppLogReload() 198
lppLogSourceFilterElm 202

M

mdm control extension functions 353

R

rMdmTermMgrRegisterDeleteEphTermCB()
172
RvCCCallState 220
rvCCCFwGetTypeNumber() 183
RvCCConnState 212
RvCCConnType 219
RvCCEventCause 230
RvCCMediaState 218
RvCCTermConnState 216
RvCCTerminalEvent 223
RvCCTerminalState 211
RvCCTerminalType 221
RvIppAddressResolveReplyCB() 274

rvIppCfwActivateCompletedCB() 185
RvIppCfwCBs 189
RvIppCfwCfg 190
rvIppCfwDeactivateCompletedCB() 187
RvIppCfwReturnReasons 193
RvIppCfwType 192
RvIppConnectionHandle 351
RvIppMdmConnectionCreatedCB() 437
RvIppMdmConnectionDestructedCB() 438
rvIppMdmConnGetCallerAddress() 429
rvIppMdmConnGetCallerId() 430
rvIppMdmConnGetCallerName() 424
rvIppMdmConnGetCallerNumber() 425
rvIppMdmConnGetCallerNumberByIndex()
427
rvIppMdmConnGetCallState() 432
rvIppMdmConnGetConnectParty() 419
rvIppMdmConnGetLineId() 414
rvIppMdmConnGetLocalMedia() 420
rvIppMdmConnGetMediaCaps() 421
rvIppMdmConnGetMediaState() 417
rvIppMdmConnGetRemotePresentationInfo()
431
rvIppMdmConnGetState() 415
rvIppMdmConnGetTerminal() 413
rvIppMdmConnGetTermState() 416
rvIppMdmConnGetType() 418
rvIppMdmConnGetUserData() 423
rvIppMdmConnSetUserData() 422
RvIppMdmDisplayCB() 441
rvIppMdmExtCbks 443
RvIppMdmMapUserEventCB() 439
RvIppMdmPostProcessEventCB() 436

RvIppMdmPreProcessEventCB()	434	rvIppMdmTerminalOtherHeldConnExist()	406
rvIppMdmProviderFindAnyTerminal()	361	rvIppMdmTerminalResetDialString()	398
rvIppMdmProviderFindTerminalByAddress()	360	rvIppMdmTerminalSendCallerId()	394
rvIppMdmProviderFindTerminalByNumber()	362	rvIppMdmTerminalSendLineActive()	391
rvIppMdmProviderFindTerminalByTermId()	359	rvIppMdmTerminalSetDisplay()	392
rvIppMdmProviderGetDialToneDuration()	357	rvIppMdmTerminalSetHoldInd()	389
rvIppMdmProviderGetDisplayData()	358	rvIppMdmTerminalSetLineInd()	390
rvIppMdmRegisterExtClbks()	355	rvIppMdmTerminalSetState()	408
rvIppMdmTerminalClearDisplay()	393	rvIppMdmTerminalSetWaitForDigits()	396
rvIppMdmTerminalGetActiveAudioTerm()	375	rvIppMdmTerminalStartBusySignal()	385
rvIppMdmTerminalGetActiveAudioType()	374	rvIppMdmTerminalStartCallWaitingCallerSignal()	384
rvIppMdmTerminalGetActiveConnection()	403	rvIppMdmTerminalStartCallWaitingSignal()	383
rvIppMdmTerminalGetCurDisplayColumn()	405	rvIppMdmTerminalStartDialToneSignal()	379
rvIppMdmTerminalGetCurDisplayRow()	404	rvIppMdmTerminalStartDTMFTone()	387
rvIppMdmTerminalGetDialString()	397	rvIppMdmTerminalStartRingbackSignal()	382
rvIppMdmTerminalGetHandle	411	rvIppMdmTerminalStartRingingSignal()	380
rvIppMdmTerminalGetHoldConn()	407	rvIppMdmTerminalStartUserSignalAT()	378
rvIppMdmTerminalGetId()	366	rvIppMdmTerminalStartUserSignalUI()	377
rvIppMdmTerminalGetLastDigit()	372	rvIppMdmTerminalStartWarningSignal()	386
rvIppMdmTerminalGetLastEvent()	368	rvIppMdmTerminalStopDTMFTone()	388
rvIppMdmTerminalGetMaxConnections()	401	rvIppMdmTerminalStopRingingSignal()	381
rvIppMdmTerminalGetMdmTerm()	410	rvIppMdmTerminalStopSignals()	376
rvIppMdmTerminalGetMediaCaps()	369	RvIppProviderHandle	349
rvIppMdmTerminalGetMediaStream()	370	rvIppSipControlGetExtUserData()	248
rvIppMdmTerminalGetNumActiveConnections()	402	rvIppSipControlGetMdmTerminal()	323
rvIppMdmTerminalGetPhoneNumber()	373	rvIppSipControlGetOutboundProxyAddress()	247
rvIppMdmTerminalGetProvider()	365	rvIppSipControlGetRegistrarAddress()	245
rvIppMdmTerminalGetState()	409	rvIppSipControlGetStackCallbacks()	251
rvIppMdmTerminalGetType()	367	rvIppSipControlGetStackHandle()	250
rvIppMdmTerminalIsDtmfPlayEnabled()	400	rvIppSipControlGetTransportType()	249
rvIppMdmTerminalIsFirstDigit()	371	rvIppSipControlGetUserDomain()	246
rvIppMdmTerminalIsOutOfBandDtmfEnabled()	399	RvIppSipControlHandle	263
		rvIppSipControlMsgSetSdpBody()	252
		RvIppSipExtClbks()	324
		rvIppSipInitConfig()	242
		RvIppSipPhoneCfg	256
		rvIppSipPhoneConstruct()	244

RvIppSipPostCallLegCreatedIncomingCB()	331	rvMdmDigitMapConstructA()	6
RvIppSipPostCallLegCreatedOutgoingCB()	334	rvMdmDigitMapCopy()	7
RvIppSipPostMsgReceivedCB()	345	rvMdmDigitMapDestruct()	8
RvIppSipPostMsgToSendCB()	341	RvMdmDigitMapMatchType	15
RvIppSipPostStateChangedCB()	337	rvMdmDigitMapSetLongTimeout()	11
RvIppSipPreCallLegCreatedIncomingCB()	329	rvMdmDigitMapSetShortTimeout()	12
RvIppSipPreCallLegCreatedOutgoingCB()	332	rvMdmDigitMapSetStartTimeout()	13
RvIppSipPreMsgReceivedCB()	343	RvMdmDigitPosition functions	17
RvIppSipPreMsgToSendCB()	339	rvMdmDigitPositionAddEvents()	22
RvIppSipPreRegClientStateChangedCB()	347	rvMdmDigitPositionConstruct()	19
RvIppSipPreStateChangedCB()	335	rvMdmDigitPositionDestruct()	20
rvIppSipRegisterExtCbks()	241	rvMdmDigitPositionSetMultipleFlag()	23
RvIppSipRegisterStackEventsCB()	328	rvMdmDigitPositionSetTimerMode()	24
RvIppSipStackCallbacks	253	RvMdmDigitPositionTimerMode	26
RvIppSipStackConfigCB()	327	RvMdmDigitString functions	27
rvIppSipStackInitialize()	243	rvMdmDigitStringAddElement()	32
rvIppSipSystemEnd()	240	rvMdmDigitStringConstruct()	29
rvIppSipSystemInit()	239	rvMdmDigitStringDestruct()	30
rvIppSipTlsInitConfig()	281	RvMdmEvent	35
RvIppSipTlsPostConnectionAssertionCB()	285	RvMdmEventData functions	33
rvIppSipTlsRegisterExtCbks()	282	rvMdmEventGetName()	37
RvIppStunAddrData	277	rvMdmEventGetParameterList()	36
RvIppStunAddressResolveComplete()	268	rvMdmMdmParameterListConstructA()	66
RvIppStunAddressResolveStartCB()	273	RvMdmMediaStreamDescr functions	39
RvIppStunIsAddressResolveNeededCB()	272	rvMdmMediaStreamDescrGetControlParameters()	41
RvIppStunMgrCreate()	267	rvMdmMediaStreamDescrGetLocalDescr()	42
RvIppStunMgrDelete()	269	rvMdmMediaStreamDescrGetMode()	43
RvIppStunMgrGetSendMethod()	270	rvMdmMediaStreamDescrGetRemoteDescr()	44
RvIppStunMgrParam	276	rvMdmMediaStreamDescrGetReserveGroupMode()	45
RvIppTerminalHandle	350	rvMdmMediaStreamDescrGetReserveValueMode()	46
RvIppTlsBufferType	291	rvMdmMediaStreamDescrReportControlParameters()	47
RvIppTlsGetBufferCB()	284	rvMdmMediaStreamDescrReportLocalDescr()	48
RvIppTransportTlsCfg	288	rvMdmMediaStreamDescrReportRemoteDescr()	49
RvMdmDigitMap functions	3, 4, 9, 14	RvMdmPackageItem functions	51
rvMdmDigitMapAddPattern()	10	rvMdmPackageItemConstruct()	53
rvMdmDigitMapConstruct()	5		

rvMdmPackageItemConstructA()	54	rvMdmTermClassAddSupportedPkg()	127
rvMdmPackageItemConstructCopy()	55	rvMdmTermClassClearMediaCapabilities()	136
rvMdmPackageItemCopy()	56	rvMdmTermClassRegisterCreateMediaCB()	128
rvMdmPackageItemDestruct()	57	rvMdmTermClassRegisterDestroyMediaCB()	129
rvMdmPackageItemEqual()	58	rvMdmTermClassRegisterMapDialStringToAddressCB()	134
rvMdmPackageItemGetItem()	60	rvMdmTermClassRegisterMatchDialStringCB()	135
rvMdmPackageItemGetPackage()	61	rvMdmTermClassRegisterModifyMediaCB()	130
RvMdmParameterList functions	63	rvMdmTermClassRegisterModifyMediaCompletedCB()	139
rvMdmParameterListConstruct()	65	rvMdmTermClassRegisterPlaySignalCB()	131
rvMdmParameterListConstructCopy()	67	rvMdmTermClassRegisterRegisterPhysTermCompletedCB()	137
rvMdmParameterListCopy()	68	rvMdmTermClassRegisterStartSignalCB()	132
rvMdmParameterListDestruct()	69	rvMdmTermClassRegisterStopSignalCB()	133
rvMdmParameterListForEach()	71	rvMdmTermClassRegisterUnregisterTermCompletedCB()	138
rvMdmParameterListGet()	72	RvMdmTermCreateMediaCB()	300
rvMdmParameterListGet2()	73	RvMdmTermDefaultProperties functions	141
rvMdmParameterListIsEmpty()	74	rvMdmTermDefaultPropertiesConstruct()	143
rvMdmParameterListSet()	75	rvMdmTermDefaultPropertiesDestruct()	144
RvMdmParameterValue functions	77	rvMdmTermDefaultPropertiesSetDigitMap()	146
rvMdmParameterValueAddToList()	86	RvMdmTermDefaultPropertiesSetPassword()	147
rvMdmParameterValueConstruct()	79	RvMdmTermDefaultPropertiesSetUsername()	148
rvMdmParameterValueConstructA()	80	RvMdmTermDestroyMediaCB()	302
rvMdmParameterValueConstructCopy()	81	rvMdmTermGetId()	111
rvMdmParameterValueConstructList()	82	rvMdmTermGetType()	112
rvMdmParameterValueConstructListA()	83	rvMdmTermGetUserData()	113
rvMdmParameterValueDestruct()	84	RvMdmTermMapDialStringToAddressCB()	297
rvMdmParameterValueGetListSize()	87	RvMdmTermMatchDialStringCB()	299
rvMdmParameterValueGetListValue()	88	RvMdmTermMgr functions	149
rvMdmParameterValueGetType()	89	RvMdmTermMgrConnectCB()	311
rvMdmParameterValueGetValue()	90		
RvMdmRelation	210		
RvMdmSignal functions	91		
rvMdmSignalGetArguments()	95		
rvMdmSignalGetId()	93		
rvMdmSignalGetPkg()	94		
RvMdmSignalType	205		
RvMdmStreamDirection	206		
RvMdmStreamMode	207		
RvMdmTerm functions	109		
RvMdmTermClass functions	123		
rvMdmTermClassAddMediaCapabilities()	125		

- rvMdmTermMgrCreateTermClass() 176
- RvMdmTermMgrDeleteEphTermCB() 313
- rvMdmTermMgrDestruct() 151
- RvMdmTermMgrDisconnectCB() 314
- rvMdmTermMgrFindTermination() 166
- rvMdmTermMgrForEachPhysicalTerm() 167
- rvMdmTermMgrGetIdleTermination() 168
- rvMdmTermMgrGetPackage() 169
- rvMdmTermMgrGetUserData() 170
- rvMdmTermMgrRegisterAllTermsToNetwork() 152
- rvMdmTermMgrRegisterConnectCB() 171
- rvMdmTermMgrRegisterDisconnectCB() 173
- rvMdmTermMgrRegisterEphemeralTermination() 155
- rvMdmTermMgrRegisterPhysicalTermination() 157
- rvMdmTermMgrRegisterPhysicalTerminationAsync() 162
- rvMdmTermMgrRegisterSelectTermCB() 174
- RvMdmTermMgrRegisterTermToNetwork() 153
- RvMdmTermMgrSelectTerminationCB() 316
- rvMdmTermMgrSetUserData() 175
- rvMdmTermMgrStart() 159
- rvMdmTermMgrStop() 160
- rvMdmTermMgrUnregisterTermFromNetwork() 154
- rvMdmTermMgrUnregisterTermination() 161
- rvMdmTermModifyMedia() 120
- RvMdmTermModifyMediaCB() 303
- RvMdmTermModifyMediaCompletedCB() 317
- RvMdmTermPlaySignalCB() 305
- rvMdmTermProcessEvent() 114
- rvMdmTermPropertiesSetPhoneNumbers() 118
- rvMdmTermPropertiesSetPresentationInfo() 117
- RvMdmTermRegisterPhysTermCompletedCB() 178
- RvMdmTermSetStateCB() 307
- rvMdmTermSetUserData() 116
- RvMdmTermStartSignalCB() 308

- RvMdmTermStopSignalCB() 310
- RvMdmTermType 209
- RvMdmTermUnregisterTermCompletedCB() 179
- RvMdmTermUnregisterTermFromNetworkCompletedCB() 180
- rvMtfTerminalMakeCall() 119

S

- SIP control extension functions 321
- SIP control functions 237
- STUN 265

T

- TLS 279

U

- user callbacks 295